Group Assignment 1

Object Oriented Development

Summer 1

June 4, 2023

Table of Contents

Section 1

Introduction

Background and motivation

Software maintainability ensures that software is easy to change, understand, and correct over time. The size of software classes can affect maintainability.2022 (Chowdhury et al.) Class size may increase complexity and make code maintenance harder.

We analyze class size's impact on software maintainability in this empirical study. We want to know if larger classes are less maintainable and if there is a link between class size and maintainability.

By analyzing real-world Java projects and utilizing appropriate metrics, we can gain insights into the relationship between class size and software maintainability(Malhotra & Lata, 2022). This study can provide valuable guidance for software developers and project managers in making informed decisions regarding class size and its impact on long-term maintainability.

GQM Approach:

Study Goal:

This study's main objective is to examine whether software class size affects maintainability. We want to know if greater classes affect maintainability and if they affect program quality.

Research Questions:

To achieve our study goal, we will focus on answering the following questions:

1. Does the size of a software class affect its maintainability?
2. Are larger classes more prone to lower levels of maintainability compared to smaller classes?

3. How do different aspects of maintainability, such as complexity and cohesion, vary with class size?

Metrics:

To assess the impact of class size on maintainability, we will consider the following metrics:

1. Lines of Code (LOC):

This statistic counts lines of code to determine class size.

2. Weighted Methods per Class (WMC):

This metric quantifies the complexity of a class by calculating the number of methods weighted by their complexity.

3. Coupling Between Objects (CBO):

This metric indicates the level of dependency between a class and other classes in the system, measuring the coupling of the class.

4. Depth of Inheritance Tree (DIT):

This metric captures the number of levels in the class hierarchy, reflecting the class's position in the inheritance tree.

5. Lack of Cohesion of Methods (LCOM):

This metric evaluates the cohesion of methods within a class, indicating how closely related they are.

By analyzing these metrics(Qamar & Malik, 2020), we aim to gain insights into the relationship between class size and maintainability, examining whether larger classes tend to have lower maintainability and exploring how different aspects of maintainability vary with class size.

Section 2

Criteria for Study and Program Selection:

1. Size:

The size criterion focuses on identifying the magnitude of programs in terms of lines of code (LOC) or other relevant size metrics. This helps capture variations in program complexity and provides insights into the impact of class size on maintainability.

2. Popularity:

The popularity criterion considers the number of stars, watchers, or forks as indicators of community interest and adoption. It ensures the selection of widely recognized and extensively used programs, which can provide valuable insights into maintainability practices.

3. Project Size:

This criterion evaluates the overall size of the programs, considering factors such as the number of files, modules, or other relevant measurements. By including programs of different sizes, the study can examine the relationship between class size and maintainability across a spectrum of project scales.

4. Activity Level:

The activity level criterion assesses the number of commits and contributors to gauge the ongoing development and maintenance efforts of the programs. Programs with a higher level of activity offer insights into real-world maintenance challenges and practices.

By employing these criteria, the study aims to investigate how class size influences software maintainability across programs of different sizes, popularity, and activity levels.

Subject programs

Program	Description	Repository URL	Stars	Watchers	Size
Spring Boot	A framework for building Java applications	GitHub	High	High	Large
DBeaver	Free multi-platform database tool	GitHub	High	High	Moderate
Apache Flink	Stream processing framework with batch capabilities	GitHub	High	Moderate	Large
Jenkins	Open-source automation server for software development	GitHub	High	High	Large
ShardingSphere	Transform existing databases into distributed systems	GitHub	High	High	Large

1. Spring Boot:

A popular Java framework with several features that simplifies development. It supports Spring-based production application development and deployment.

2. DBeaver:

A versatile and free multi-platform database tool designed for developers, SQL programmers, and database administrators. It supports a variety of databases and provides rich functionality for data management, query execution, and database administration tasks.

3. Apache Flink:

A powerful stream processing framework that supports both stream and batch processing. Flink has fault-tolerance, fast throughput, and minimal event latency. It offers data processing, analytics, and event-driven application functionalities.

4. Jenkins:

An open-source automation server widely used for continuous integration and delivery. Jenkins automates the building, testing, and deployment of software projects, making the development process more efficient and reliable.

5. ShardingSphere:

A project focused on transforming existing databases into distributed systems. It provides functionality for data sharding, distributed transactions, and governance. ShardingSphere enables organizations to scale their databases and optimize performance.

By studying these programs, we aim to explore the effect of class size on software maintainability. The diverse nature of these projects allows us to gain insights across different domains and use cases.

