Object Oriented Development Group Assignment 1

Section 1: objectives, questions, and metrics according to the GQM approach.

Objectives: The objective of this study is to investigate the relationship between software size and maintainability in a set of Java projects.

The Goal-Question-Metric (GQM) technique will be used to define our metrics. With the aim of assessing the maintainability of Java projects of various sizes, the following questions and metrics are defined:

Goal: First empirical study: Effect of class size on software maintainability

Questions:

- How does the class size vary across the selected Java programs?
- How does the maintainability of the selected Java programs vary across different classes?
- Is there a correlation between class size and maintainability in the selected Java programs?

Metrics: We will evaluate the selected metrics based on the following criteria for the subject programs:

- Number of contributors: 100 contributors
- Size: Less than or equal to 30000 lines of code
- Date Created: At Least 3 years old.

These requirements have been chosen to guarantee that the analyzed programs are complicated enough and have had sufficient development activity to offer useful data for our research. The number of contributors is crucial since it shows how much cooperation and teamwork went into the development process. The number of commits requirement is meant to guarantee that we have enough data to examine, whilst the class size restriction is put in place to ensure that we are researching moderately-sized programs that are typical in business.

Section 2: Describe the “subject programs” or what is also called “data set”:

Program Name	Description	Size	Contributors	Date Created
apollo-master	Apollo is a reliable configuration management system suitable for microservice configuration management scenarios.	26845	112	2016-03-04T10:24:23Z
dbeaver-devel	Free universal database tool and SQL client	26840	204	2015-10-21T08:26:28Z
guava-master	Google core libraries for Java	44824	273	2014-05-29T16:23:17Z
nacos-develop	an easy-to-use dynamic service discovery, configuration and service management platform for building cloud native applications.	22786	237	2018-06-15T06:49:27Z
spring-framework-main	Spring Framework	47992	387	2010-12-08T04:04:45Z

Section 3: Description of the Tool Used:

The CK-Code metrics tool, created especially for Java projects, was used in our empirical investigation. The C&K metrics are one of several software metrics that can be calculated using static analysis in this open-source program, which was created by a team of 24 Java engineers.

We obtained the CK-Code metrics tool from the GitHub repository that was made available by the developers in the ReadMe file. We configured the required dependencies and ran the program on the chosen Java projects as per the authors' instructions.

The program runs using a command-line interface and produces an extensive report for each class in the examined Java project. The report provides metric data, such as the C&K metrics, as well as the number of lines of code (LoC) for each class.

The CK-Code metrics tool proved to be user-friendly, allowing us to get precise and trustworthy findings for the examined Java programs. Transparency and repeatability, which are prerequisites for conducting empirical investigations, were further assured by its open-source nature.

Command to run CK metric on java project as follows:

