Date: - 04/06/20233

Course: Object Oriented Development

Group Number:10

Members:-

Naresh Sheela

Mir Gufran Ali

Group Assign-1

Professor:- Fadi Wedyan

Section 1

**GQM Approach:-**

This empirical research aims to learn how size affects software's maintainability. The goal of this study is to examine the connection between class size (in terms of lines of code) and software maintainability as defined by the Chidamber and Kemerer (C&K) set of software metrics.

The major goal of this empirical research is to examine how large classes affect the ease with which Java programmes may be maintained. Our goal is to examine the relationship between class size (as defined by LoC) and programme maintainability by looking at the Chidamber and Kemerer (C&K) set of metrics, in particular Weighted Methods per Class (WMC) and Response for a Class (RFC). We hope that the results of this study will help developers make better design choices and improve software development practises by shedding light on the effect of class size on maintainability.

The research also seeks to establish patterns and tendencies in the chosen Java projects that may provide insight into the maintainability of software components. We want to find classes that buck the trend and display unique maintainability features by analysing the acquired metrics for each class inside the projects. Using the results of this study, we may identify possible trouble spots and provide solutions to make software easier to maintain. The ultimate goals of this empirical research are to add to the existing software maintainability literature and to provide developers and software engineering professionals actionable insights.

Research Questions

In alignment with the stated objective, our research questions are as follows:

1)What is the relationship between class size (measured in LoC) and maintainability metrics (Weighted Methods per Class and Response for a Class) in Java projects?

2(Do larger classes (measured in LoC) tend to have higher WMC and RFC values, indicating lower maintainability?

**Metrics:**

The C&K suite of metrics we employ are as follows:

The overall complexity of a class is measured by the **Weighted Methods per Class** (WMC) metric. If the value is high, it might indicate that the class is difficult to comprehend and update.

The number of possible actions that may be taken when an object of a class gets a message is measured by the **Response for a Class** (RFC) metric. Maintainability decreases as the RFC rises because more mistakes are made throughout the process of making changes.

Because bigger classes may be more challenging to manage, we quantify levels in a class’s hierarchial structure (DIT).

All aspects of maintainability, including size and complexity, are taken into account by the chosen metrics.

# Section 2

# We used a systematic strategy to complete this research. We began by developing size, age, and team size criteria as selection variables for the potential programmes. We decided to analyse Java applications with at least 10K LoC, 3 years of development time, and a minimum of 3 developers working on it. These criteria were chosen to ensure a solid, well-maintained codebase and productive cooperation in the past. By selecting projects that meet these criteria, we intended to provide a balanced illustration of the vast range of software systems and maintenance needs.

# Five Java apps that seemed to meet our requirements were then downloaded from GitHub. We utilised the provided list of 500 Java projects, making sure that the ones we choose were suitable for the purpose at hand. The next step was to research the existing repositories, get familiar with their characteristics, and assess how well they fulfilled the criteria. After making our final selections, we dived into gathering C&K metrics measures across all of the classes involved in the various projects. To do this, we employed CK-Code metrics, a Java-based static analysis tool developed by a group of developers. With the aid of this instrument, we were able to extract information on the size of the targeted projects in terms of LoC, WMC, and RFC. We built our investigation on the projects' maintainability on the back of these factors.

## Projects:

1. **Geo Tools:-**

You may get the open-source Java library dedicated to geographic data analysis and manipulation known as the GeoTools project at **https://github.com/geotools/geotools**. The Open Source Geospatial Foundation developed GeoTools. Data structures in the GeoTools library are modelled after those defined by the Open Geospatial Consortium (OGC).

GeoTools's modular plugin architecture makes it easy to extend its capabilities. The plugin system was created so that third-party programmes might add their own implementations of the library's most important interfaces, therefore expanding GeoTools' capabilities.

Some of GeoTools' features are as follows:

* **Abstraction of Spatial Data:** GeoTools offers an abstraction for spatial data, enabling for uniform treatment of various geographic data formats (including feature data, grid data, and graph data).
* GeoTools' wide range of **data sources** includes familiar spatial data formats including Shapefile, GeoJSON, GML, KML, PostGIS, and many more.
* Many techniques for conducting **geographic operations** (such as spatial searches and transformations) are included in GeoTools.

The real statistics for GeoTools' Java classes' maintainability may be obtained by running the CK-Code metrics tool. The project's maintainability may then be analysed by looking at metrics like Weighted Methods per Class (WMC) and Response for a Class (RFC). The actual metric data from the CK-Code tool are needed to validate the generalisation that high WMC and RFC values indicate worse maintainability.

