Object Oriented Development

Assignment 1

Group 3

April 8, 2023

Table of Contents

[**Section 1** 2](#_Toc131544498)

[**GQM Approach** 2](#_Toc131544499)

[**Objective:** 2](#_Toc131544500)

[**Question:** 2](#_Toc131544501)

[**Metrics:** 3](#_Toc131544502)

[**Section 2** 3](#_Toc131544503)

[**Criteria for Selection** 3](#_Toc131544504)

[**Studied Programs:** 3](#_Toc131544505)

[**Program 1:** 3](#_Toc131544506)

[**Program 2:** 4](#_Toc131544507)

[**Program 3:** 4](#_Toc131544508)

[**Program 4:** 5](#_Toc131544509)

[**Program 5:** 5](#_Toc131544510)

[**Section 3** 6](#_Toc131544511)

[**Tool Used** 6](#_Toc131544512)

[**Section 4** 7](#_Toc131544513)

[**Results:** 7](#_Toc131544514)

[**Analyzing 1st Project:** 7](#_Toc131544515)

[**Analyzing 2nd Project:** 8](#_Toc131544516)

[**Analyzing 3rd Project:** 9](#_Toc131544517)

[**Analyzing 4th Project:** 10](#_Toc131544518)

[**Analyzing 5th Project:** 11](#_Toc131544519)

[**Section 5** 13](#_Toc131544520)

[**Conclusion** 13](#_Toc131544521)

[**References** 14](#_Toc131544522)

# **Section 1**

# **GQM Approach**

It is possible to specify and quantify software quality using a Goal-Question-Metric (GQM) method. It is a method for making sure a software project has well-defined objectives and accurate metrics by which one can evaluate its performance. Here are some of GQM's fundamental parts:

* **Objectives:** The goals of the software development project are established as the first stage in the GQM methodology. These objectives should be well-defined and measurable. The goals should be realistic and in line with the overall objectives of the company.
* **Questions:** The following stage, after deciding what needs to be done, is to recognize the problems that need fixing. The questions must be specific, measurable, and timely. It is vital that these questions have quantifiable answers.
* **Metrics:** Determining what data points will be used to evaluate success is the last stage in the GQM method. There should be a clear connection between the questions being posed and the metrics being used. The measures used must also be in harmony with the project's intended outcomes.

Using the GQM methodology, software development teams can be confident they are working toward the right objectives, are investigating the right questions, and are evaluating their progress with the right metrics. This method guarantees that the project is in line with the organization's goals and contributes to a higher standard of software development.

Based on the GQM (Goal-Question-Metric) approach, the following are the objectives, questions, and metrics for the empirical study:

## **Objective:**

To measure the maintainability of randomly selected software components.

### **Question:**

What is the level of maintainability of the randomly selected software components based on the C&K metrics?

### **Metrics:**

* ***Coupling Between Objects*** is a measure used to ascertain the extent to which one class depends on the capabilities of other classes.
* ***Weighted Methods per Class:*** When measuring the number of methods in a class, the WMC metric considers the class's complexity.
* ***Depth of Inheritance Tree*** counts the levels of class nesting.
* ***Response for a Class*** measure specifies how many methods can be called when communicating with an object of a given type(*Project Metrics Help - Chidamber & Kemerer Object-Oriented Metrics Suite*, n.d.).

The above objectives, questions, and metrics follow the GQM approach, which helps to ensure that the study's goals are clearly defined, and the metrics used are appropriate for measuring the desired outcomes. These objectives, questions, and metrics will guide the study and provide a clear understanding of what needs to be measured and analyzed to achieve the research goal.

# **Section 2**

## **Criteria for Selection**

To select the projects for analysis, I set the criteria that the programs must be at least 5 years old, but not older than 5 years, to ensure that they have undergone various maintenance activities. I also looked for programs that are actively developed and have a significant user base to ensure that they are relevant and important projects.

After considering various open-source software projects, I selected five programs for analysis: Ghidra, Minestom, Halo, VirtualXposed, and Akhq-dev. Each of these applications has a sizable user base and has been receiving regular updates from a diverse group of programmers over the past few years. The selected programs meet the criteria for being actively developed, widely used, and not too old. They represent a diverse range of software types and are expected to provide useful insights into the maintainability of open-source software projects.

