**Assignment 1**

**Group Assignment**

**Subject: Object Oriented Development**

**Table of Contents**

[Section 1 1](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815676)

[Introduction 4](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815677)

[Background and motivation 4](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815678)

[GQM Approach: 4](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815679)

[Study Goal: 4](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815680)

[Research Questions: 4](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815681)

[Metrics: 5](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815682)

[1. Lines of Code (LOC): 5](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815683)

[2. Weighted Methods per Class (WMC): 5](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815684)

[3. Coupling Between Objects (CBO): 5](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815685)

[4. Depth of Inheritance Tree (DIT): 5](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815686)

[5. Lack of Cohesion of Methods (LCOM): 5](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815687)

[Section 2 6](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815688)

[Criteria for Study and Program Selection: 6](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815689)

[1. Size: 6](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815690)

[2. Popularity: 6](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815691)

[3. Project Size: 6](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815692)

[4. Activity Level: 6](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815693)

[Subject programs 7](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815694)

[1. Spring Boot: 7](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815695)

[2. DBeaver: 7](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815696)

[3. Apache Flink: 8](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815697)

[4. Jenkins: 8](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815698)

[5. ShardingSphere: 8](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815699)

[Rationale for Selection Criteria and their Implications for Maintainability 8](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815700)

[1. Size: 8](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815701)

[2. Popularity: 8](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815702)

[3. Project Complexity: 9](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815703)

[4. Activity Level: 9](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815704)

[Section 3 9](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815705)

[Tool Description 9](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815706)

[Overview of the Tool: 9](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815707)

[How it Measures C&K Metrics for Java Code: 10](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815708)

[Instructions for Using the Tool: 10](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815709)

[1. Obtaining the Tool from GitHub: 10](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815710)

[2. Following the Instructions in the ReadMe File: 10](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815711)

[Section 4 11](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815712)

[Results 11](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815713)

[Springboot 11](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815714)

[DBEAVER 12](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815715)

[Flink 14](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815716)

[Jenkins 15](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815717)

[Sharding sphere: 17](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815718)

[Findings based on Results: 19](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815719)

[Section 5 20](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815720)

[Conclusion 20](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815721)

[References 21](file:///C:\Users\user\Pictures\ALPHA\ORIGINAL%20FILES\Report%20(GroupAssignment1).docx#_Toc136815722)

# Section 1

# Introduction

## **Background and motivation**

It is referred to as maintainable software when the software can be easily modified, comprehended, and fixed during the course of its lifetime. The ease with which software classes can be maintained might be impacted by the size of the classes themselves. In accordance with the findings of the size of the class has the ability to increase complexity and make code maintenance more difficult.

As part of this empirical study, we investigate the relationship between class size and the maintainability of software. If there is a correlation between the size of a class and its maintainability, we would appreciate it if you could let us know whether or not larger classes are intrinsically less maintainable.

Through the examination of real-world Java projects and the utilization of appropriate metrics, we are able to gain insight into the relationship between the size of classes and the maintainability of software (Malhotra & Lata, 2022). The findings of this study can be utilized by software developers and project managers to impact decisions regarding class size and the long-term effects that it has on maintainability.

# GQM Approach:

## **Study Goal:**

One of the most important questions that is driving this research is: to what extent does the number of software classes have an impact on the maintainability of the software? The possibility that the program's quality and its capacity to continue for an extended period of time could be impacted by the number of participants has piqued our excitement.

## **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:

### Lines of Code (LOC):

It is possible to determine the size of the class by utilizing this statistic, which involves summing up the lines of code.

### Weighted Methods per Class (WMC):

This metric provides a numerical representation of the complexity of a class by counting the methods and assigning a weight to each one of them after the count is complete.

### Coupling Between Objects (CBO):

This metric is used to assess the degree to which a class is dependent on other classes in the system. It also measures the degree to which a class is coupled to other classes.

### Depth of Inheritance Tree (DIT):

The position of the class in the succession tree can be expressed by the use of this metric, which counts the number of levels in the class hierarchy.

### Lack of Cohesion of Methods (LCOM):

The objective of this statistic is to provide an accurate measurement of the degree to which a collection of procedures that are related to one another is cohesive.

By analyzing these metrics (Qamar & Malik, 2020), This study aims to examine the correlation between class size and maintainability in more detail. Our focus will be on finding out whether larger classes are inherently less maintainable and, if so, how various characteristics of maintainability change as class size changes.

