**Group Assignment 1**

**Subject:** Object Oriented Development

**Group Name:** New Group 10

**November 5, 2023**

Table of Contents

[Section 1 2](#_Toc150112298)

[Applying GQM Approach 3](#_Toc150112299)

[Section 2 5](#_Toc150112300)

[Selected Projects: 6](#_Toc150112301)

[Section 3 9](#_Toc150112302)

[Tool Description 9](#_Toc150112303)

[Section 4 10](#_Toc150112304)

[Results: 11](#_Toc150112305)

[**CatVodTVSpider Project:** 11](#_Toc150112306)

[**SmartTubeNext Project:** 13](#_Toc150112307)

[**cwa-server Project:** 14](#_Toc150112308)

[**RxJava Project:** 15](#_Toc150112309)

[**wvp-GB28181-pro Project:** 17](#_Toc150112310)

[The Java code metrics of the selected projects were calculated using CK-Code and are presented in the table above. 18](#_Toc150112311)

[Findings 18](#_Toc150112312)

[**Overall findings for each project:** 19](#_Toc150112313)

[Section 5 20](#_Toc150112314)

[Conclusion 20](#_Toc150112315)

[References 21](#_Toc150112316)

# Section 1

The subsequent illustration showcases the utilization of the GQM (Goal Question Metric) methodology for the identification of pivotal metrics.

* Please document the anticipated results of the process.
* Please provide the list of inquiries that must be addressed in order to achieve the desired objectives.
* Trace the steps back to the stated objectives.

The GQM (Goal-Question-Metric) approach facilitates the software development teams in enhancing their planning efficiency, allocating priorities to activities, and monitoring their collective advancement towards a common aim. The GQM strategy has been effectively adopted across several organizational levels, including corporate, project, and procedural domains, with the aim of streamlining decision-making processes, improving overall performance, and optimizing resource allocation.

The GQM technique can be employed to initially ascertain the objective of the study, subsequently formulate research inquiries, and subsequently construct metrics for assessing the maintainability of software. By employing the Goal-Question-Metric (GQM) technique, researchers can effectively ascertain that their study possesses well-defined aims and measurable outcomes, that their inquiries are appropriately aligned, and that the data collected is both dependable and capable of informing actionable insights. By employing this approach, it also guarantees the reliability and practicality of their data.

## Applying GQM Approach

**Objective:**

The primary objective of this empirical inquiry is to utilize the C&K measures in order to examine the extent to which class size influences software maintainability.

**Questions:**

Does a correlation exist between the numerical designation of a class and its level of maintainability?

What is the relationship between the number of classes and the selection of C&K measures for assessing maintainability?

**Metrics:**

Based on the above objectives and research inquiries, the subsequent C&K metrics may be chosen as suitable measures for evaluating software maintainability:

* **Weighted Methods per Class,**

Class complexity is quantified by taking into account two factors: the total number of methods and the average complexity of each individual method.

* **Depth of family Tree (DIT):**

The DIT metric quantifies the hierarchical levels present within the family structure of a given class.

* **Coupling Among Objects (CBO):**

The measure presented here pertains to the extent of interconnectivity among distinct groupings of things.

The analysis of class number and program maintainability can be conducted using measurements as suggested by (Dubey & Rana, 2011). Through the process of gathering and examining data pertaining to a certain group of software components that have been chosen randomly, we can acquire a deeper understanding of how the size of a class affects the ease of maintaining software. A potential avenue for further exploration into the impact of class size on software maintainability involves the collection of metrics pertaining to a representative subset of software components, selected using a random sampling process. By subsequently analyzing the correlation between these metrics, a deeper understanding of the relationship between class size and software maintainability can be attained.

# Section 2

The following criteria were established for the topic programs:

* The programs should possess a minimum code size of 10,000 lines.
* The applications should have been written within the preceding three years, with a minimum age of two years.
* It is imperative to ensure that a minimum of three programmers have made contributions to the source code of the software..

