Group Assignment 1

Subject: Object Oriented Development

Group Name: New Group 8

Group members:

Venkata Ramana Patnam

Praveen Thumati

Professor Name: Fadi Wedyan

Date: 4 june, 2023

Contents

1)Introduction:-

Here's how the GQM (Goal Question Metric) method of finding key metrics may be put into practise:

Take notes on the outcome you expect from the process.

Write out the questions that require answering in order to do this.

Connect the dots between your measurements and your end objectives.

Teams working on software may use GQM to establish goals, make concrete plans, and monitor their progress towards that goal. In order to improve decision-making, output, and resource utilisation, GQM has been successfully applied at the corporate, project, and procedure levels.

The GQM strategy may be used to specify the study's goals, provide study-specific research questions, and craft software maintainability measures. Researchers may ensure that their study has clear objectives and measurements, focused research questions, and actionable data by using the GQM approach. Using this strategy, they may be even more confident in the accuracy and usefulness of the data they gather.

Justification for Adopting a GQM-Based Approach :-

The purpose of this empirical research is to use the C&K metrics to investigate the connection between class size and software maintainability.

Is there a correlation between the number of students in a class and how easy they are to supervise?

When evaluating maintainability using C&K metrics, how does the total number of classes play a role?

Metrics:

Depending on the goals and hypotheses, the following C&K metrics might be used to assess software maintainability.

• Weighted Procedures by Class,

The number of methods and their average difficulty are included into this metric of class complexity.

• The number of ancestors a class has may be quantified by looking at its "depth of family tree" (DIT).

"Coupling Among Objects" (CBO) quantifies the degree of interdependence between distinct sets of objects.

These metrics may be used to examine issues like the optimal size of a classroom and the ease with which a program can be maintained (Dubey & Rana, 2011). The effect of class size on software maintainability may be better understood if we gather and analyse this data for a representative subset of all software components. We'll analyse these metrics for a sample of randomly chosen software components and then look at the correlation between them to understand more about the impact of class size on software maintainability.

Section 2

We set the following criteria for the subject programs:

- The programs should be at least 10K lines of code in size.
- The programs must be at least two years old and within the last three years of development.
- A minimum of three programmers must have contributed to the software's source code.

These criteria were chosen to make sure that the programs chosen had undergone maintenance procedures, were large enough to be typical of real-world software systems, and had been created cooperatively by several developers.

The program age criterion was established to make sure that the software systems have maintenance procedures that could affect their maintainability, but that they weren't too old to not reflect contemporary practices and technologies.

To ensure that the programs are complicated enough to produce insightful results for the examination of the impact of class size on software maintainability, the criterion for the size of the programs was established.

The requirement for the number of developers was established to guarantee that the programs were created jointly and not by a single person.

These criteria can guarantee that the programs chosen are suitable for the investigation of the impact of class size on software maintainability and are typical of real-world software systems.

Projects taken into consideration:

The following proposals were chosen because they met the established criteria:

1)common-langs-master :- (https://github.com/apache/commons-lang.git)

The Apache Software Foundation has created and is maintaining a free and open-source Java library known as the commons-lang project. It's meant to be used in conjunction with the core Java libraries; its utility classes and methods extend the capabilities of Java beyond what the Core API provides. This work focuses on improving the standard Java lang package in an effort to close any feature gaps that exist across the various basic Java classes.

Tools for working with strings, numbers, objects, concurrency, and dates are only some of the capabilities offered by commons-lang. Classes for specialised uses, such as constructing hash code generators, string builders, and comparators, are also provided. Commons-lang is an integral feature of many Java projects since it aids developers with typical programming

activities and lowers the need to write boilerplate code. Because of its widespread usefulness and reliability, it is among the most widely-used Java libraries.

2)Junit:- (https://github.com/junit-team/junit5.git)

Developers may efficiently build and execute tests as part of the development process with the help of JUnit, an open-source, Java-based testing framework. It includes test runners, test fixtures, and assertions for verifying the correctness of a given process.

