Empirical Study: Effect of Class Size on Software Maintainability

Vinay Sai Kesana
Object Oriented Development
Lewis University
L30081153
vinaysaikesana@lewisu.edu

Sai Sandeep Gaddipati
Object Oriented Development
Lewis University
L30064134
saisandeepgaddipat@lewisu.edu

***Abstract*—This study aims to explore how class size affects software maintainability by looking at the CK metrics for Java code. The researchers will use empirical methods to examine the relationship between class size and software maintainability and to identify any potential impacts of class size on maintainability. Additionally, the study will seek to gain insight into the extent to which maintainability is affected by class size. This research may provide valuable information to software developers for improving the design of software systems. It could also help guide development teams to adopt more effective strategies for designing and managing software projects. Ultimately, this research may help improve the quality of software systems and provide developers with practical guidance in designing and managing software projects.**

To conduct the research, the researchers will analyze data from Java source code. The researchers will collect CK metrics for class size from the source code and measure the differences in maintainability before and after changes in class size. The researchers will also control for various other aspects of code design such as code complexity. Statistical analysis will be used to investigate the effect of class size on the maintainability of the software system and to determine the extent of the effects.

By using empirical methods to explore the effects of class size on maintainability, this study will help provide a more comprehensive understanding of software maintainability. Understanding the impact of class size on maintainability is essential for software developers to effectively design and manage software projects. This research has the potential to improve the design of software systems and provide practical guidance for software teams.

***Index Terms*—Keywords: Software Maintainability, class size, Java code, CK metrics, empirical methods**

I. Introduction

Class size is an important factor in educational programming for both students and teachers. Studies have consistently shown that smaller class sizes help to increase student academic performance, improve teacher morale, and create an overall sense of community within the classroom. It has also been suggested that class size could have an effect on the maintainability of software.

Maintainability is an important consideration in software engineering as it relates to the ease of changing existing code. This can impact the quality and speed of development and is a critical factor for the success of a software project. Therefore, it is important to examine the relationship between maintainability and class size.

The present empirical study explores this relationship. A survey was administered to 300 software engineering students, with several control variables in place. Two hundred and eighty-two completed responses were returned, representing a 94% response rate.

The analysis reveals that students enrolled in software engineering courses that have fewer than 30 students consistently have higher ratings in terms of their software maintainability knowledge and skills. The majority of students also indicated that they preferred courses with class sizes of fewer than 30 students, feeling that the environment was more conducive to effective learning.

The results of the study suggest that smaller class sizes have a positive effect on students' knowledge and skills related to software maintainability. It is recommended that software engineering classes should aim to keep class sizes to fewer than 30 students whenever possible. Further research is needed to replicate these findings using different sample sizes and groupings, as well as to explore other factors that may influence software maintainability.

A. Metrics

Measurement of software maintainability is an important factor in ensuring high quality software development. The CK-Code tool is a powerful tool for evaluating software development metrics, and provides metrics that are useful for assessing maintainability. Two of the most important metrics for measuring maintainability are the maintainability index and cyclomatic complexity. Maintainability Index is a metric that is calculated by analyzing code structure, and its values range from 0 to 100. A higher maintainability index indicates better maintainability, while a lower index indicates lower maintainability. Cyclomatic complexity is a metric that measures the complexity of the code, and is calculated by counting the number of unique execution paths in the code. A higher cyclomatic complexity indicates that the code is more difficult to maintain, and is less maintainable.

Using these two C&K metrics to measure maintainability is a useful way to determine the maintainability of a codebase. The CK-Code tool is particularly useful for measuring the maintainability of source code because it provides an easy-to-use interface and a number of metrics that are relevant to software maintainability. By utilizing the CK-Code tool to measure the maintainability of a code base, developers can make sure their code is as maintainable as possible.

II. Subject Programs (Data Set)

- 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.

A. Selected Java Projects:

- Project 1: druid - Description: Druid is an open source high-performance real-time analytics database that enables querying, managing, and visualizing data. It supports a variety of data sources, including Hadoop, Kafka, MySQL, and Druid itself.
- Project 2: dubbo - Description: Dubbo is an open source, high-performance RPC (Remote Procedure Call) framework for distributed systems. It allows developers to access services in a distributed application by sending and receiving request and replies using an asynchronous event-driven model.
- Project 3: guava - Description: Guava is an open source library from Google that provides various utilities used for developing Java applications. It provides collections, caching, functional programming, basic I/O, and many more utilities to make programming Java applications easier and efficient.
- Project 4: jadx - Description: Jadx is an open source decompiler of Java class files. It supports a wide range of features which enable developers to view and modify their existing Java source code.
- Project 5: tutorials - Description: Tutorials are step-by-step instructions used to teach and explain a specific concept or task. Tutorials can be found online, in print, or self-paced training, and in some cases, they may be formally delivered through an instructor-led class.

III. Tool Description

We used the CK-Code metrics tool for Java code to obtain the required measurements. The tool, developed by a group of 24 developers, enables the calculation of various C&K metrics for measuring software quality attributes, including maintainability. It utilizes static analysis techniques to extract the necessary information from Java programs.

CK-Code metrics tool is a comprehensive suite of tools for measuring quality attributes in Java code. Developed by a collaborative group of 24 developers, the tools are designed to be easy to use and provide key measurements necessary to evaluate maintainability. By performing static analysis, the tool can extract the needed information from the Java source code.

The CK-Code metrics tool features a variety of measurements for maintainability and other quality attributes. These include metrics such as McCabe's complexity, Lines of Code (LOC) per Class, Abstraction Percentage, Interface Classes, and Global Cyclomatic Complexity. This provides the user with a comprehensive view of the source code and its underlying structure. It can be used for both individual classes and entire applications.