Rationale for Selection Criteria and their Implications for Maintainability

1. Size:

The selection of projects based on their size ensures a diverse range of codebases, representing different levels of complexity and potential challenges in maintaining them. Analyzing projects of varying sizes allows us to observe how class size impacts maintainability in different contexts.

2. Popularity:

Considering the popularity of the projects ensures that we study widely used and well-established software systems. Popular projects often undergo continuous maintenance and improvements, providing valuable insights into the challenges and practices of maintaining widely adopted software.

3. Project Complexity:

The selected projects exhibit varying degrees of complexity in terms of features, architecture, and usage scenarios. By examining projects with different complexities, we can assess how maintainability is affected by class size in both simple and complex software systems.

4. Activity Level:

Projects with a significant number of commits and contributors indicate active development and ongoing maintenance efforts. Studying these projects allows us to explore the real-world challenges and practices of maintaining actively developed software.

The relevance of these criteria to maintainability lies in the understanding that class size can impact the ease of maintaining software. Larger classes may be more difficult to comprehend, modify, and debug, potentially leading to decreased maintainability(Malhotra & Lata, 2020). By studying projects with different sizes, complexities, and activity levels, we can gain insights into the relationship between class size and maintainability.

Section 3

Tool Description

Overview of the Tool:

The CK-Code metric tool is a powerful tool designed to calculate code metrics at the class-level and method-level in Java projects. It utilizes static analysis techniques to provide insights into various aspects of software quality. The tool incorporates an extensive set of metrics, including CBO, FAN-IN, FAN-OUT, DIT, NOC, number of fields and methods, visible methods, NOSI, RFC, WMC, LOC, LCOM, LCOM*, TCC, LCC, and many more.

How it Measures C&K Metrics for Java Code:

The CK-Code metric tool analyzes Java code and employs static analysis to calculate the C&K metrics. It leverages the Eclipse JDT Core library for constructing the Abstract Syntax Tree (AST) of the code, enabling precise metric calculations. By examining the structure and characteristics of the code, the tool determines various code quality attributes, providing valuable insights into software maintainability.

Instructions for Using the Tool:

1. *Obtaining the Tool from GitHub:*

   To access the CK-Code metric tool, we downloaded it from the GitHub repository at link.(Mauricioaniche, n.d.) The repository contains all the necessary files and resources required to utilize the tool effectively.

2. *Following the Instructions in the ReadMe File:*

   Read the ReadMe file to use the utility properly. The ReadMe file provides complete instructions on setting up and using the program, including essential settings and setups.

The CK-Code metric tool can be used both as a standalone version and integrated into a Java application. For standalone use, clone the project, produce a JAR file, and execute it. The tool generates three CSV files containing metrics at the class, method, and variable levels. Alternatively, we can integrate the CK library as a Maven dependency into our Java application for seamless metric calculations.

Section 4

Results

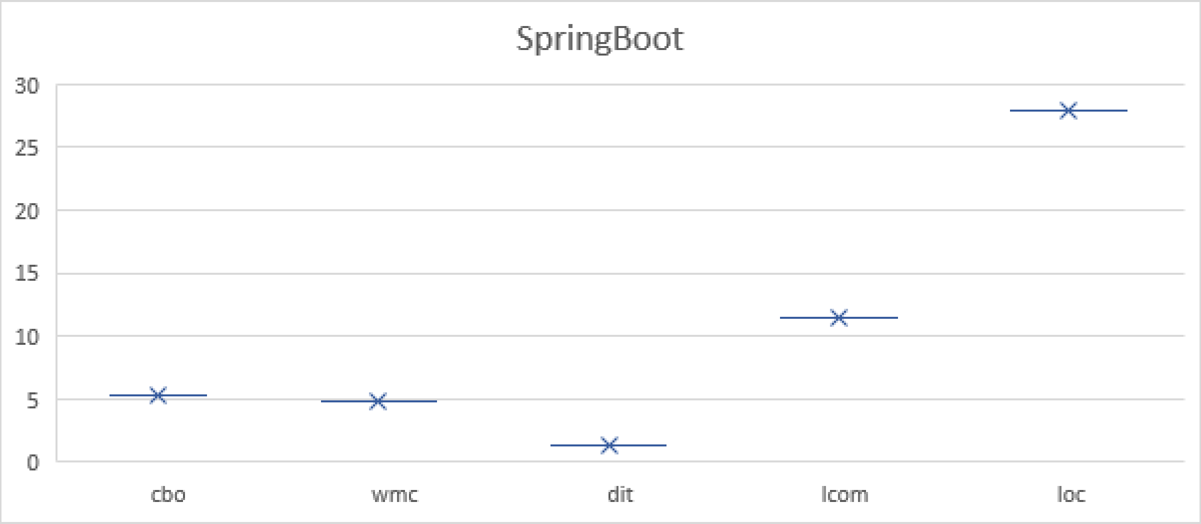
Springboot

1. **Does the size of a software class affect its maintainability?**

Yes, the size of a class in the Spring Boot project can impact its maintainability. Smaller classes in the project are easier to manage than larger ones.