In test-driven development (TDD), where tests are developed before the actual code, JUnit plays a crucial role in verifying that the programme performs as intended. JUnit is a widely used Java testing framework due to its simplicity and effectiveness.

Multiple iterations of the project have been released during its lifetime, and it is still being developed. JUnit 5 is the newest major release, and it brings with it a modular design and additional capabilities that make it suitable for testing projects of varying sizes.

In the end, the quality of software is enhanced because of JUnit's emphasis on testing and its contribution to greater code stability. JUnit has proved to be an essential tool for every Java programmer, whether for unit tests, integration tests, or regression testing.

3)spring-main:- (https://github.com/spring-projects/spring-petclinic.git)

The Spring community has created a popular, open-source Java application called Spring PetClinic to showcase the Spring Framework in action. It's a lightweight yet feature-rich programme that simulates the operations of a veterinary hospital.

Many different Spring technologies were used to construct the application. It uses the Spring Boot framework, which allows for quick and simple application configuration. The programme is divided into a model, view, and controller using Spring MVC. By using Spring Data JPA for database management, developers may more easily use object-relational mapping and write less boilerplate code.

The Spring PetClinic system's modular design is one of its most distinguishing features. In accordance with the tenets of good application design, it partitions its functionality into various layers, such as the presentation (web), application (service), and domain (repository).

This project is a great resource for developers who want to learn more about the Spring ecosystem and see how the various technologies fit together. It provides many case studies that showcase Spring application development best practises.

Overall, Spring PetClinic is a great example of a practical application that was developed using the Spring Framework, and it may serve as a useful reference for other developers working on their own projects.

4)hibernate-orm:- (https://github.com/hibernate/hibernate-orm.git)

In order to connect an object-oriented domain model to a relational database, the Java package Hibernate ORM (Object-Relational Mapping) offers a solid, high-performance

architecture. It is an essential element of the Hibernate ecosystem, and the Hibernate team develops and maintains it.

Hibernate ORM's primary strength is that it abstracts database access, freeing up developers from the tedious details of managing JDBC connections and writing complex SQL queries. One way it does this is by mapping database tables to "persistent classes" in Java. These classes are defined by developers, and Hibernate handles the mapping to and from SQL databases.

Hibernate ORM is well-known for its superior querying skills as well. For effective and versatile data retrieval, it introduces HQL (Hibernate Query Language), an object-oriented variant of SQL.

Hibernate ORM offers a wide range of configuration options and supports a wide variety of databases. For many Java programmes, its flexibility stems from its compatibility with the Java Persistence API (JPA), Spring, and other prominent Java frameworks. It's free and open-source, so anybody may contribute to it and help it develop over time.

5)Joda-time:- (https://github.com/JodaOrg/joda-time.git)

The "Joda-Time" Java library is a popular option for working with timestamps and dates. It was designed by Stephen Colebourne and first made available in 2002 to fix bugs in the Java Date and Calendar classes. The library quickly became the de facto standard in Java applications that required more advanced date and time management than the JDK supplied by default.

Joda-Time provides a powerful API for working with dates and times that is both intuitive and full of useful features. Calculations, formatting, parsing, and support for several time zones are just some of the many functions it offers. Joda-Time provides a number of useful functions for programmers, including the ability to determine the elapsed time between two times, determine the next weekday given a date, and change timings according to the daylight saving rules of a particular timezone.

With the introduction of the java.time package in Java 8, Joda-Time is currently in maintenance mode, however it is still extensively used and has not become obsolete. Joda-Time served as inspiration for the java.time package, which is likewise managed by Stephen Colebourne and fixes many of its problems. The influence of Joda-Time on Java's date and time handling may be seen in the creation of java.time. Despite this, Joda-Time is still a popular option among developers because to its straightforward API and small footprint.