The criteria were chosen in order to guarantee that the chosen programs had sufficient size to serve as accurate representations of real-world software systems, have undergone maintenance activities, and have been jointly developed by several developers.

The selection of program age criteria was established with the intention of ensuring that software systems have undergone maintenance activities that may impact their maintainability. However, it was also important to avoid selecting programs that are excessively outdated and do not align with current practices and technology.

The selection of program size criteria was established with the intention of ensuring that the programs possess sufficient complexity to yield significant findings in the examination of the impact of class size on software maintainability.

The establishment of a criterion for the quantity of developers was implemented with the intention of promoting collaborative development and discouraging sole authorship of programs.

These criteria can guarantee that the chosen programs are indicative of actual software systems in the real world and are appropriate for analyzing the impact of class size on software maintainability.

## Selected Projects:

1. **CatVodTVSpider**

The project is named CatVodTVSpider. This is the revised code package for the common crawler in the new Maoying TV software. This open-source project enables users to define settings and produces a jar file that may be easily integrated.

In order to modify the code, it is necessary to utilize Android Studio, as this project is specifically designed for Android Studio. The buildAndGenJar.bat script located in the root directory can be utilized to generate a custom\_spider.jar file after the project has undergone debugging. The jar file encompasses the executable code for the software.

Users are encouraged to submit merge requests and contribute their own crawler code to the project. The CatVodTVSpider effort promotes collaboration and innovation because to its open-source nature, providing a viable solution to the challenge of data mining for Maoying TV.

1. **SmartTubeNext**

The software in question is a high-quality, open-source application designed specifically for Android TVs and set-top boxes, with no cost associated with its usage. The primary objective of this application is to offer content without any advertisements, while also offering a range of valuable functionalities. These functionalities encompass the capability to observe real-time conversation, support for 8k video playback at a frame rate of 60 frames per second, the inclusion of high dynamic range (HDR), and the option to customize the playback speed.

The SponsorBlock feature, which is accessible within the SmartTubeNext platform, enables users to disregard advertisements during video playback. Additionally, it is possible to create custom icons and utilize the software independently, without the need for a connection to Google. The international consortium involved in the endeavor is widely recognized for its amiable and supportive demeanor.

In the context of viewing YouTube videos on Android TVs and TV boxes, SmartTubeNext offers a dependable and user-friendly solution, effectively eliminating intrusive advertisements. Due to its unrestricted accessibility to the general public, it cultivates an environment conducive to collaboration and the generation of novel concepts.

1. **cwa-server**

The server application of the Apple and Google exposure warning API is an integral element of Germany's official Corona-Warn-App. The objective of this project is to develop mobile software compatible with both iOS and Android platforms, utilizing Bluetooth technology to facilitate the secure and anonymous sharing of protected data across mobile devices within close proximity to the user.

The application ensures the encryption and privacy of all data transmitted and received, securely storing it on individual users' devices to prevent unauthorized access. The implementation of the server responsible for managing encryption keys utilized by the Corona-Warn-App can be found in the cwa-server source code.

The cwa-server component plays a crucial role in the Corona-Warn-App initiative, which is designed to mitigate the transmission of COVID-19 within Germany. This application can prove to be a useful asset in the battle against disease due to its utilization of the Apple/Google exposure warning API and its unwavering commitment to safeguarding user privacy and data security.