# Section 2

# Criteria for Study and Program Selection:

## **Size:**

To determine how large programs are in comparison to other relevant size metrics, such as the number of lines of code (LOC), the primary objective of the size criteria is to determine how big programs are. At the conclusion of the day, we gained a deeper understanding of the ways in which the diversity of program complexity and the effects of class size on maintainability interact.

## **Popularity:**

There are a number of indicators that are taken into consideration by the popularity criterion in order to determine the level of community interest and approval. These indicators include the number of views, stars, and forks. Utilizing this method ensures that popular and widely used applications are selected, which can teach us a great deal about techniques that are helpful in maintaining maintainability.

## **Project Size:**

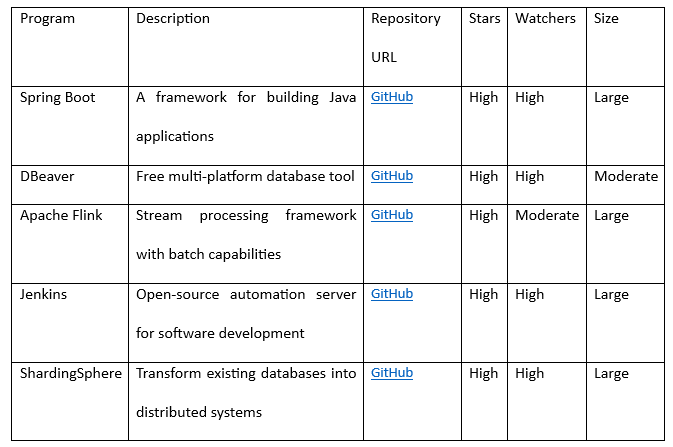
With the help of this criterion, the total size of the apps is calculated by adding together all of the pertinent metrics, which include the number of files and modules. Due to the fact that the research studies programs of varying sizes, it is able to investigate the ways in which the size of the class influences the maintainability of projects of varying sizes.

## **Activity Level:**

The amount of labor that goes into the continuing development and maintenance of the programs is determined by the activity level criterion, which is determined by counting the number of commits and contributors from the program. By running more programs, one can gain a better understanding of the best practices for maintenance and the challenges that are encountered in the real world.

By utilizing these parameters, the research intends to analyze the impact that class size has on the maintainability of software in programs that range in size, popularity, and activity levels.

## **Subject programs**



### Spring Boot:

A widely used Java framework that substantially simplifies programming because to its numerous useful features. Using Spring simplifies not just the development of Spring-based production apps but also their deployment.

### DBeaver:

All three categories of database managers—developers, SQL programmers, and administrators—can benefit from this flexible and free database solution. Maybe things will work out well for everyone involved. In addition to being compatible with, it also complies with, a wide range of operating systems. Data management, query execution, and monitoring database administration requirements are all made easy with this program's extensive feature set. It works with many different database formats and administration systems.

### Apache Flink:

This architecture greatly facilitates the operation of both batch and stream processing. It is my firm belief that you will enjoy it. Flink excels in all of these areas and more, including having a low event latency, a high throughput, and fault tolerance. However, two of its strongest points are event-driven programming and data processing and analytics. Our usage of event-driven programming is another strength of ours.

### Jenkins:

Integration and delivery processes can be made more efficient with the help of this open-source automation server. Jenkins streamlines software development by automating project construction, testing, and deployment, greatly improving effectiveness and efficiency.

### ShardingSphere:

This project's objective was to build distributed systems, and one of the ways that it was going to accomplish this was by making use of existing database designs. Governance, distributed transactions, and data sharding are some of the characteristics that are incorporated into the service that it provides. The utilization of ShardingSphere makes it possible to optimize the speed of the database in addition to augmenting its scalability.

By making use of these applications, we hope to acquire a deeper comprehension of the ways in which the size of a class influences the maintainability of software. Due to the fact that these projects encompass such a wide variety of kinds of organizations and use cases, this presents us with an excellent opportunity to gain knowledge.

## Rationale for Selection Criteria and their Implications for Maintainability

### Size:

By selecting projects based on their individual sizes, we ensure that there is a diverse range of codebases that are available to choose from. There are differing degrees of complexity and potential concerns regarding maintenance that are associated with each of these codebases. It may be beneficial to examine projects of varying sizes in order to gain an understanding of the ways in which the size of the class affects the maintainability of the system under various circumstances.