## **Studied Programs:**

### **Program 1:**

The National Security Agency's (NSA) Research Directorate created and continues to update a software reverse engineering system called Ghidra. For users interested in analyzing compiled code on different platforms, it offers a full suite of software analysis tools. The system supports a wide variety of operations, including disassembly, assembly, decompilation, graphing, and scripting. Ghidra can be executed either interactively by the user or automatically, and it supports multiple instruction sets and executable formats for various CPU architectures. Users can also create their own Ghidra programs and extensions in Java or Python. To aid the NSA in their cybersecurity goal, Ghidra serves as a flexible and adaptable SRE research platform. Malicious code analysis and network/system vulnerability detection have benefited from its use.

Dolphin Scheduler is an advanced data workflow orchestration tool that can manage data pipelines with complex task dependencies. It comes preconfigured with a robust user interface and a wide range of jobs. It has four distinct deployment options and several process construction approaches. The platform supports horizontal growth and has a high availability rate. It is fast, and it can accommodate multiple tenants. Dolphin Scheduler facilitates straightforward orchestration of processes and dependencies across a wide range of job types. Tasks in progress can be seen in real time, and there are tools available for maintaining track on completion rates and reviewing past workflows. The platform is also internationalized and includes permission management at the project, resource, and data source levels. In addition, Dolphin Scheduler is built for the cloud and can orchestrate processes across multiple cloud providers and data centers.

### **Program 2:**

With Minestom, developers do not have to depend on Mojang's code to create their own Minecraft server software. A Minestom-based server is not like Mojang's basic server in that it has no functionality pre-installed. Instead, the API provided by the framework makes it much easier to create specialized Minecraft server software.

Minestom's application programming interface (API) is strictly for programmers and should not be used by regular users. Minestom cannot substitute Bukkit, Forge, or Sponge because it does not support their application programming interfaces.

### **Program 3:**

Everyone has free access to Halo, a powerful and intuitive website-building platform. It provides a wide variety of tools that speed up, streamline, and simplify the process of creating a website.

The Halo group has developed this initiative, and it is based on sound open-source and collaborative programming practices. Users can make their own webpages without needing any specialized information or coding skills.

With Halo, users can quickly publish and update material, modify design templates, and build mobile-friendly, aesthetically pleasing webpages. It also works with a broad variety of extensions, which we can use to extend our site's capabilities. Halo makes creating webpages easier and faster for developers of all levels.

### **Program 4:**

We can use Xposed Modules on our Android device with VirtualXposed, a user-friendly program built on VirtualApp and Epic that allows us to do so without having to root our device, unlock our bootloader, or install a modified system image. Android 5.0 through Android 10.0 can all be used with this app.

VirtualXposed's only two limitations are its inability to change the system, which means any Module that modifies the system will not work, and its lack of support for resource hooks. As a result, theming plugins that rely on resource callbacks are incompatible with this software.

This GitHub project provides a practical workaround for people who want to use Xposed Modules on their Android device but do not want to deal with the inconvenience of rooting and activating their device.

### **Program 5:**

We can manage Kafka elements like topics, topic data, buying groups, the schema registration, connections, and more with the help of Akhq, a Kafka graphical user interface. This program has a simple graphical user interface (GUI) for controlling and monitoring Kafka groups.

The developers are constantly working to enhance the functionality of their GitHub-hosted project for the benefit of its users. Akhq is a helpful tool for companies of all sizes due to its intuitive interface and ability to facilitate the creation, execution, scheduling, and monitoring of complicated pipelines.

In addition, the compatibility between Akhq and Apache Kafka makes it an attractive choice for companies that are already using Kafka for their messaging needs. It is the best option for handling Kafka groups and pipelines because it is scalable, reliable, and adaptable.

Table summarizing the key attributes of the projects:

|  |  |
| --- | --- |
| **Project Name** | **Key Attributes** |
| AKHQ | Open-source, web-based interface, easy to deploy, customizable, multi-cluster support, security features, active development |
| Ghidra | Open-source, cross-platform, disassembler and decompiler, scripting support, collaboration features, customizable UI, extensible, advanced analysis capabilities, active development |
| Halo | Open-source, cross-platform, easy to use, customizable, SEO-friendly, multi-language support, markdown editor, active development |
| Minestom | Open-source, high-performance, customizable, lightweight, multi-version support, API support, active development |
| VirtualXposed | Open-source, virtual environment, easy to use, customizable, multi-version support, security features, active development |

# **Section 3**

## **Tool Used**

The CK-Code information for the five selected Java apps was gathered using the GitHub application. There is a link to the program on GitHub (*GitHub - Mauricioaniche/Ck: Code Metrics for Java Code by Means of Static Analysis*, n.d.). Twenty-four Java developers collaborated on this tool for structural analysis of Java code and acquiring CK-Code information.