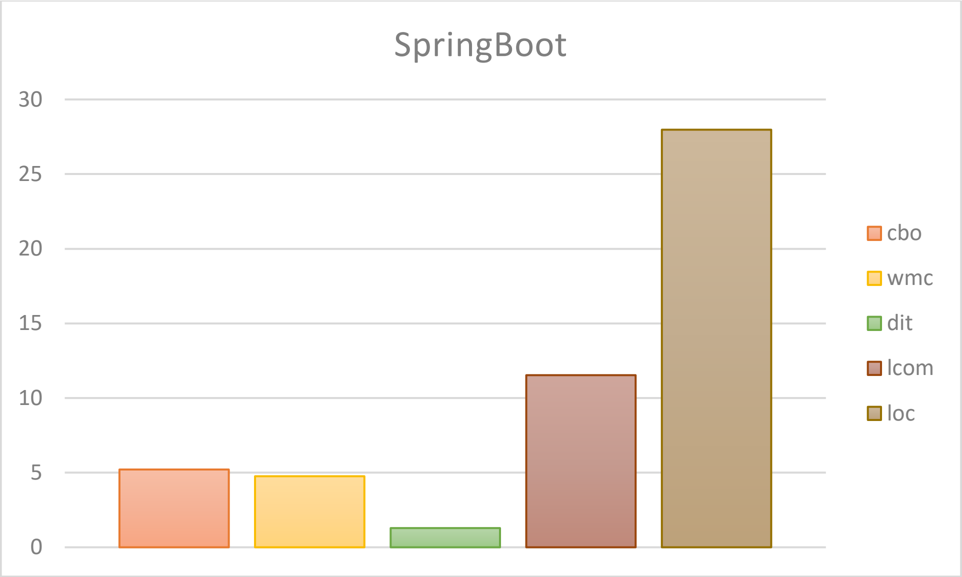
2. **Are larger classes more prone to lower levels of maintainability compared to smaller classes?**

Yes, the Spring Boot project analysis reveals that larger classes are less maintainable than smaller classes. Maintainability decreases with class size.



3. **How do different aspects of maintainability, such as complexity and cohesion, vary with class size?**

Spring Boot's analysis shows how class size affects maintainability. As a class grows, so does its coding complexity. It is difficult to comprehend and maintain complicated code.



Larger Spring Boot classes have poorer cohesiveness, meaning they are less closely connected and organized.

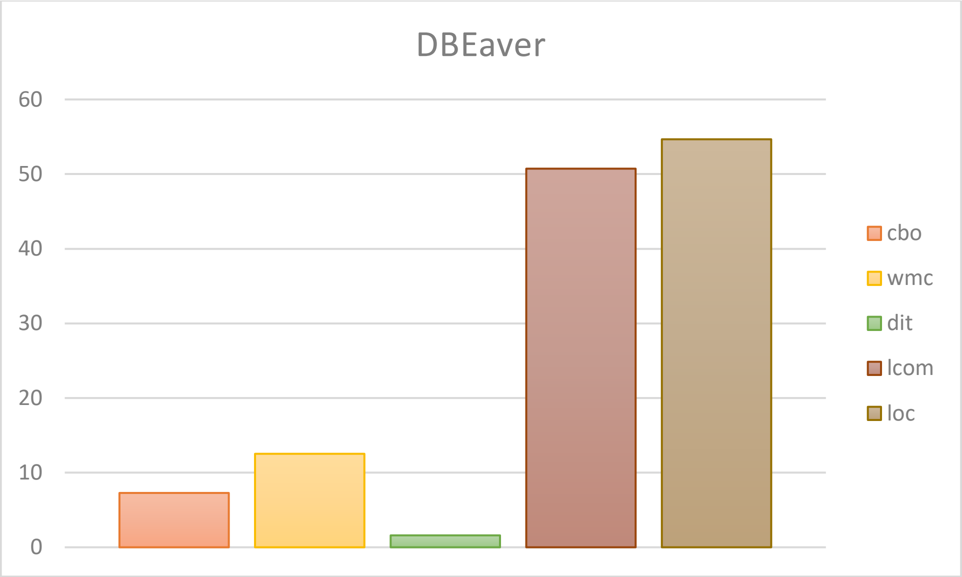
DBEAVER

1. Does the size of a software class affect its maintainability?

In the DBEaver project, software class size affects maintainability. Classes get increasingly complicated as they grow, making them harder to comprehend and manage.