### Popularity:

Examining the degrees of popularity of the projects is one method that may be utilized to acquire a more comprehensive comprehension of the software systems that are already in use. Through the examination of well-known projects, which typically go through ongoing maintenance and changes, one can gain a great deal of knowledge regarding the complexity involved in the maintenance of software that is widely used.

### Project Complexity:

The characteristics, architecture, and use cases of the selected projects exhibit a wide range of complexity, which should be taken into consideration. There is a correlation between the size of a class and the maintainability of software systems, regardless of how simple or complex they are. This correlation may be observed in projects that have varied degrees of complexity.

### Activity Level:

### The act of making commitments and the presence of a large number of personnel working on the project are both indicators of active development and upkeep. Taking a look at these examples can help throw light on the challenges and triumphs that are associated with actively building software maintenance in the real world.

It has been discovered that the dimensions of a class have an effect on the ease with which software may be maintained, which is why these requirements are essential for maintainability. The capacity to understand, edit, and troubleshoot larger classes becomes more difficult, which in turn has an impact on the maintainability of such classes (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:

When working on a Java project, the CK-Code metric tool is an invaluable resource for calculating code metrics on both the class and method levels. It sheds light on numerous facets of software quality via the use of static analysis techniques. Along with numerous other metrics, the tool incorporates the following: CBO, FAN-IN, FAN-OUT, DIT, NOC, visible methods, NOSI, RFC, WMC, LOC, LCOM, LCOM\*, TCC, LCC, and a plethora of others.

### How it Measures C&K Metrics for Java Code:

The CK-Code metric tool is used to do static analysis on Java code in order to determine the C&K metrics. For the purpose of doing precise metric computations, it makes use of the Eclipse JDT Core package to construct the Abstract Syntax Tree (AST) of the code. It is possible to have a better understanding of software maintainability with the assistance of this tool, which examines the structure and properties of the code in order to identify a number of criteria for code quality.

### Instructions for Using the Tool:

#### **Obtaining the Tool from GitHub:**

To access the CK-Code metric tool, we downloaded it from the GitHub repository at [link](https://github.com/mauricioaniche/ck) (Mauricioaniche, n.d.) The repository contains all the necessary files and resources required to utilize the tool effectively.

#### **Following the Instructions in the ReadMe File:**

By reading the ReadMe file, you can confirm that the tool can be used in the appropriate manner. In the ReadMe file, you will find instructions that are both clear and comprehensive for installing and configuring the application, as well as advice for making certain important adjustments.

Either you can use the CK-Code metric tool on its own or you can incorporate it into your Java program. Both opportunities are available to you. Any one of the two strategies might be successful. It is necessary to create a copy of the project and then execute the JAR file that is produced in order to use it on your own. Different metrics for variables, classes, and methods are included in each of the three distinct CSV files that are generated by the software. The addition of the CK library as a Maven dependency to our Java program is yet another alternative for performing metric computations to consider.

# Section 4

## **Results**

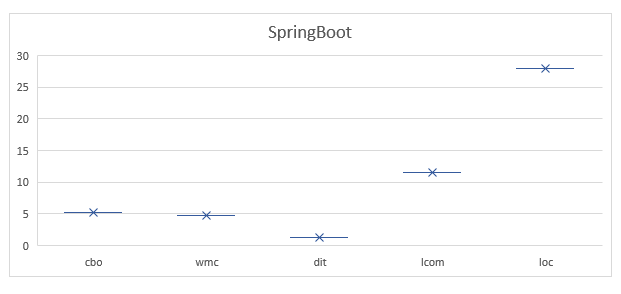
### Springboot

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

Yes, take into consideration that the class sizes of a Spring Boot project have an impact on the maintainability of the project. The maintenance of smaller classes within a project is simpler than the maintenance of bigger classes.