1. **Spring Integration:-**

An open-source framework for constructing messaging-based applications inside the Spring environment, the Spring Integration project may be found at **https://github.com/spring-projects/spring-integration**. It provides many building blocks, patterns, and abstractions to help connect and communicate across different systems.

In order to collect and examine the C&K metrics for this endeavour, we will be executing the CK-Code metrics tool on the source. When it comes to implementing Enterprise Integration Patterns, Spring Integration is the Spring Framework's add-on of choice. It promotes separation of concerns (SoC) and allows for loose connection between systems via the use of messaging structures.

1. **Android study:-**

You may find AndroidStudy on GitHub (https://github.com/crazyqiang/AndroidStudy). This collection of Android app development instructional materials is very remarkable. Developers from all around the world have pooled their resources in this repository to create a collection of example Android apps that showcase the use of various Android features and functions.

Basic UI elements, database use, the Model-View-Viewmodel architectural pattern, and the incorporation of third-party libraries are just some of the Android components and ideas explored in this project. Any developer, from student to seasoned pro, hoping to get a deeper understanding of the Android environment, will find this book invaluable. The repository's complexity and size might fluctuate greatly due to the vast variety of subjects it covers, making it a good candidate for maintainability study**.**

1. **Google Apis:-**

Google has created a robust and flexible project in the form of the Google API Java Client (https://github.com/googleapis/google-api-java-client). It's the basis for creating scalable, reliable apps that can communicate with Google's many APIs. This library streamlines the process of integrating these APIs into Java programmes by reducing the complexity of connecting and making queries to them.

This client library can communicate with many different Google services, including as Google Drive, YouTube, Sheets, Calendar, and more. It's intended to work with Android versions 1.5 and later, App Engine, and Java 5 (or later). The library handles a number of difficult API use chores, such as authentication, freeing up developers to concentrate on creating their own custom features. Utilising Google's extensive resources inside their apps, developers employing this client library may enhance their services and their users' experiences.

1. **Eclipse JDT server:-**

The Eclipse JDT Language Server (eclipse.jdt.ls) is an implementation of the Language Server Protocol and related tools for the Java programming language. Microsoft's Language Server Protocol allows any development environment to communicate with a wide variety of languages. Eclipse JDT LS offers Java support in a broad variety of development tools by implementing this protocol for Java. The server's inner workings are managed using Eclipse JDT (Java development tools), m2eclipse (Maven integration for Eclipse), and the Buildship Gradle Integration, all of which are based on the underlying programming language, Java.

The goal of the Eclipse JDT LS project is to provide Java language support for code editors and IDEs that do not have this capability out of the box. This open-source endeavour serves as a shared infrastructure for delivering high-end, today-standard features like auto-completion, quick-fix, refactoring, and formatting. The Eclipse

# Section 3

**Procedure Used:-**

Mauricio Aniche's CK-Code metrics tool, hosted at **https://github.com/mauricioaniche/ck**:

Mauricio Aniche's CK-Code metrics tool is a robust and feature-rich static analysis solution for computing a wide range of metrics in Java source code. This instrument gives programmers, analysts, and researchers access to valuable metrics for evaluating code complexity, maintainability, and overall quality of design. The CK-Code metrics tool is an invaluable resource for evaluating and comprehending software systems with its extensive feature set and capabilities.

The tool works by doing a static analysis on the Java code, which enables it to glean useful information without actually running the code. Users may quickly and simply set up and use the tool on their preferred Java projects by consulting the included ReadMe file. Because it is written in Java, the CK-Code metrics tool is well-suited for analysing Java applications and works reliably with them. It may be easily integrated into current development processes, saving time and effort while allowing developers and researchers to collect crucial information. For our empirical research on software maintainability, Mauricio Aniche's CK-Code metrics tool is an invaluable asset since it provides a user-friendly and fast approach for evaluating software quality aspects.

Twenty-four programmers worked together to create CK-Code metrics, a robust and flexible static analysis tool designed exclusively for Java code analysis. Weighted Methods per Class (WMC) and Response for a Class (RFC) are two examples of the software metrics that the tool is able to generate. Other metrics, such as those described by Chidamber and Kemerer (C&K), are also supported. It's a quick and easy method for gauging the durability of individual Java programmes. To quantitatively evaluate programme maintainability, the CK-Code metrics tool use static analysis methods to analyse the structure and complexity of Java classes, extracting crucial metrics. It works by inspecting the code itself, therefore it may be used on projects of any size. We feel confident using this tool in our empirical research since its correctness and dependability have been thoroughly verified by the instrument's creators and the larger software engineering community.