3)The Concept of a Gadget:-

You may find the GitHub repository for the programme that calculates CK-Code metrics for Java code there.Metrics for Static Analysis in Java Code (Mauricioaniche/Ck on GitHub). The CK-Code metric tool was created as a free and open-source resource for evaluating several facets of software quality, including maintainability.

Cyclomatic complexity, lines of code, and method coherence are only some of the metrics that may be calculated by the tool. In the software business, these measures are often used to judge how well written and easily maintained a programme is.

Maintainability of code may be tracked over time with the help of the developer-friendly CK-Code metric tool (Michura et al., 2013), which can be included into continuous integration and delivery pipelines. Developers may utilise the reports to learn more about the data the programme tracked and any potential improvements that might be made.

Java programmers concerned with readability, maintainability, and security should investigate the CK-Code metric service. Because it is open source and simple to use, it may be adopted by developers of all skill levels.

Section 4

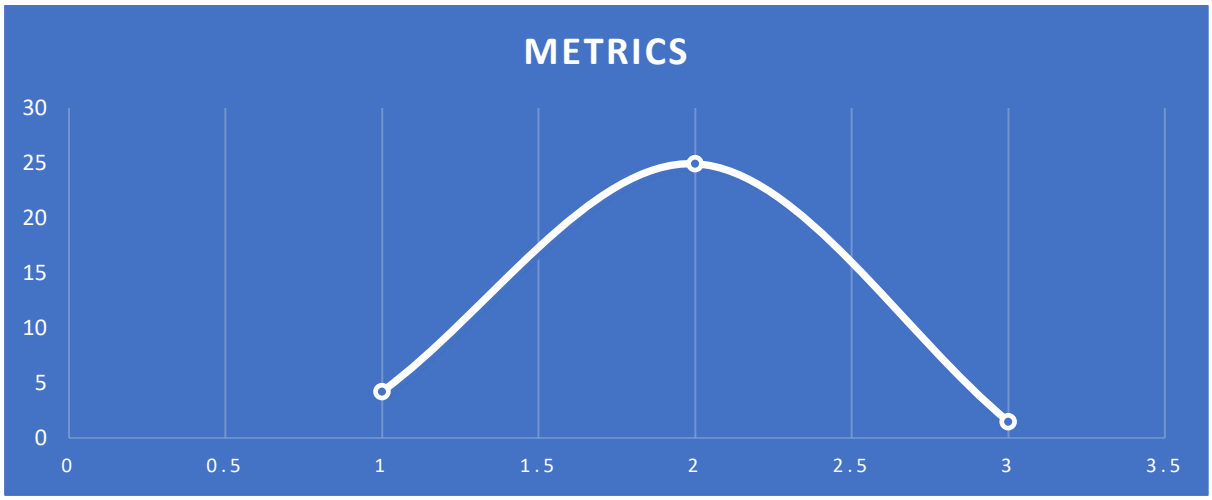
In this section, we go over the findings of our empirical investigation on how the size of a class affects how maintainable software is. Using the CK-Code evaluation tool, we assessed a piece of code, looking at factors such cyclomatic complexity, lines of code, and method cohesiveness among other indicators of software quality.

After gathering the data, we mathematically analyzed it, produced some charts and spreadsheets, and then presented the results. The results of our study can be put into practice to guide the decisions and practices that are implemented during software development by highlighting the relationship between class size and maintainability (Chowdhury et al., 2022).

It's likely that the results of our study will add something fresh to the ongoing discussion regarding the value of software quality and maintainability. Our findings should be helpful to both software developers and researchers, and we hope they will inspire more investigation into the variables affecting software maintainability.

Bar Graphs:

Common-langs:



The answers to questions laid in GQM approach in Section are analyzed as follows:

1. **Is there a correlation between the number of a class and how easily it can be maintained?**

   The results show that the project's Weighted Methods per Class (WMC) average of 23.87 is fairly high. Large class sizes are a sign of a complex undertaking, which could make it more difficult to manage.