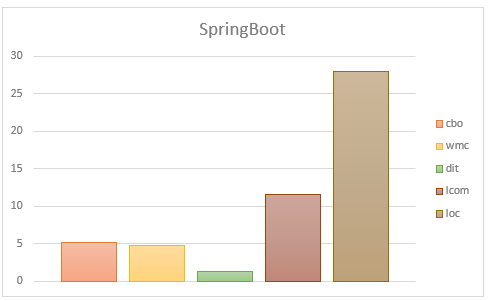
1. **Are larger classes more prone to lower levels of maintainability compared to smaller classes?**

Yes, according to the findings of the Spring Boot project's investigation, it is simpler to maintain classes that are smaller in size than those that are larger. Taking into account the results of their investigation, the researchers came to this conclusion. The size of the class has a significant impact on its capacity to be maintained.



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

An concern with regard to maintainability is the size of the class, as demonstrated by the analysis performed by Spring Boot. Coding gets more difficult when there are more students in a classroom than there were before. Complex code is a source of aggravation due to the fact that it is difficult to understand and frequently cannot be maintained.



When it comes to cohesion, which is defined as the degree to which students are connected to one another and structured, it would appear that larger Spring Boot classrooms have lower levels of cohesion.

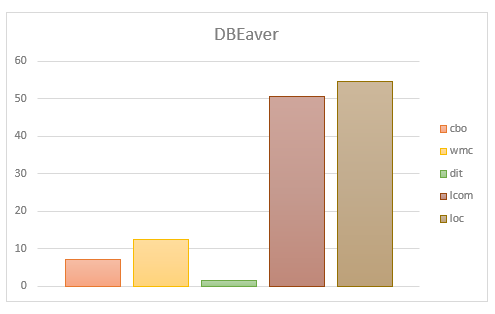
### DBEAVER

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

It is important to note that the quantity of software classes in the DBEaver project has an impact on maintainability. As students continue through their educational journey, it becomes increasingly challenging for them to comprehend and organize the academic activities they are required to complete.

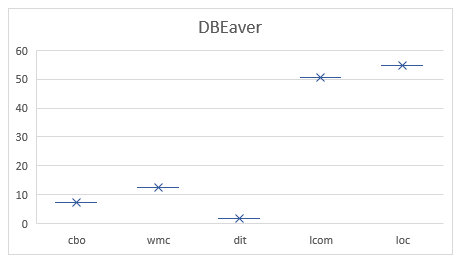
1. **Are larger classes more prone to lower levels of maintainability compared to smaller classes?**

According to the DBEaver project, larger courses pose a greater challenge in terms of maintenance. When compared to a smaller class, a larger class is more difficult to maintain and adapt due to its higher level of complexity and internal consistency.



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

As demonstrated by the DBEaver project, the size of the class has an impact on a number of maintainability parameters, including cohesiveness and complexity and cohesion. As the number of students in a class increases, the cyclomatic complexity (CC) also increases. It is more challenging and time-consuming to instruct a larger class than it is to instruct individuals in a smaller class.



In addition to this, the Lack of Cohesion in Methods (LCOM) metric has a tendency to increase in tandem with the growth of class composition. Given that larger classes may include methods that are less intimately related to one another, this has consequences for the ease with which maintenance can be performed over time. In accordance with the Depth of Inheritance Tree (DIT) metric, the inheritance structure of the DBEaver project has a little impact on the maintainability of the respective program. This is the case regardless of the overall number of students who are registered in the facility.

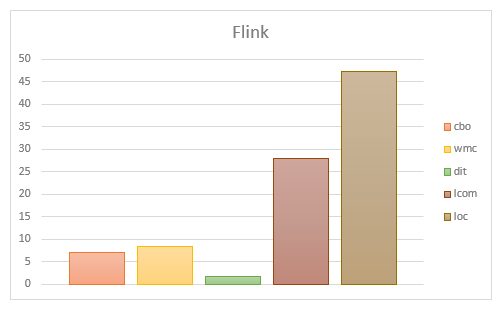
### Flink

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

Maintaining Flink becomes more difficult as the number of students in a class increases. As the number of pupils in the classroom increases and the level of difficulty of the curriculum also increases, maintaining a clean and organized classroom becomes an increasingly tough mission.

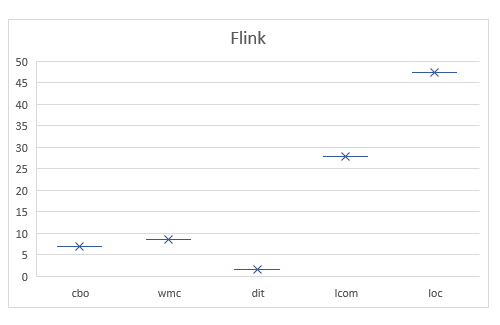
1. **Are larger classes more prone to lower levels of maintainability compared to smaller classes?**

In the context of the Flink project, it is anticipated that classes with a smaller size will be simpler to manage than classes with a bigger size. This overarching recommendation ought to be adhered to in each and every classroom. Working with excessively lengthy and complex classes may be challenging due to the additional code and complexity that you will be required to implement.