The ReadMe page in the GitHub repository was very helpful in understanding how to operate the program. All five projects used the software to collect CBO, WMC, DIT, and RFC values. The data analysis allowed us to draw conclusions about the projects' maintainability.

In a nutshell the GitHub measure tool CK-Code was used to assess the maintainability of the five selected Java projects. The tool served its purpose by furnishing us with data from which to deduce the projects' maintainability.

# **Section 4**

## **Results:**

### **Analyzing 1st Project:**

Four CK measures have numbers that can be analyzed:

* **CBO:** Each class in the chosen software components is linked to about three other classes, as indicated by the average CBO number of 4.12. This points to a program that is only moderately complicated and dependent.
* **WMC:** The average number of methods per class in the chosen software components is 17.13, which indicates an average degree of complexity according to the WMC metric.
* **DIT:** With an average value of 1.06, DIT indicates that each class in the chosen software components has a linear family tree with only one degree of inheritance.
* **RFC:** The average RFC value is 11.48, which indicates that the chosen software components have classes with a modest degree of responsibility and complexity in terms of the number of methods they can perform.

We can infer that the randomly sampled software components in the Ghidra project have an average degree of maintainability based on these average values. Metrics such as CBO and RFC point to some degree of complexity and reliance, while WMC and DIT point to a small degree of complexity and inheritance. These are just averages; some classes may have much greater or lower numbers for these metrics, which may influence the codebase's maintainability.

### **Analyzing 2nd Project:**

For the Minestom project, we have access to the following four CK metrics:

* **CBO:** The average number of CBO is 6.04, which indicates that each class in the chosen software components is linked to approximately 6 other classes. This points to code intricacy and dependencies that are moderately to extremely high.
* **WMC:** Each class in the chosen software components has a medium degree of complexity in terms of the number of methods it includes, according to the average value of WMC, which is 9.27.
* **DIT:** The average value of DIT for the chosen software components is 1.70, showing that on average, each class has a middling degree of inheritance structure with more than one level of inheritance.
* **RFC:** Each class in the chosen software components has a modest degree of responsibility and complexity in terms of the number of methods it can perform, as indicated by the average RFC value of 8.90.

These median numbers suggest that the software components in the Minestom project can be maintained with a reasonable amount of effort. Indicators like CBO and DIT point to a moderate to high degree of complexity and dependence, while indicators like WMC and RFC point to a moderate degree of complexity and responsibility. It is worth noting that these are simply median values; individual classes may have vastly different values for these measures, which could have an effect on the codebase's readability and maintainability.

### **Analyzing 3rd Project:**

Values for the following four CK measures can be analyzed:

* **CBO:** At a value of 7.94, CBO indicates that each class in the sampled software components is typically linked to around 8 other classes. This is indicative of a codebase with a high degree of complexity and interdependence.
* **WMC:** The average number of methods per class in the chosen software components is relatively low, as measured by the value of WMC, which is 5.41.
* **DIT:** With value of 1.21, DIT indicates that, on average, only one or two layers of inheritance separate each class in the chosen software components.
* **RFC:** With an average value of 16.50, RFC indicates that each class in the chosen software components has a high degree of responsibility and intricacy in regard to the total number of methods it can perform.

The Halo project has an equally complex and reliant codebase, based on these numbers, with classes that have a low to moderate degree of complexity according to of the number of methods they contain. Given the comparatively high RFC number and mostly level inheritance tree, it is possible that the codebase is complicated and poorly maintained.

### **Analyzing 4th Project:**

The VirtualXposed project's four CK indicators are as follows.

* **CBO:** The average CBO number is 6.78, which indicates that each class in the chosen software components is linked to roughly 7 other classes. This indicates a program that is only moderately complicated and dependent.
* **WMC:** Each class in the chosen software components has a medium degree of complexity because of the number of methods it includes, according to the average value of WMC, which is 12.76.
* **DIT:** With a mean value of 1.56, DIT indicates that the chosen software components have a shallow inheritance structure with one degree of inheritance for each class.
* **RFC:** The average RFC value is 14.64, which indicates that the chosen software components have classes with a relatively low level of responsibility and complexity relative to the total amount of methods they can perform.

It can be concluded from these median numbers that the software components in the VirtualXposed project are moderately maintainable. Metrics such as CBO and RFC point to some degree of complexity and reliance, while WMC and DIT point to a modest degree of complexity and inheritance.