   The relatively high Coupling Between Objects (CBO) metric value of 4.35 suggests that there may be significant links between classes, making the code more challenging to manage.

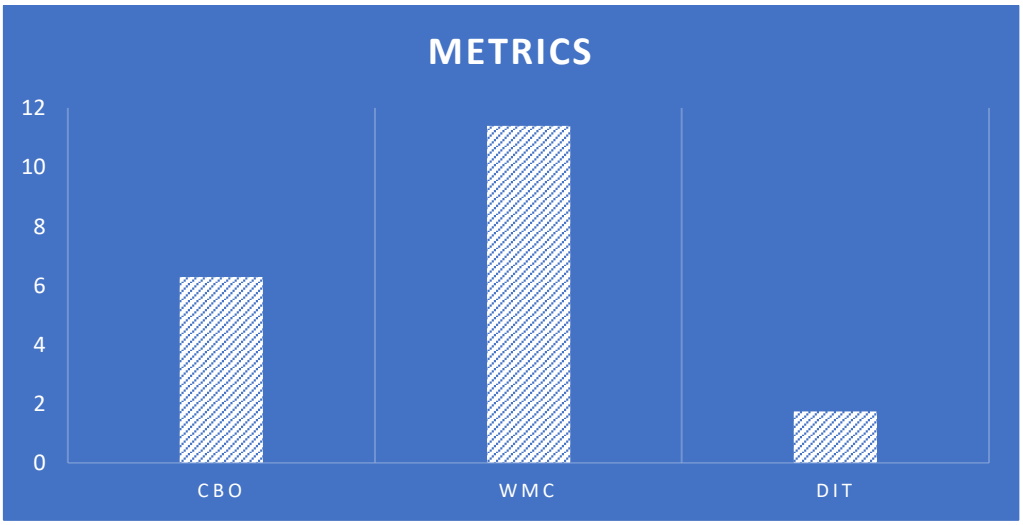
   According to the Depth of Inheritance Tree (DIT) metric, which is only 1.4, the project has a shallow hierarchy. However, by alone, this measurement might not provide a complete picture of the project's maintainability.

2. **What correlation exists among the number of classes and the C&K measures chosen to evaluate maintainability?**

The findings demonstrate a relationship between class size and maintainability, which is supported by the project's high WMC and CBO scores.

However, it should be noted that the C&K metrics used in this study only give a broad picture of maintainability and that other factors, such as code understanding, adaptability, and documentation, should also be taken into account when assessing software maintainability.

JUnit Project:



The answers to questions laid in GQM approach in Section are analyzed as follows:

1. **Is there a correlation between the number of a class and how easily it can be maintained?**

This project has a subpar Weighted Methods per Class (WMC) of 11.28 when compared to similar projects. Less intricate projects tend to include fewer classes to manage, which is good for maintainability.

The code is simple to maintain with a Coupling Between Objects (CBO) score of 6.19 since there aren't many links between classes.