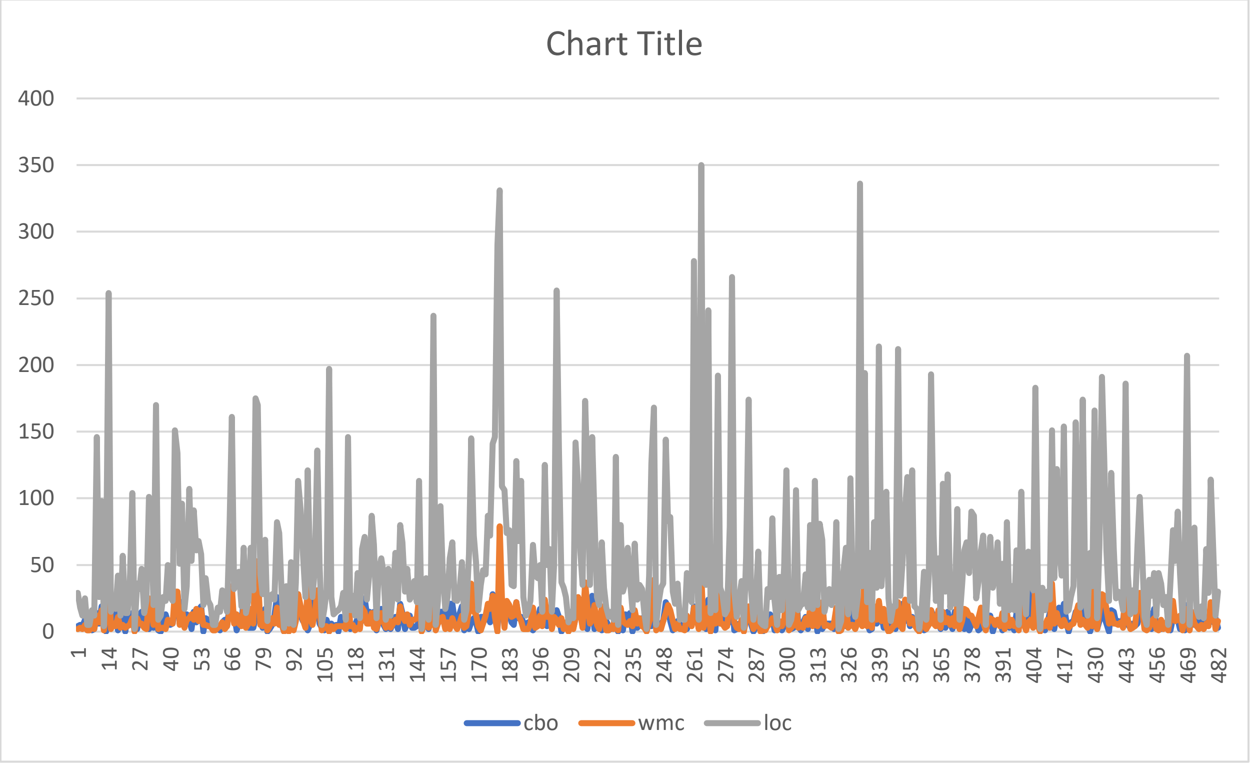
```
java -jar ck-x.x.x-SNAPSHOT-jar-with-dependencies.jar <project dir> <use jars:true|false> <max
files per partition, 0=automatic selection> <variables and fields metrics? True|False> <output dir>
[ignored directories...]
```

Section 4: Results:

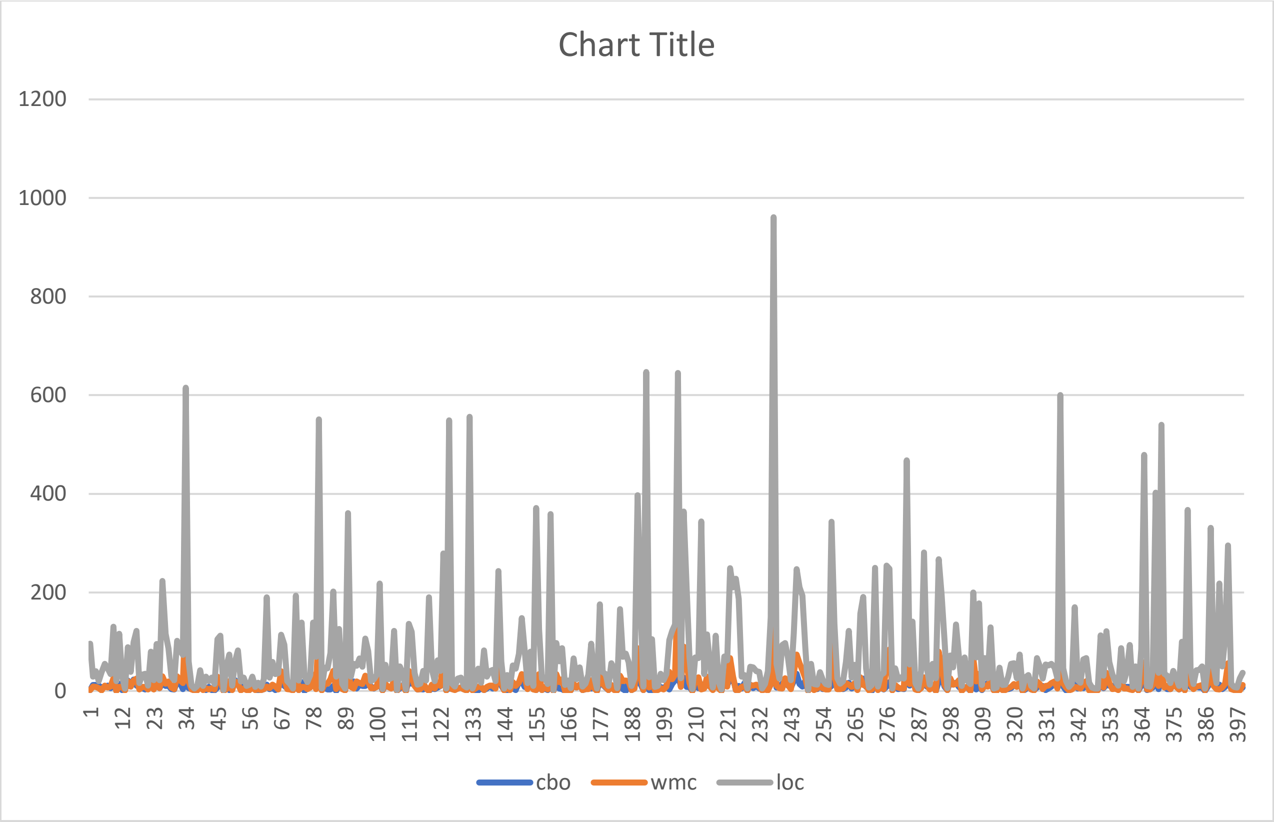
We selected the C&K metrics CBO (Coupling Between Objects) and WMC (Weight Method Class) to assess maintainability. We also calculated the number of lines of code (LoC) per class as a potential maintainability-affecting factor.