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

Within the scope of the Flink project, the complexity of a class increases in a manner that is linearly proportional to the size of the class. This indicates that larger classes typically have more complicated code structures, which makes them more difficult to maintain than smaller classes.



Additionally, it is possible that the larger classes that are part of the Flink project are not as unified as they may be. When more responsibilities are consolidated into a single class, it becomes more difficult to maintain consistency, which may result in problems with maintenance.

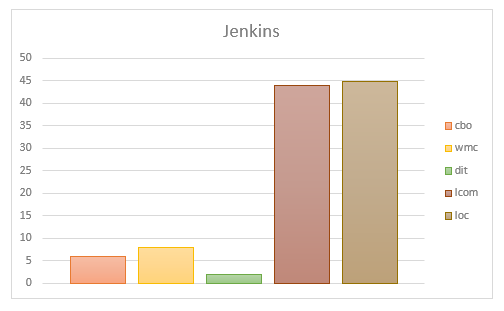
### Jenkins

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

When it comes to maintainability, Jenkins makes the influence of software class size considerably more obvious than it would otherwise exist. Understanding, modifying, and maintaining the most recent version of the code associated with a class becomes increasingly challenging as the class grows in size.

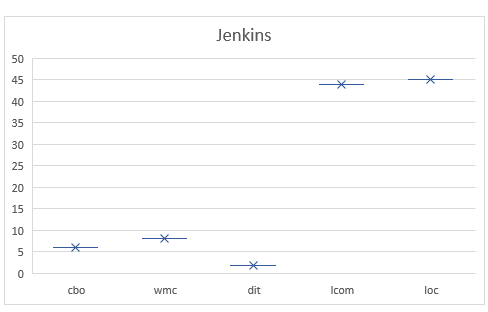
1. **Are larger classes more prone to lower levels of maintainability compared to smaller classes?**

Classes in Jenkins that have a greater number of components are more difficult to update and fix than classes that have a smaller number of components. The management of larger classes can be more difficult to accomplish due to the intricacy and coding involved in them.



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

While utilizing Jenkins, the complexity of the code increases in a manner that is directly proportional to the size of the class. Managing larger classes that have more complicated code structures is becoming an increasingly tough task overall.



The Jenkins project's cohesiveness might be harmed by the addition of more elaborate classes. The process of consolidating an increasing number of jobs into a single class makes it more difficult to keep everyone informed and in the loop, which may result in confusion regarding the requirements for maintenance.

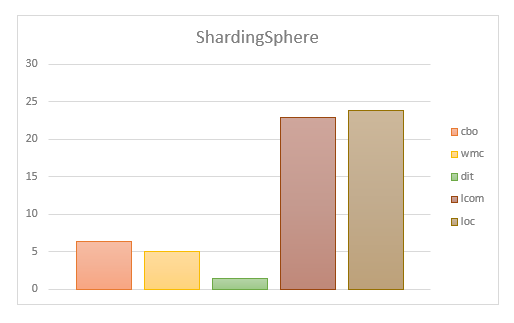
### Sharding sphere:

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

Larger class sizes are more challenging to manage, as the ShardingSphere project proved. The challenge of keeping the code current while still being manageable may increase in direct proportion to the number of students enrolled in a given class.

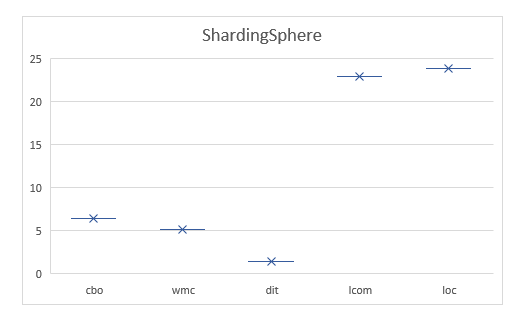
1. **Are larger classes more prone to lower levels of maintainability compared to smaller classes?**

Keeping a ShardingSphere project class with fewer members is usually easier than keeping a larger class with same functionality. Class sizes have a negative correlation with code complexity, making larger classes harder to understand, modify, and maintain. This reduces the usability of bigger classes.