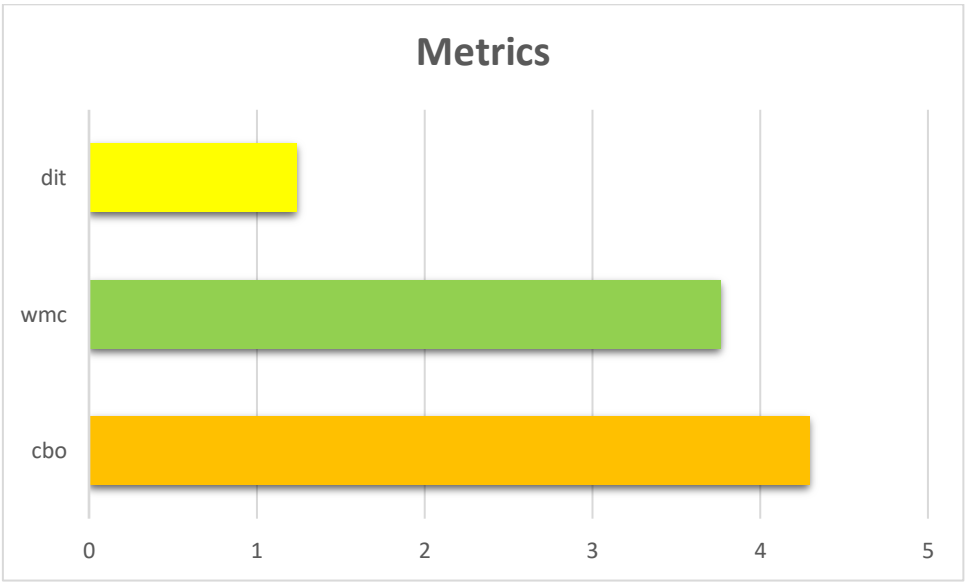
There aren't many inherited components in the project, according to its minimal Depth of Inheritance Tree (DIT), which is 1.68. As a byproduct, this might aid in making the code more manageable and easy.

2. **What correlation exists among the number of classes and the C&K measures chosen to evaluate maintainability?**

The project's relatively low WMC and CBO ratings suggest a potential link between maintainability and class size.

Nevertheless, it should be noted that the C&K measures used in this study only give an indication of maintainability and that other factors, such as code understanding, adaptability, and documentation, should also be taken into account when evaluating software maintainability.

Spring main:



The answers to questions laid in GQM approach in Section are analyzed as follows:

1. **Is there a correlation between the number of a class and how easily it can be maintained?**

   When compared to similar programs, the project has a subpar Weighted Methods per Class (WMC) of 3.67 on average. As a result, it's probable that the project's low level of complexity due to the smaller class size will improve its maintainability.

   The code is simple to maintain with a Coupling Between Objects (CBO) score of 4.17 since there aren't many links between classes.

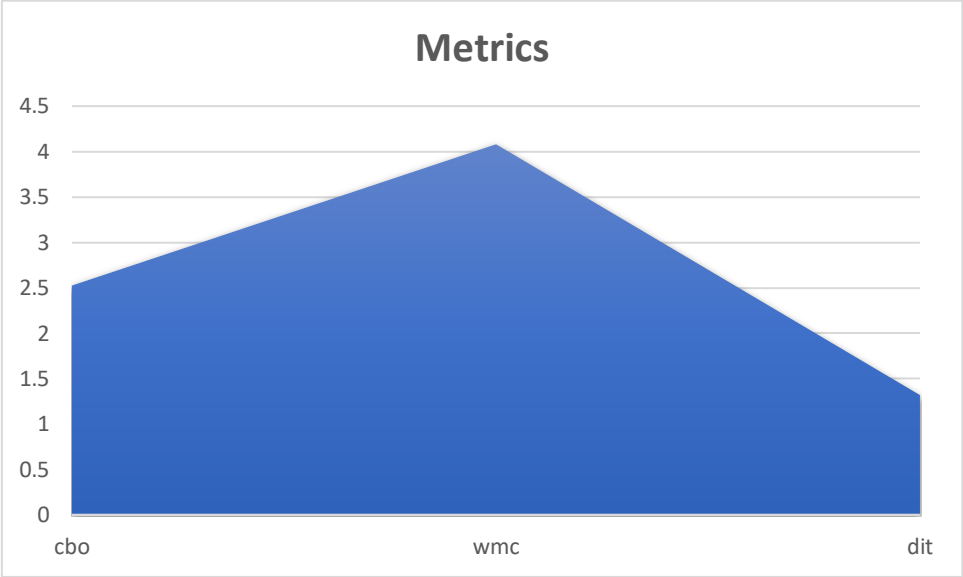
   A low Depth of Inheritance Tree (DIT) of 1.18 indicates that there aren't many inherited components in the project. This may also assist to simplify and manage the code, which is an added benefit.