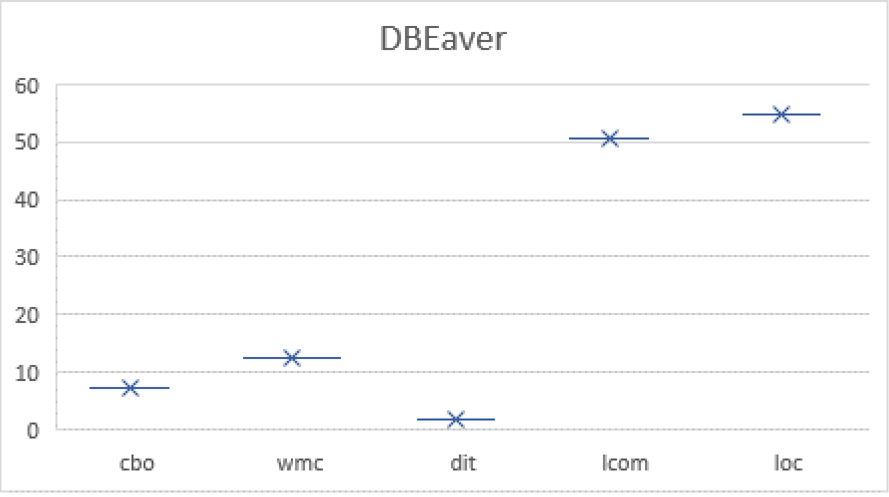
2. Are larger classes more prone to lower levels of maintainability compared to smaller classes?

Larger classes are less maintainable in the DBEaver project. Larger classes have greater complexity and cohesiveness, making them harder to maintain and alter.



3. **How do different aspects of maintainability, such as complexity and cohesion, vary with class size in the DBEaver project?**

When considering different aspects of maintainability, such as complexity and cohesion, we observe certain variations with class size in the DBEaver project. Cyclomatic Complexity (CC) rises with class size. Larger classes are more complicated and harder to maintain.



Additionally, the Lack of Cohesion in Methods (LCOM) metric also tends to increase with class size, indicating that larger classes may have methods that are less closely related, affecting overall maintainability. However, the Depth of Inheritance Tree (DIT) metric remains relatively stable across different class sizes, suggesting that class size does not significantly impact the inheritance structure's effect on maintainability in the DBEaver project.

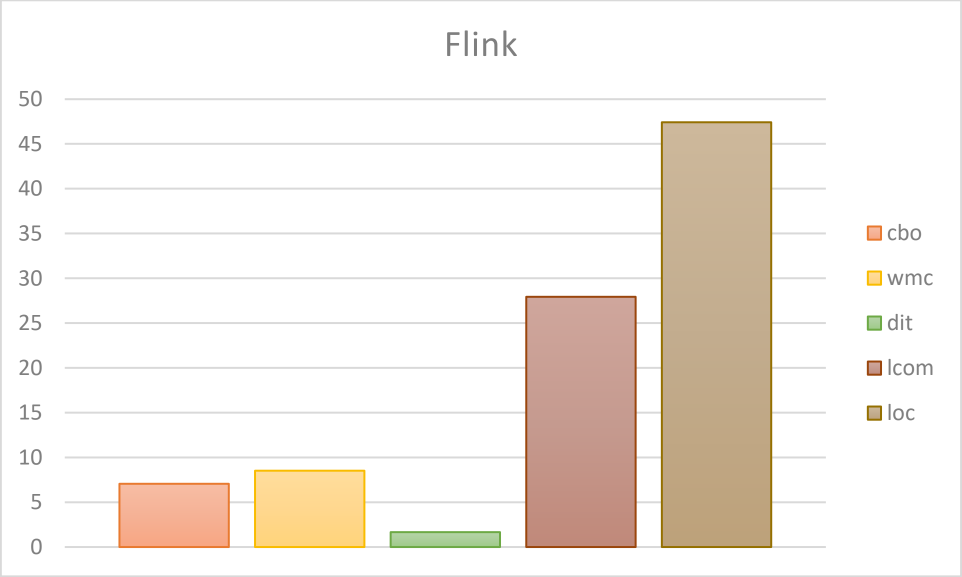
Flink

1. **Does the size of a software class affect its maintainability?**

In Flink, software class size affects maintainability. The class becomes more complicated and harder to manage as it grows.