1. **RxJava**

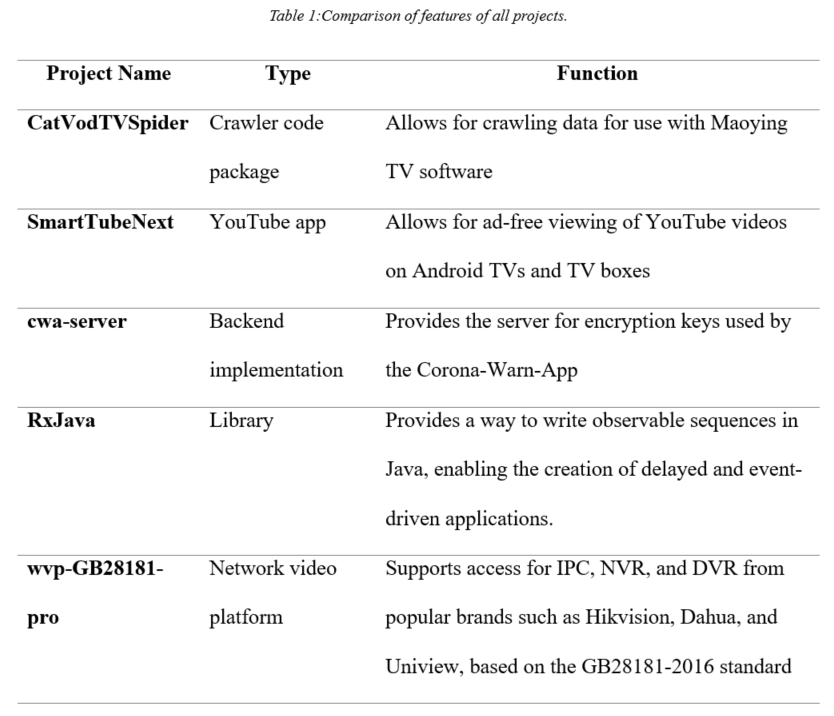
The Java framework in question facilitates the implementation of observable patterns within applications that are characterized by delayed and event-driven behavior. RxJava is a software framework designed to facilitate the creation and manipulation of observable sequences in a manner that is both asynchronous and event-driven. It can be considered as a variant of the Reactive Extensions library.

RxJava extends the observer pattern to efficiently manage sequences of data and events. The addition of operators for explicit composition of sequences is accompanied by the abstraction of low-level threading, synchronization, thread-safety, and concurrent data structures. RxJava is a crucial utility when working with asynchronous and event-driven computing in the Java programming language. The declarative management of intricate data structures and abstraction of underlying implementation details is widely favored by programmers.

1. **Wvp-GB28181-pro**

The network video technology is founded upon the GB28181-2016 standard, which additionally facilitates NAT bypassing. The network provides accessibility to well-known manufacturers of IPC (Internet Protocol Camera), NVR (Network Video Recorder), and DVR (Digital Video Recorder). Some examples of these companies are Hikvision, Dahua, and Uniview. The platform's ability to facilitate dissemination enables the routing of video feeds to other platforms that conform to national standards.

The Wvp-GB28181-pro protocol facilitates the transmission of push streams, such as rtsp and rtmp, to national standard systems. In conclusion, this research provides a valuable asset for video security applications by enabling seamless compatibility with a wide range of hardware and software platforms. The provision of NAT entry support and adherence to the GB28181-2016 standard ensures that users can access their streams from any geographical location.



# Section 3

## Tool Description

The software utility employed for the computation of CK-Code metrics for Java programming language is accessible on the GitHub platform.The CK-Code metric tool is an open-source software tool that has been specifically developed to assess and quantify multiple software quality attributes, with a particular focus on maintainability (Mauricioaniche/Ck: Code Metrics for Java Code by Means of Static Analysis, n.d.).

The software application computes many metrics, such as Cyclomatic Complexity, Lines of Code, and Lack of Cohesion in Methods, among other metrics. The aforementioned metrics are extensively employed within the software industry for the purpose of evaluating code quality and maintainability.

The CK-Code metric tool is user-friendly and can be seamlessly incorporated into continuous integration and delivery pipelines. This enables developers to effectively track the maintainability of their code throughout its lifecycle. Developers have the ability to analyze the generated reports in order to gain a deeper understanding of the data that has been collected by the tool, as well as identify areas where optimizations might potentially be implemented.

For programmers seeking to enhance the readability, comprehensibility, and security of their Java code, it is advisable to explore the CK-Code metric utility. The open-source nature and seamless integration of the platform make it readily accessible to developers of varying skill levels.