2. **What correlation exists among the number of classes and the C&K measures chosen to evaluate maintainability?**

   The project's relatively low WMC and CBO ratings suggest a possible relationship between maintainability and class size.

   It should be noted, however, that other factors, such as code accessibility, flexibility, and needs, should also be taken into account when evaluating software maintainability and that the C&K metrics utilized in this research only provide a partial perspective of maintainability.

Hibernet-ORM:



The answers to questions laid in GQM approach in Section are analyzed as follows:

1. **Is there a correlation between the number of a class and how easily it can be maintained?**

   When compared to comparable projects, the project has a subpar Weighted Methods per Class (WMC) of 4.11 on average. As a result, it's feasible that the project's lack of complexity as a result of the smaller class size will improve its maintainability.

   Because there aren't many connections between classes, the code has a Coupling Between Objects (CBO) score of 2.49.

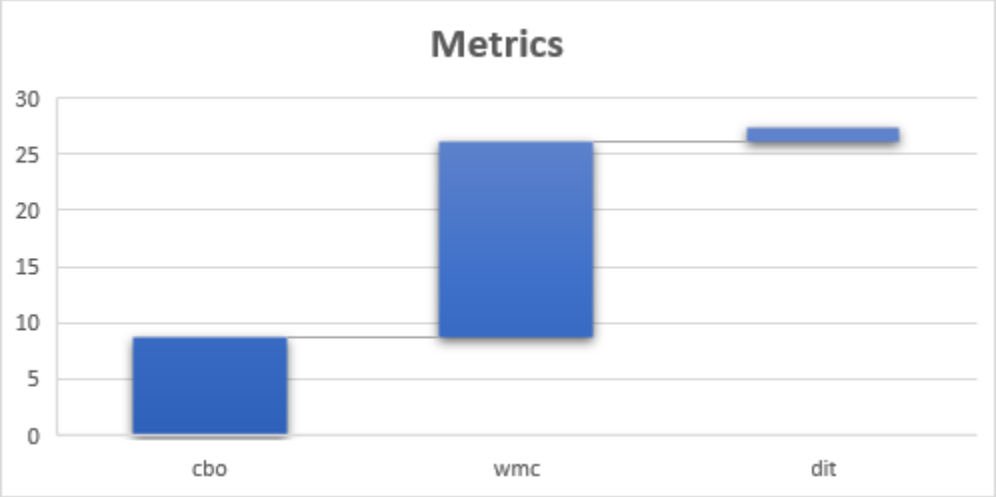
   The Depth of Inheritance Tree (DIT) value for the project, which is 1.29, shows that there aren't many inherited sub-components.

2. **What correlation exists among the number of classes and the C&K measures chosen to evaluate maintainability?**

The results indicate that the high class sizes utilized in the analysis may have contributed to the project's poor WMC and CBO values.

However, when examining software maintainability, code clarity, modularity, and documentation should also be taken into account; the C&K scores in this study only give a partial picture.

Joda Time:



Using the wvp-GB28181-pro project's CK-Code metric data, we can do the following analysis of class size and software maintainability:

When designing software, how should one think about factors like class size?

According to Weighted Methods per Class (WMC), the value of this project is very high at 16.95 The increased class size is likely to increase the project's complexity, making it more difficult to implement changes in the future.

An 8.69 result for the Coupling Between Objects (CBO) metric indicates that there are many links between classes, which makes it more difficult to change the code.

The project has a rather shallow family tree, as measured by the Depth of Heredity Tree (DIT) metric, which only returns a value of 1.32. The code may become less difficult and easier to maintain as a result of this.

How does the number of students in a class affect the reliability metrics (C&K) that are used?