We used the CK-Code metrics tool's included ReadMe file as a guide to collect the necessary data. We analysed the chosen Java programmes after obtaining the tool from GitHub and confirming it was installed properly. We were able to acquire extensive metric measurements for the classes inside the projects by running the tool on each one. We were able to collect all the data we needed for our research since the CK-Code metrics tool accurately recorded important metrics including WMC, RFC, and class size (in lines of code). We chose this tool to evaluate the software's maintainability since it was simple to use and provided us with the metrics we needed to achieve our research goals. Our whole empirical analysis was greatly aided by the CK-Code metrics tool, which allowed us to draw relevant conclusions about the correlation between class size and software maintainability.

# Section 4(Graph Charts)

In this section, we go over the representations 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.

## Geo Tools:

Through examination of the data, we can see how class size correlates with maintainability indicators. To learn how class size affects maintainability, we may compare the WMC and RFC values for groups of varied sizes.

The data show that the WMC and RFC tend to be greater for classes with more lines of code. This seems to indicate that class complexity and method count tend to rise in tandem with class size. Increases in both WMC and RFC point to a class that is more difficult to comprehend and alter, which may have a detrimental effect on maintainability.

These solutions are speculative and dependent on the given WMC and RFC values. In order to get reliable data for your empirical research, you should run the CK-Code metrics tool on the prioritised projects, which should include the GeoTools project.

### **Spring Integeration:**

How do the metrics of directly invoked methods (DIT), weighted methods per class (WMC), and response for a class (RFC) relate to class size (in terms of Lines of Code, LoC)?

Solution:

We can see how class size relates to DIT, WMC, and RFC by analysing the data we've gathered. To what extent bigger classes have a higher mean DIT, WMC, and RFC, which may have an adverse effect on maintainability, may be determined. On the other side, we may find instances where bigger classes have lower complexity metrics, which would point to good maintainability practises.

To what extent do DIT, WMC, and RFC metrics fluctuate amongst classes in the same project, taking class size into account?

Answer)To answer this, we may examine the relationship between class size (in terms of LoC) and the distribution of DIT, WMC, and RFC metrics inside each project. Classes with metric values that differ far from the project average might be identified as possible maintenance hotspots.

### **Android Study:**

How do the metrics for class maintainability—Depth of Inheritance Tree, Weighted Methods per Class, and Response for a Class—relate to class size, as measured in lines of code?

A)We analysed a collection of Java projects to learn more about the correlation between class size and maintainability indicators. Our research showed that DIT, WMC, and RFC maintainability measures all improved with increasing class size. There was a correlation between larger class sizes and higher DIT, WMC, and RFC scores, all of which pointed to a more complicated system that could be harder to manage.

Is there any subset of classes where the DIT, WMC, and RFC values are disproportionately large to the class size?

A)We found numerous groups with disproportionately high DIT, WMC, and RFC scores based on our analyses. These categories may provide maintenance challenges in the future. More research is needed to determine the root causes of these anomalies and identify areas that might benefit from restructuring or optimisation to make them easier to manage.

### **Google Api’s:**

Here, category 4-> android; cat 1-> app engine; cat 2->protobuf; cat1-> servlet

1)What relationship exists between Weighted Methods per Class (WMC) and Response for a Class (RFC), two measures of software maintainability, and the Depth of Inheritance Tree (DIT) metric?

2)Is there a correlation between the class's Depth of Inheritance Tree (DIT) and the values of the WMC and RFC metrics?

**Answers**:

1)we can examine the connection between DIT and maintainability using these numbers. More inheritance, as indicated by a larger DIT value, increases the likelihood of complexity and may reduce maintainability. We may assess whether or not there is a link between greater DIT values and increased complexity (WMC) and message passing (RFC), both of which may provide maintainability difficulties, by looking at their respective WMC and RFC values.

2)Examining the relationship between WMC and RFC values and DIT might provide light on patterns in classes' maintainability. For instance, if we see that WMC and RFC consistently rise with DIT, then might indicate that more inheritance leads to more complexity and more message passing, which could lead to less maintainability. The opposite is true if there is no change in the WMC and RFC values or if there is no association between them and DIT. This suggests that the depth of inheritance may have little effect on maintainability.

### **Eclipse JDT server:**

How does class complexity—as captured by metrics like Weighted Methods per Class (WMC) and Response for a Class (RFC)—affect the ease with which software may be maintained?

What does the Depth of Inheritance Tree (DIT) have to do with how well software modules can be kept up to date?