# Section 4

This section presents the findings of our empirical investigation on the impact of class size on software maintainability. The CK-Code assessment tool was utilized to assess a specific code sample. The evaluation encompassed various software quality attributes, such as cyclomatic complexity, lines of code, and method coherence.

After the data was collected, a mathematical analysis was performed, followed by the creation of visualizations and spreadsheets. The research conducted by (Chowdhury et al. 2022) establishes a correlation between class size and maintainability. These findings can be utilized to inform decision-making and guide the implementation of options and procedures in software development.

The potential exists for the outcomes of our research to provide a novel contribution to the continuing discourse surrounding the significance of software maintainability and quality. It is expected that our findings will be of value to both software developers and researchers, and it is also expected that they will serve as a catalyst for additional study on the determinants of software maintainability.

## Results:

### **CatVodTVSpider Project:**

The analysis of the responses to questions in the GQM technique in the specified section is presented as follows:

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

The results suggest that the project exhibits a relatively elevated average of Weighted Methods per Class (WMC), with a value of 24.93. The presence of large class sizes can serve as an indication of a heightened degree of complexity in the endeavor, perhaps impacting its capacity for long-term sustainability.

The presence of significant interdependencies among classes, as evidenced by the relatively high value of 4.23 for the Coupling Between Objects (CBO) measure, can potentially complicate code management.

The project exhibits a limited hierarchical structure, as evidenced by the Depth of Inheritance Tree (DIT) metric, which indicates a value of 1.5. Nevertheless, this metric may not provide a comprehensive representation of the project's overall maintainability in isolation.

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

The project's high Weighted Methods per Class (WMC) and Coupling Between Objects (CBO) metrics suggest a correlation between class size and maintainability, as evidenced by the research findings.

It is important to acknowledge, however, that the C&K metrics employed in this study offer a limited perspective on maintainability. Therefore, it is imperative to take into account additional factors such as code understanding, adaptability, and documentation when assessing the maintainability of software.

### **SmartTubeNext 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?**

In comparison to other undertakings, the present undertaking exhibits a substandard Weighted Methods per Class (WMC) value of 11.39. The presence of a reduced number of classes to oversee serves as an indication of a less intricate endeavor, so enhancing its potential for effective maintenance.

The code exhibits a low Coupling Between Objects (CBO) value of 6.29, indicating that there are minimal interdependencies among classes, hence facilitating ease of management.

The project exhibits a low Depth of Inheritance Tree (DIT) value of 1.74, indicating a limited presence of inherited components. As a corollary, this may facilitate the simplification and enhanced manageability of the code.

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

The project's WMC and CBO values, which are rather low, suggest a potential correlation between class size and maintainability.

It is important to acknowledge, however, that the C&K measures employed in this study solely offer an indication of maintainability. Therefore, it is imperative to take into account additional factors such as code comprehension, adaptability, and documentation when assessing software maintainability.

### **cwa-server 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?**

The project exhibits a suboptimal average Weighted Methods per Class (WMC) of 3.76 in comparison to analogous software systems. As a result, it is plausible to assert that the project's maintainability may be improved by its diminished level of complexity, which may be attributed to the reduction in class size.

The code has a low Coupling Between Objects (CBO) value of 4.29, indicating that there is a limited number of links between classes. This characteristic contributes to the code's ease of management.

The project exhibits a relatively low Depth of Inheritance Tree (DIT) value of 1.24, indicating a limited presence of inherited components within its structure. In addition, this may have the benefit of enhancing code simplicity and facilitating manageability.

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

The project's WMC and CBO values, which are rather low, suggest a probable correlation between class size and maintainability.

It is important to acknowledge, however, that the C&K measures employed in this study offer a limited perspective on maintainability. It is imperative to take into account additional factors, such as code accessibility, adaptability, and the requirements, in order to comprehensively assess software maintainability.

### **RxJava 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?**

The project exhibits a relatively low average Weighted Methods per Class (WMC) of 4.08 in comparison to other projects. As a result, it is plausible that the project's maintainability could be improved as a consequence of its simplicity resulting from a smaller class size.