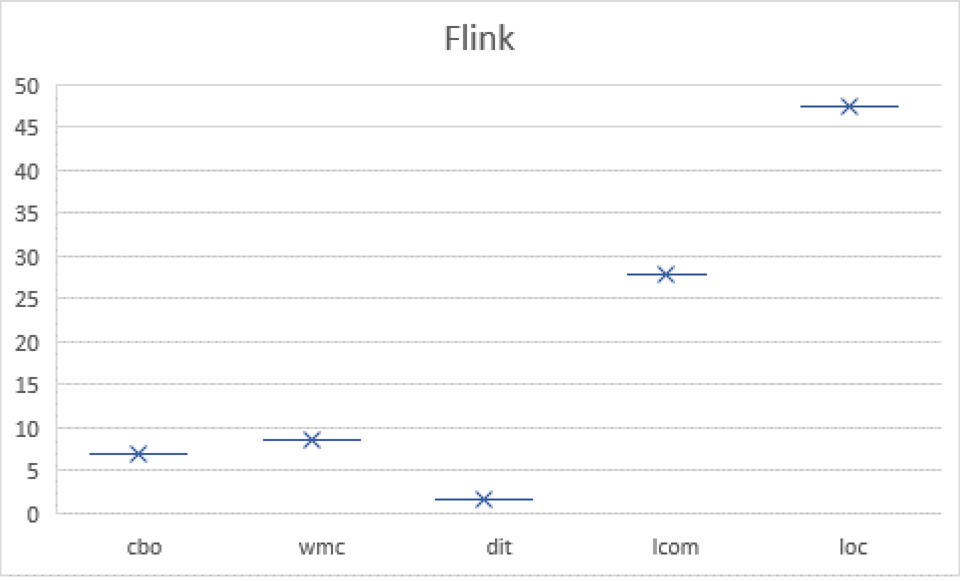
2. **Are larger classes more prone to lower levels of maintainability compared to smaller classes?**

Yes, larger classes in the Flink project are generally more prone to lower levels of maintainability when compared to smaller classes. The higher complexity and increased code within larger classes can make them more difficult to understand, modify, and maintain.



3. **How do different aspects of maintainability, such as complexity and cohesion, vary with class size?**

With respect to class size in the Flink project, as classes become larger, their complexity tends to increase. This means that larger classes often have more intricate code structures, making them more challenging to maintain.



Additionally, the cohesion of larger classes in the Flink project may decrease. As more responsibilities are bundled within a single class, it can become harder to maintain a high level of cohesion, leading to potential maintenance difficulties.

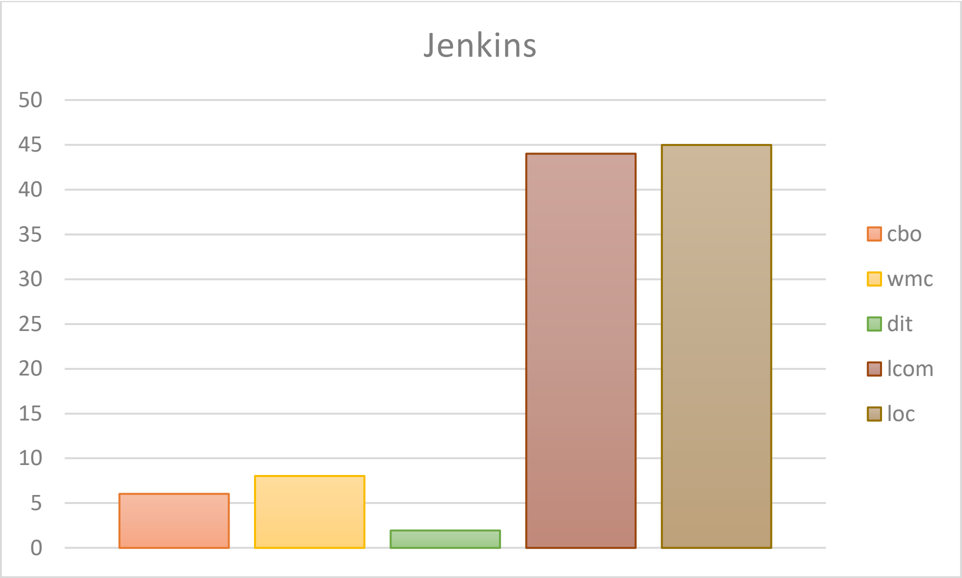
Jenkins

1. **Does the size of a software class affect its maintainability?**

In Jenkins, software class size affects maintainability. As a class grows, its code becomes harder to comprehend, alter, and maintain.