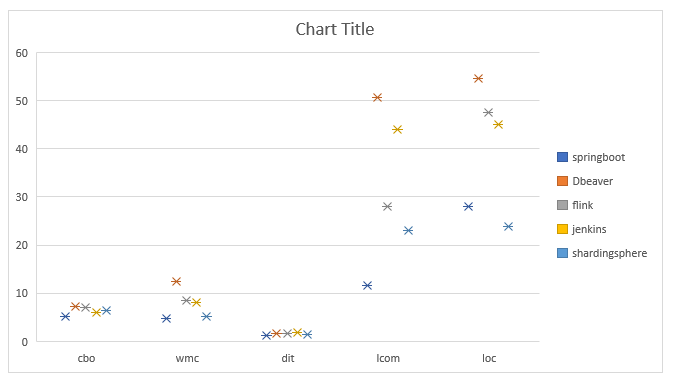


1. **How do different aspects of maintainability, such as complexity and cohesion, vary with class size?**

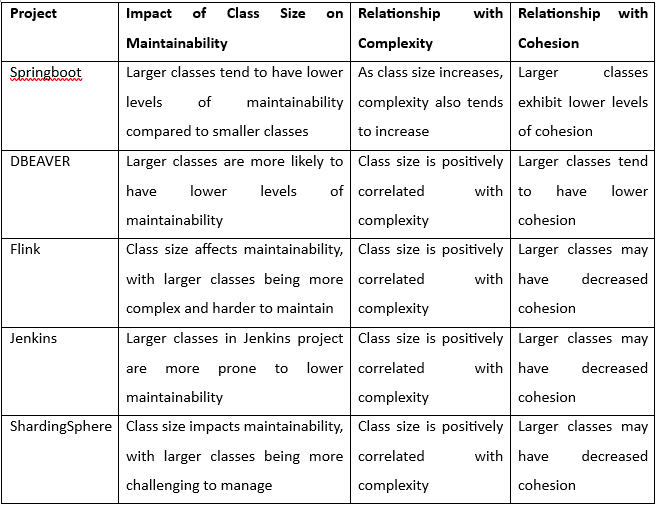
According to the findings of the ShardingSphere experiment, class sizes have a tendency to expand in an exponential manner as the complexity of the code increases. When code is complicated, it is typically tough to comprehend, and it is even more challenging to keep things running smoothly.



## **Findings based on Results:**



1. By comparing different projects, we are able to see that the size of the class has an effect on the maintainability of the software.
2. According to maintainability measurements (cbo, wmc, dit, lcom, and loc), class sizes that have a greater number of members are more difficult to keep up with.
3. In general, classes with fewer members are easier to maintain since they typically have better maintainability metrics. This makes those classes easier to manage. Because it is much simpler to manage a smaller class, the reason for this is that.
4. This makes it quite clear that the size of the class has an effect on how simple it is to maintain software.
5. As the number of students in a class increases, the convenience of maintenance reduces.
6. There is a significant impact that the class size has on the maintainability of the program.



# Section 5

## **Conclusion**

In the course of our empirical analysis, the primary focus was on determining how the size of the class affected the maintainability of the software. Through the completion of our research, we were able to shed light on the relationship that exists between the size of the class and specific maintainability measures that were evaluated in various projects.

It became quite evident to us that when class sizes are excessively large, maintainability suffers a significant setback. Because of the increased complexity of these larger classrooms, it is more difficult to comprehend and keep up with students in them. The fact that larger classrooms might not be as cohesive as smaller ones is another discovery that was made. In light of this, it is possible that the components that constitute the larger classes are not as well organized or cohesive.

On the other hand, classes that had a smaller number of students received higher scores on the criteria that evaluate maintainability. The more intimate and focused classes were far easier to comprehend, easily adjust to, and keep up with when compared to the larger classes that were offered. According to our findings, the size of the class is a significant determinant in the maintainability of the software.

By taking into account the number of students in the class while making decisions, software engineers and project managers can improve the product's ability to be maintained over many years. We might be able to increase the efficiency of code maintenance and make it simpler to make improvements in the future if we give priority to classes that are smaller and more well-structured.

# 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