### **Analyzing 5th Project:**

The Akhq-dev project's four CK metrics are as follows:

* **CBO:** The average CBO number is 8.09, which means that each class in the chosen software components tends to be connected to approximately 8 other classes. The software appears to be moderately complicated and dependent.
* **WMC:** At a mean value of 9.00, this metric indicates that the average class in the sampled software modules has a low complexity level in terms of the number of methods it implements.
* **DIT:** The average value of DIT is 1.29, showing that on average, only one or two layers of inheritance exist between classes in the chosen software components.
* **RFC:** With an average value of 14.85, RFC indicates that the selected software components have classes with a moderate degree of responsibility and complexity in respect to the total number of methods they can carry out.

These median numbers suggest that the software components in the Akhq-dev project are moderately easy to manage. Some degree of complexity and reliance is indicated by the CBO and RFC metrics, while a minimal degree of complexity and inheritance is indicated by the WMC and DIT metrics. However, as with any project, these measures should be considered alongside others, such as the project's unique objectives and needs, to provide an accurate representation of its maintainability.

Table showing summary of list of Projects with their C&K Metrices

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Project Name** | **CBO (avg)** | **WMC (avg)** | **DIT (avg)** | **RFC (avg)** |
| Ghidra | 4.12 | 17.13 | 1.06 | 11.48 |
| Minestom | 6.04 | 9.27 | 1.70 | 8.90 |
| Halo | 7.94 | 5.41 | 1.21 | 16.50 |
| VirtualXposed | 6.78 | 12.76 | 1.56 | 14.64 |
| Akhq-dev | 8.09 | 9.00 | 1.29 | 14.85 |

The average values of the four CK measures (Coupling Between Objects, Weighted Methods per Class, Depth of Inheritance Tree, and Response for a Class) across all five examined projects are summarized in the chart below. The maintainability of a codebase can be affected by several factors, and these measures provides information on those factors, including its complexity, interdependence, inheritance, and responsibility. Based on the data in the table, we can conclude that the maintainability of all five projects is about average, with some variation in complexity, dependencies, inheritance, and responsibility.

The graph that was created with CK measures shows that the five software projects were selected at random and have different degrees of maintainability. This information was obtained from the graph.

Because of their low CBO and RFC scores, as well as their low WMC and DIT values, the Ghidra and VirtualXposed systems are both simple to maintain (Khan et al., 2012).

Metrics from CBO and DIT suggest an average to high level of complexity and reliance for both the Minestom and Akhq-dev projects, whereas metrics from WMC and RFC indicate to a moderate degree of both complexity and responsibility for these projects.

In the codebase for the Halo project, the classes range from simple to complicated in terms of the number of functions that they have, which is what contributes to the complexity and dependence of the codebase. The comparatively high number of RFCs and the relatively consistent inheritance tree are two indicators that suggest that the software's maintainability may be negatively impacted by its complexity and responsibility.

# **Section 5**

## **Conclusion**

The CK metrics of the five selected software projects demonstrate that size can have a significant impact on maintainability.

Let us start by defining "size" in the context of our computing efforts. The number of methods and classes in the codebase, to be precise. We can put this into numbers using the WMC CK measure. (Weighted Methods per Class). The higher the WMC, the more complicated the class's methods are.

The least maintainable of these projects are the ones with the highest WMC ratings (Minestom and VirtualXposed). This could be because larger and more method-rich classes make it more challenging to understand and handle complicated and dependent systems.

However, the lower WMC numbers of Ghidra, Halo, and Akhq-dev make them more manageable. This suggests that classes with fewer methods are easier to understand and keep up to date.

When compared to other initiatives like VirtualXposed, Minestom had a much larger average amount of lines of code per class. Both initiatives were similarly simple to manage, however, when measured against the CK criteria.

The difficulty of maintaining software may be affected by factors such as the code's complexity, the number of its connections, and the depth of its inheritance structure. For this reason, evaluating the maintainability of software projects requires looking at more than just the number of classes involved.

However, it's important to note that size alone cannot fully determine the maintainability of a software project. Other factors such as the design patterns, coding standards, and documentation can also have an impact on maintainability.

Thus, the size of a software project can have a significant impact on its maintainability. Projects with larger classes and methods may be more complex and harder to maintain, while projects with smaller classes and methods may be easier to maintain. It's important to consider the size of a project along with other factors when assessing its maintainability.

# **References**

*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

Khan, Y. A., Elish, M. O., & El-Attar, M. (2012). A systematic review on the impact of CK metrics on the functional correctness of object-oriented classes. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7336 LNCS*(PART 4), 258–273. https://doi.org/10.1007/978-3-642-31128-4\_19/COVER

*Project Metrics Help - Chidamber & Kemerer object-oriented metrics suite*. (n.d.). Retrieved April 4, 2023, from https://www.aivosto.com/project/help/pm-oo-ck.html