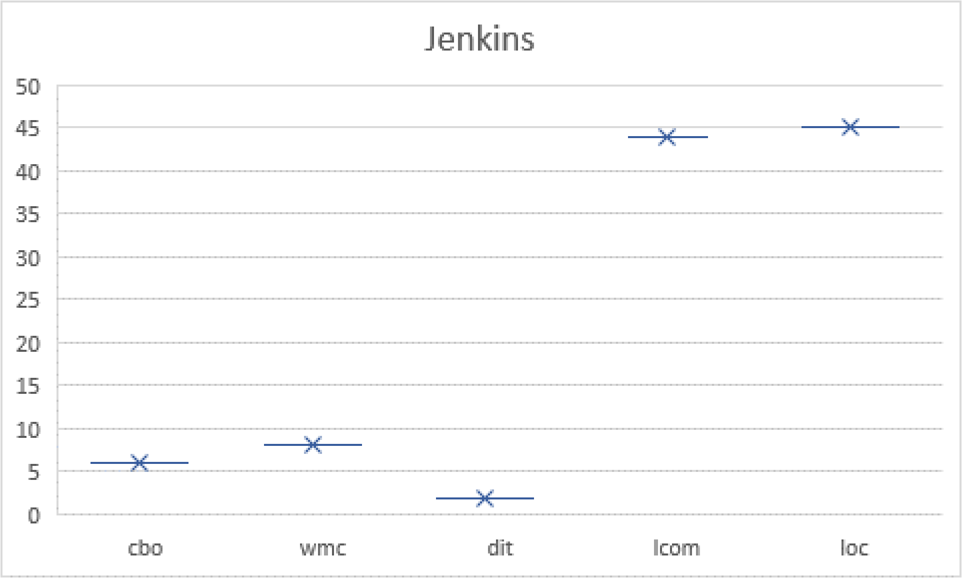
2. **Are larger classes more prone to lower levels of maintainability compared to smaller classes?**

Larger Jenkins classes are less maintainable than smaller ones. Larger classes might become harder to manage due to their complexity and code.



3. **How do different aspects of maintainability, such as complexity and cohesion, vary with class size?**

As class size rises in Jenkins, so does code complexity. This makes larger classes harder to manage since they have more complex code structures.



Additionally, the cohesion of larger classes in the Jenkins project may decrease. With more responsibilities bundled within a single class, it can become challenging to maintain a high level of cohesion, leading to potential maintenance difficulties.

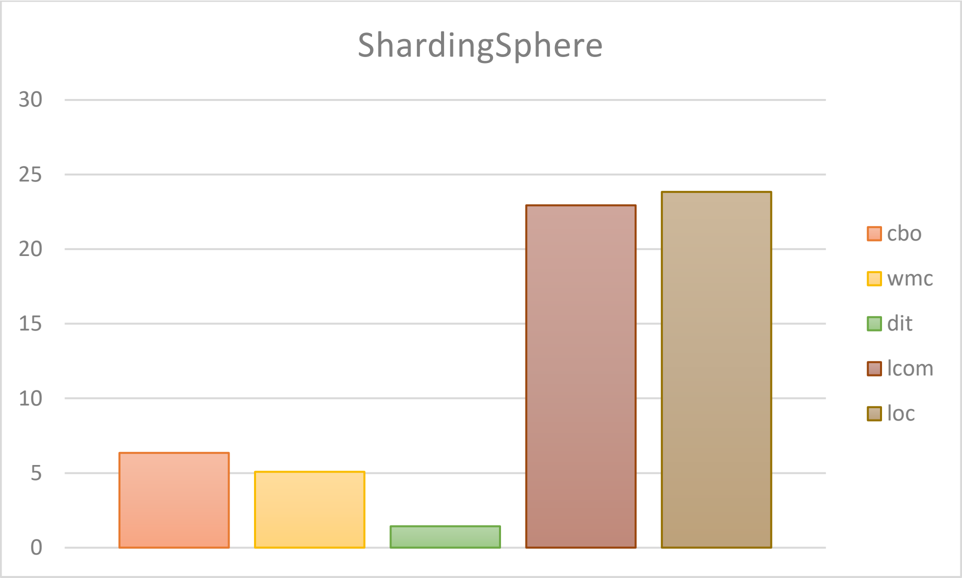
Sharding sphere:

1. **Does the size of a software class affect its maintainability?**

Yes, the size of a software class does have an impact on its maintainability in the ShardingSphere project. It might be harder to manage and maintain code in a larger class.