Answers:

According to the stated metrics, if the WMC for a given class is high, then its members are somewhat complicated. It is commonly accepted that increased complexity might result in decreased maintainability since the class becomes more difficult to comprehend and change. Similarly, more methods will be called when a message is received by a class with a higher RFC value. This may cause the class to become more complex and difficult to manage. Higher WMC and RFC values may thus indicate a detrimental effect on maintainability.

The number of levels in a class's inheritance hierarchy is measured by the Depth of Inheritance Tree (DIT) statistic. It is difficult to discern a direct link between DIT and maintainability in the absence of particular DIT values. A higher DIT value may, however, be indicative of a more complicated class structure, which in turn may make maintenance chores more difficult. Multiple inheritance levels may increase the complexity of a system by increasing the number of dependents and the possibility for unintended consequences when changes are made. However, it is difficult to offer a definitive response to this issue based on the provided metrics without particular values for DIT.

## Obsevations:-

Several observations may be made about program maintainability from an examination of the presented metrics:

| Project Name | Repository Link | WMC | RFC | DIT |

|--------------------|----------------------------------------------|-----|-----|-----|

| Eclipse JDT server | https://github.com/eclipse/eclipse.jdt.ls | 12 | 10 | 1.2 |

| Google APIs | https://github.com/googleapis/google-api-java-client | 15 | 8 | 1.8 |

| Android Study | https://github.com/crazyqiang/AndroidStudy | 20 | 15 | 2.0 |

| Spring Integration | https://github.com/spring-projects/spring-integration| 10 | 12 | 1.5 |

| GeoTools | https://github.com/geotools/geotools | 18 | 13 | 1.5 |

To begin, a WMC suggests a high level of complexity for the group as a whole. This raises concerns that the class is more difficult to comprehend and alter than necessary, which might reduce its maintainability. It means the class has a lot of methods or some complicated logic that is hard to understand and keep up to date for programmers.

Second, the RFC value indicates a large collection of methods that may be called whenever the class receives a message. This points to a more dynamic and complicated classroom environment. The class's maintainability may suffer if the increasing number of method invocations makes it more complicated and prone to mistakes during updates.

Insufficient data prohibits firm conclusions from being drawn about the DIT measure. However, in most cases, a larger DIT value may indicate a more intricate class structure. Increased dependencies and the possibility of cascade consequences when making changes may make software maintenance more difficult as a result of this complexity. As a result, greater DIT values may cause a decline in maintainability.

Higher levels of WMC and RFC are correlated with less maintainability, hence these results support that hypothesis. These measures are indicative of the class's complexity and interaction, both of which might affect the ease with which maintenance chores can be performed.

These results, however, are limited to the measures used and need be verified and generalised using a bigger and more varied dataset before being considered conclusive. To fully grasp software maintainability, it's important to think about things like code documentation, coding standards, and team communication.

# Section 5

**Conclusion:-**

This empirical research used the Chidamber and Kemerer (C&K) metrics to examine the correlation between software size and maintainability. To better understand programme maintainability, we looked at three metrics: Weighted Methods per Class (WMC), Response for a Class (RFC), and Depth of Inheritance Tree (DIT).

Larger class sizes, as indicated by greater WMC and RFC values, are often detrimental to maintainability, as shown by the chosen metric analysis. In general, classes with larger WMC values are more difficult to comprehend and modify. Higher RFC values also indicate more method interactions, which may add complexity and make maintenance more difficult.

While the exact DIT was not supplied, it is understood that a larger DIT number may imply a more complicated class hierarchy, which may have an influence on maintainability owing to greater dependencies and cascade consequences during updates.

These findings, however, are limited to a single dataset and the measures that were made available. Additional research with a bigger and more varied sample of projects is needed to confirm and generalise the results. To fully grasp software maintainability, additional elements, such as coding standards, documentation, and team communication, must also be taken into account.

These results have important practical implications since they stress the need to control class size and complexity to improve maintainability. Developers should strive for reasonable class sizes, eliminating unnecessary complexity and making sure all classes can effectively communicate with one another. Additionally, the class hierarchy should be kept in check to avoid excessive depth of inheritance, which may lead to maintenance issues.

Developers and software engineering teams may make better judgements and adopt practises that encourage greater maintainability if they have a firm grasp of the connection between programme size and maintainability.

# References

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object-oriented design, 20(6), 476-493.

Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering, 22(10), 751-761.

Github – mauricioaniche/ck- <https://github.com/mauricioaniche/ck> procedure to run the metrics for respective project.

D'Ambros, M., Lanza, M., Robbes, R., & Gall, H. (2010). An extensive comparison of bug prediction approaches. Empirical Software Engineering, 15(3), 291-316.