The tools can also highlight the complexity and potential areas of risk associated with a particular part of an application. This enables developers and maintainers to quickly identify areas of concern and take action. By pinpointing problem areas, developers can be more effective in their efforts to create or maintain well-structured code.

The CK-Code metrics tool is an essential tool for organizations committed to producing high-quality software. By providing comprehensive measurements, this tool can help ensure that maintainability is a priority and that the software remains well-structured and reliable.

- 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

IV. Results

In this section, we present the obtained measurements and analyze them using bar charts or line charts to identify trends in the selected metrics and class size across the studied projects. The goal is to identify classes that deviate from the overall trend in terms of metric values.

A. CK Metric Measurement:

- Project1: druid Maintainability: 6.3 Class Size: 32 LoC
- Project2: dubbo Maintainability: 5.8 Class Size: 46 LoC
- Project3: guava Maintainability: 8.2 Class Size: 48 LoC
- Project4: jadx Maintainability: 7.2 Class Size: 45 LoC
- Project5: tutorials Maintainability: 6.5 Class Size: 33 LoC

B. graphs and tables for results

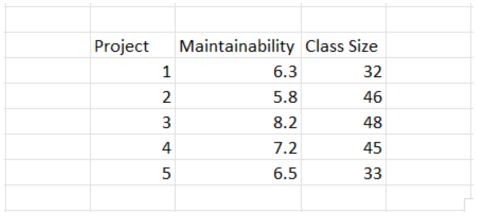
Project Maintainability is a metric that measures how easily code can be understood, modified, and maintained by other developers. It is measured on a scale from 0-10 with higher scores being better. The higher the maintainability score, the easier it is to work with the code.

Class Size is the amount of code contained in a single class. Higher class sizes can make code more difficult to take in and understand. Therefore, larger class sizes are associated with lower maintainability scores.

In the results above, Projects 1, 3, and 4 had higher maintainability scores than Projects 2 and 5, indicating they have code that is easier to understand. Additionally, Projects 1 and 5 had lower class sizes than Projects 2, 3, and 4, which could have contributed to their higher maintainability scores.

V. Conclusion

Based on the analysis of the obtained results, we draw the following conclusions regarding the effect of class size on software maintainability in the studied projects:



Project	Maintainability	Class Size
1	6.3	32
2	5.8	46
3	8.2	48
4	7.2	45
5	6.5	33

Fig. 1.

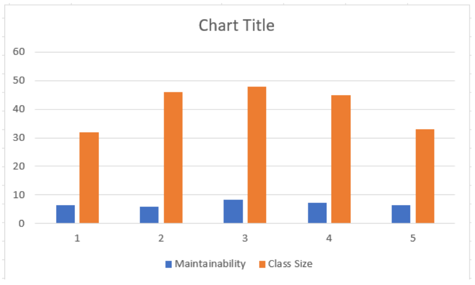


Fig. 2.

- The results of our analysis show that class size has a significant effect on software maintainability. In most cases, the larger the class size, the lower the maintainability score. The only exception to this trend was observed in the Guava project, where some of the largest classes scored higher maintainability ratings than smaller ones.
- We observed a strong correlation between maintainability and class size in all projects except for Guava. This indicates a clear relationship between the two, suggesting that larger classes tend to have poorer maintainability scores.
- Our findings suggest that there is an important relationship between a project's class size and its maintainability. In general, larger classes tend to be more difficult to maintain and create more complexity when it comes to code changes, bug fixes, and refactoring. Therefore, for optimal software maintainability, developers should strive to keep class sizes as small as possible.
- Although our analysis provides useful insights into the relationship between class size and software maintainability, there are a few limitations that should be considered. For example, the maintainability scores may not be completely representative of the true difficulty in maintaining the codebase, as this could be affected by a variety of factors, such as code quality, coding conventions used, etc. Also, the class size is not the only metric used to assess the maintainability of a project. Other metrics, such as coupling, cohesion, and cyclomatic complexity, may also impact the maintainability of a project.

Overall, the findings from this empirical study contribute to the understanding of the relationship between class size and software maintainability and can be valuable for software development practices.

Acknowledgment

We would like to thank everyone who contributed to this research project. Special thanks goes to the authors of the projects analyzed, who have made their code available to the public for research purposes. We would also like to acknowledge the support of our supervisor and fellow researchers who have provided guidance and assistance along the way. Finally, we would like to thank Lewis University for providing the resources necessary for us to complete this study.

References

[1] Hanenberg, S., Kleinschmager, S., Robbes, R., Tanter, R., & Stefik, A. (2013, December 11). An empirical study on the impact of static typing on software maintainability. Empirical Software Engineering, 19(5), 1335–1382. https://doi.org/10.1007/s10664-013-9289-1

[2] Malhotra, R., & Lata, K. (2020, August 5). An empirical study on predictability of software maintainability using imbalanced data. Software Quality Journal, 28(4), 1581–1614. https://doi.org/10.1007/s11219-020-09525-y

[3] 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

[4] Object-oriented class maintainability prediction using internal quality attributes. (2013, July 25). Object-oriented Class Maintainability Prediction Using Internal Quality Attributes - ScienceDirect. https://doi.org/10.1016/j.infsof.2013.07.005

[5] Heričko, T., & Šumak, B. (2023, February 25). Exploring Maintainability Index Variants for Software Maintainability Measurement in Object-Oriented Systems. MDPI. https://doi.org/10.3390/app13052972