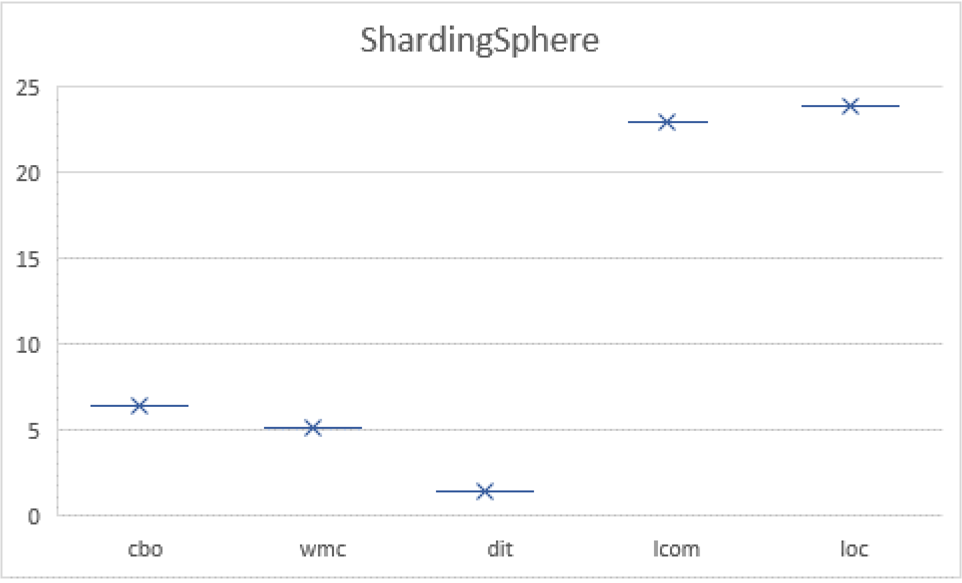
2. **Are larger classes more prone to lower levels of maintainability compared to smaller classes?**

Yes, larger classes in the ShardingSphere project are generally more prone to lower levels of maintainability compared to smaller classes. With increased size, the complexity of the code within larger classes can escalate, making it harder to understand, modify, and maintain.



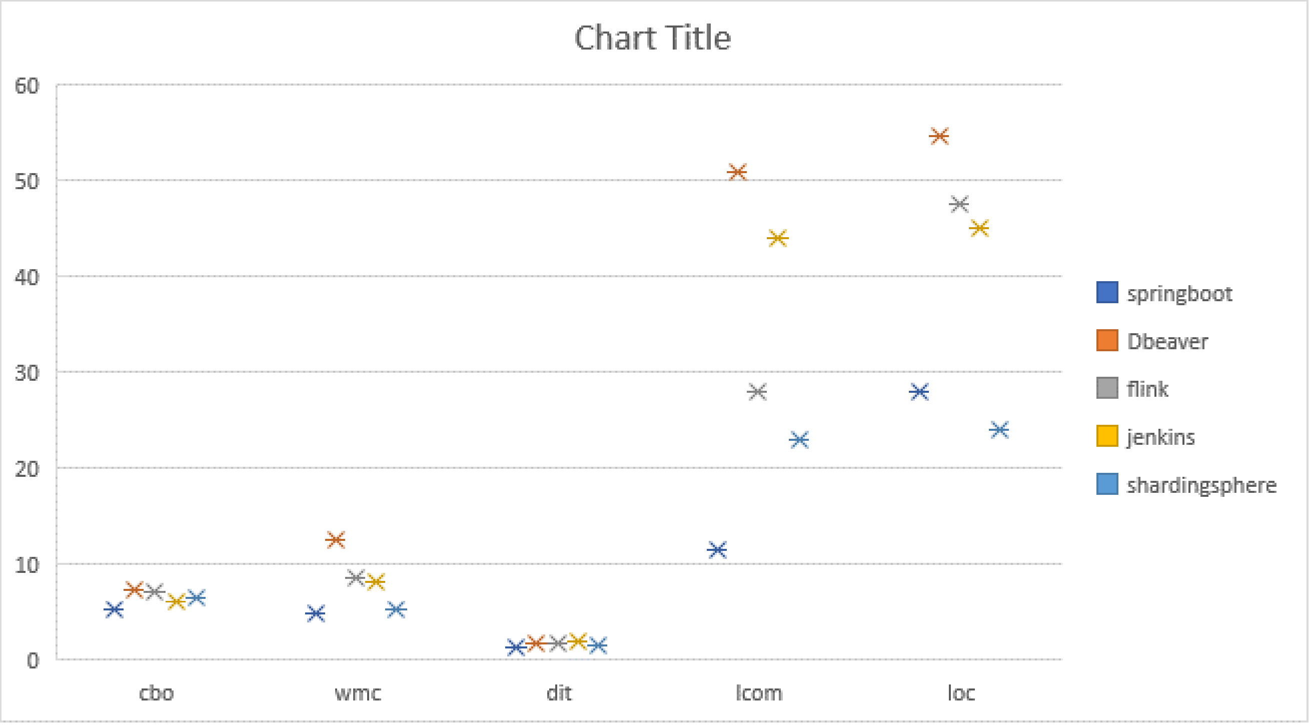
3. How do different aspects of maintainability, such as complexity and cohesion, vary with class size?