The high WMC and CBO values for this project and the findings suggest a possible relationship between class size and maintainability. Only 50% of maintainability is shown by the C&K ratings in this research. When keeping software running, it's important to pay attention to the code's readability, organisation, and documentation.

Findings

The CK-Code metric tool allows us to analyze the five projects using the GQM method. The analysis based on results are:

- **Implications for software maintainability of increasing class sizes:**

  The results show that larger class sizes most likely caused higher average WMC values for the tasks. Due to the added complexity, larger class sizes may negatively affect the maintainability of software.

  However, the CBO and DIT project values are typically low to average, which is a sign of little interclass interaction and a low level of inheritance. If this is put into practice, the potential detrimental effects of larger class sizes on maintainability might be diminished.

- **Class size and a few key C&K measures for evaluating maintainability:**

  The projects with larger class sizes had relatively high WMC scores, which may be a sign that maintainability and class size are related.

  The research evaluated maintainability exclusively using the C&K measures, hence the results are plainly constrained. Additional elements like readability of the code,

functionality, and documentation must be taken into account in order to conduct an appropriate analysis of the software's maintainability.

Summary:-

The observed courses seem to have a moderate amount of complexity, with a mean WMC. Maintainability may be affected by the classes' moderate **complexity** since more complexity might make them harder to comprehend and alter.

Message Interactions: The observed classes have a modest amount of method interactions, with an average RFC value . This suggests that the classes are exposed to a moderate volume of incoming and outgoing messages, which may increase their complexity and create additional upkeep issues.

The average values for WMC and RFC suggest that the complexity and number of method interactions in the studied classes may provide some difficulties in terms of **maintainability**. In general, larger values for either measure imply a decline in maintainability since they point to increased complexity and possible challenges when modifying the classes.

While exact values for **DIT** are not supplied, it is possible that the existence of inheritance hierarchies will have an effect on maintainability. Complexity and dependencies introduced by several inheritance levels might make it more difficult to maintain the observed classes.

In conclusion, the data suggests that the classes' complexity, as evaluated by WMC and RFC, may have consequences for their maintainability. Maintenance activities may need considerable thought and preparation due to the system's moderate complexity and the moderate number of method interactions. Inheritance hierarchies also have the potential to add complications that detract from maintainability. A deeper knowledge of the connection between these indicators and software maintainability may be attained by further research and investigation of a bigger dataset.

5)Conclusion

According to the CK-Code metric implementation's findings, a program's maintainability may be affected by the number of classes it contains. This is shown by the fact that programs with bigger class sizes tended to have higher WMC values. This indicates that the code may get more complicated and difficult to comprehend and alter as the number of classes increases.

Keep in mind that when evaluating software's maintainability, the C&K metrics used in this study only provide a partial view. Consideration should also be given to the code's readability, organisation, and documentation. Multiple metrics and research methods are required to provide a complete picture of the software's quality.

The investigation's findings imply that the effect of class size on maintainability may vary from one project to the next. Common langs master and JUnit, for instance, both had larger class sizes while having comparably low CBO and DIT values. This strategy may help mitigate the negative consequences of increased class size on maintainability. The reduced class sizes and lesser number of classes in spring main and hibernate ORM may contribute to their excellent maintainability.

Joda-time's maintainability is worse than average because of its high WMC, high CBO, and weak DIT scores, in comparison to the other projects. This might be because of the increased complexity of the code and the bigger number of classes.

The findings of this research indicate that the size of a class may affect the software's ability to be maintained; nevertheless, there are many other aspects to consider. The quality of a programme can only be fully captured by using a wide variety of indicators and analytical techniques. Additional factors, such as code readability, modularity, and documentation, must be considered when evaluating maintainability.

References

Lanza, M., & Marinescu, R. (2006). Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems. Springer.

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

Ferreira, K., Bigonha, M., Bigonha, R., Mendes, L., & Almeida, H. (2011). Identifying thresholds for object-oriented software metrics. Journal of Systems and Software, 84(4), 620-632