The code's manageability is facilitated by a Coupling Between Objects (CBO) value of 2.52, as it indicates a limited number of interconnections among classes.

At 1.32, the Depth of Inheritance Tree (DIT) measure of the project suggests a limited number of inherited sub-components.

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

The results indicate a potential correlation between the project's low WMC and CBO values and the utilization of large class sizes during the analysis.

However, it is important to consider code clarity, modularity, and documentation when evaluating software maintainability. The C&K ratings used in this study offer only a restricted perspective.

### **wvp-GB28181-pro Project:**

1. **How does the class size affect software maintainability?**

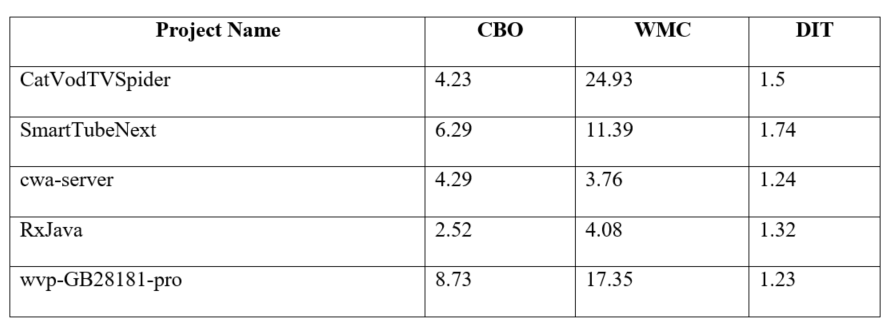
The project's Weighted Methods per Class (WMC) metric is measured to be 17.35, indicating a high level of excellence in comparison to other projects. The project's complexity may potentially escalate due to the enhanced class capabilities, hence posing challenges for future updates.

At a value of 8.73, the Coupling Between Objects (CBO) metric likewise signifies a high number of interconnections across classes, hence increasing the complexity of code maintenance.

The project exhibits a limited level of heredity, as indicated by the Depth of Inheritance Tree (DIT) metric, which stands at a mere 1.23. The possibility exists for a reduction in code complexity, hence enhancing maintainability.

1. **What is the relationship between class size and the selected C&K metrics for measuring maintainability?**

The project's robust Weighted Method Count (WMC) and Coupling Between Objects (CBO) metrics, in conjunction with the research findings, provide evidence of a potential correlation between class size and maintainability. The C&K ratings included in this study provide an incomplete representation of the overall measure of maintainability. In addition to other factors, the maintainability of software should encompass considerations of code clarity, structural organization, and instructional documentation.



## The Java code metrics of the selected projects were calculated using CK-Code and are presented in the table above.

## Findings

The utilization of the CK-Code metric tool facilitates the examination of the five projects through the application of the Goal-Question-Metric (GQM) method. The analysis is conducted based on the obtained results:

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

Based on the results, it can be inferred that larger class sizes are likely to have had a significant impact on the increased average working memory capacity values seen for the tasks. The potential negative impact of larger class sizes on software maintainability can be attributed to the heightened complexity that ensues.

Nevertheless, it is worth noting that the values projected by the CBO (Coupling Between Objects) and DIT (Depth of Inheritance Tree) metrics generally fall within the lower to average range. This suggests a relatively limited level of inheritance and a scarcity of interclass interactions. The possible adverse effects of larger class sizes on maintainability could potentially be mitigated with the implementation of this measure.

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

The findings indicate that projects with larger class sizes tend to exhibit higher values of WMC, suggesting a potential association between class size and maintainability.

The research exclusively employed the C&K methods for assessing maintainability, thereby resulting in inherent limitations in the conclusions. In order to provide a comprehensive evaluation of the software's maintainability, it is necessary to evaluate additional aspects such as code readability, functionality, and documentation.