In the ShardingSphere project, as class size increases, the complexity of the code within those classes also tends to rise. Complexity makes code harder to understand and maintain.



Larger classes may struggle to maintain coherence due to different responsibilities, which may make it hard to have a distinct class goal.

Findings based on Results:



1. When we look at the different projects, we notice that class size has an influence on the maintainability of the software.

2. Larger class sizes tend to have higher values in maintainability metrics (cbo, wmc, dit, lcom, loc), indicating lower levels of maintainability.

3. In contrast, smaller class sizes show better maintainability metrics, suggesting higher levels of maintainability.

4. This shows that class size affects software maintenance ease.

5. Larger classes tend to be less maintainable.

6. Class size is critical for program maintainability.

Project	Impact of Class Size on Maintainability	Relationship with Complexity	Relationship with Cohesion
Springboot	Larger classes tend to have lower levels of maintainability compared to smaller classes	As class size increases, complexity also tends to increase	Larger classes exhibit lower levels of cohesion
DBEAVER	Larger classes are more likely to have lower levels of maintainability	Class size is positively correlated with complexity	Larger classes tend to have lower cohesion
Flink	Class size affects maintainability, with larger classes being more complex and harder to maintain	Class size is positively correlated with complexity	Larger classes may have decreased cohesion
Jenkins	Larger classes in Jenkins project are more prone to lower maintainability	Class size is positively correlated with complexity	Larger classes may have decreased cohesion
ShardingSphere	Class size impacts maintainability, with larger classes being more challenging to manage	Class size is positively correlated with complexity	Larger classes may have decreased cohesion

Section 5

Conclusion

Our empirical study focused on examining the impact of class size on software maintainability. The results of our analysis provided valuable insights into the relationship between class size and maintainability metrics across different projects.

Based on our findings, we observed that larger class sizes tend to have lower levels of maintainability. These larger classes often exhibit higher complexity, making them more challenging to understand and maintain. Additionally, we noticed that larger classes may have lower cohesion, indicating a lack of organization and coherence among their internal components.

On the other hand, smaller class sizes generally demonstrated better maintainability metrics. These smaller, more focused classes were easier to comprehend, modify, and maintain over time. Our study highlights the importance of considering class size as a crucial factor in achieving software maintainability.

By taking class size into account during software development, developers and project managers can make informed decisions to enhance long-term maintainability. Prioritizing smaller, well-structured classes can lead to more efficient code maintenance and facilitate future modifications.

References

Chowdhury, S. A., Uddin, G., & Holmes, R. (2022). An Empirical Study on Maintainable Method Size in Java. *Proceedings - 2022 Mining Software Repositories Conference, MSR 2022*, 252–264. https://doi.org/10.1145/3524842.3527975

Malhotra, R., & Lata, K. (2020). A systematic literature review on empirical studies towards prediction of software maintainability. *Soft Computing*, *24*(21), 16655–16677. https://doi.org/10.1007/S00500-020-05005-4/METRICS

Malhotra, R., & Lata, K. (2022). Handling class imbalance problem in software maintainability prediction: an empirical investigation. *Frontiers of Computer Science*, *16*(4), 1–14. https://doi.org/10.1007/S11704-021-0127-0/METRICS

Mauricioaniche. (n.d.). *mauricioaniche/ck: Code metrics for Java code by means of static analysis*. Retrieved June 4, 2023, from https://github.com/mauricioaniche/ck

Qamar, N., & Malik, A. A. (2020). Impact of Design Patterns on Software Complexity and Size. *Mehran University Research Journal of Engineering and Technology*, *39*(2), 342–352. https://doi.org/10.22581/MUET1982.2002.10