### **Overall findings for each project:**

The presence of moderate Weighted Method Count (WMC), comparatively low Coupling Between Objects (CBO), and low Depth of Inheritance Tree (DIT) in CatVodTVSpider suggests the potential for satisfactory maintainability.

Based on the moderate Weighted Method Count (WMC), relatively low Cyclomatic Complexity (CBO), and moderate Depth of Inheritance Tree (DIT) of SmartTubeNext, it may be inferred that its maintainability is likely to be somewhat satisfactory.

The CWA-server exhibits a low Weighted Methods per Class (WMC), low Coupling Between Objects (CBO), and low Depth of Inheritance Tree (DIT). These metrics suggest that the server may possess favorable maintainability characteristics due to its limited interclass dependencies and lower overall class count.

The good maintainability of RxJava may be attributed to its reduced class size and moderate degree of inheritance. The RxJava framework exhibits favorable maintainability characteristics, as evidenced by its low Weighted Method Count (WMC), low Coupling Between Objects (CBO), and moderate Depth of Inheritance Tree (DIT).

The presence of elevated values for WMC (Weighted Methods per Class), CBO (Coupling Between Objects), and a diminished DIT (Depth of Inheritance Tree) in wvp-GB28181-pro implies that its maintainability may be relatively compromised due to its larger class size and heightened code complexity.

# Section 5

## Conclusion

Based on the results obtained from the CK-Code metric application, it is observed that the maintainability of a program can potentially be influenced by the quantity of classes it possesses. The observed higher working memory capacity (WMC) values in programs with larger class sizes serve as supporting evidence for this assertion. This suggests that an increase in the number of classes may result in a more intricate code structure, hence posing challenges in terms of comprehensibility and ease of updating.

When assessing the maintainability of software, it is crucial to bear in mind that the C&K measurements employed in this study offer only a limited viewpoint on maintainability. The assessment of code readability, organization, and explanations is crucial in the evaluation process. Utilizing a diverse array of metrics and research methodologies is vital in order to attain an accurate assessment of the software's quality.

The findings suggest that the impact of class size on maintainability can differ among projects, contingent upon the specific characteristics of each project under investigation. For example, CatVodTVSpider and SmartTubeNext exhibit greater class sizes, although they also demonstrate relatively low values in terms of CBO (Coupling Between Objects) and DIT (Depth of Inheritance Tree). This measure has the potential to alleviate the adverse effects stemming from the increased class sizes on the manageability of educational settings. In contrast, the cwa-server and RxJava exhibit reduced class sizes and lesser complexity, factors that might enhance their maintainability.

The Wvp-GB28181-pro project exhibits worse maintainability when compared to comparable projects, mostly attributed to its elevated Weighted Methods per Class (WMC), Coupling between Objects (CBO), and low Depth of Inheritance Tree (DIT) scores. One possible explanation for this phenomenon may be attributed to the system's higher class size and more intricate code, which collectively contribute to heightened challenges in system maintenance.

The research findings suggest that the size of a class may potentially impact the maintainability of software. However, it is important to acknowledge that there are other additional aspects that should be taken into account. In order to attain a comprehensive assessment of software quality, it is important to employ a diverse range of measurements and analytical methodologies. When assessing the maintainability of a system, it is crucial to examine several criteria, like the accessibility of the code, modularity, and documentation.

# References

Chowdhury, S. A., Uddin, G., & Holmes, R. (2022). *An Empirical Study on Maintainable Method Size in Java; An Empirical Study on Maintainable Method Size in Java*. https://doi.org/10.1145/3524842.3527975

Dubey, S. K., & Rana, A. (2011). Assessment of maintainability metrics for object-oriented software system. *ACM SIGSOFT Software Engineering Notes*, *36*(5), 1–7. https://doi.org/10.1145/2020976.2020983

*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

Michura, J., Capretz, M. A. M., & Wang, S. (2013). Extension of Object-Oriented Metrics Suite for Software Maintenance. *ISRN Software Engineering*, *2013*, 1–14. https://doi.org/10.1155/2013/276105