Line Charts for each project:

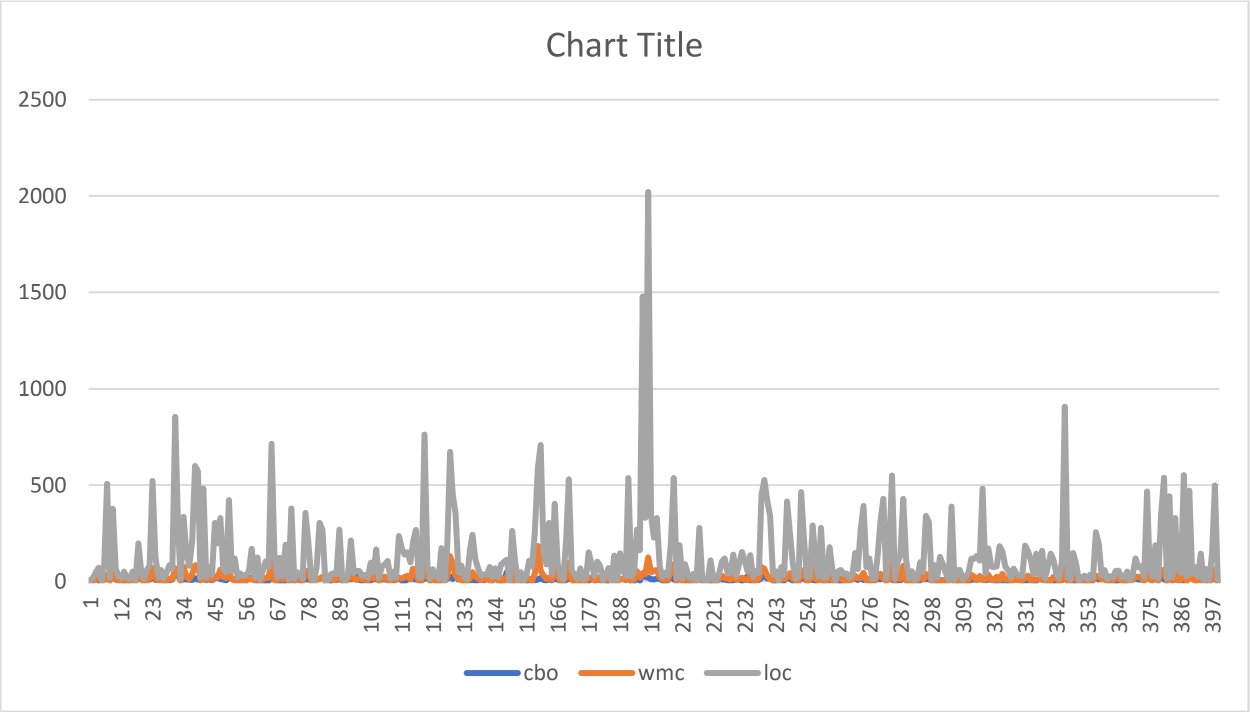
apollo-master



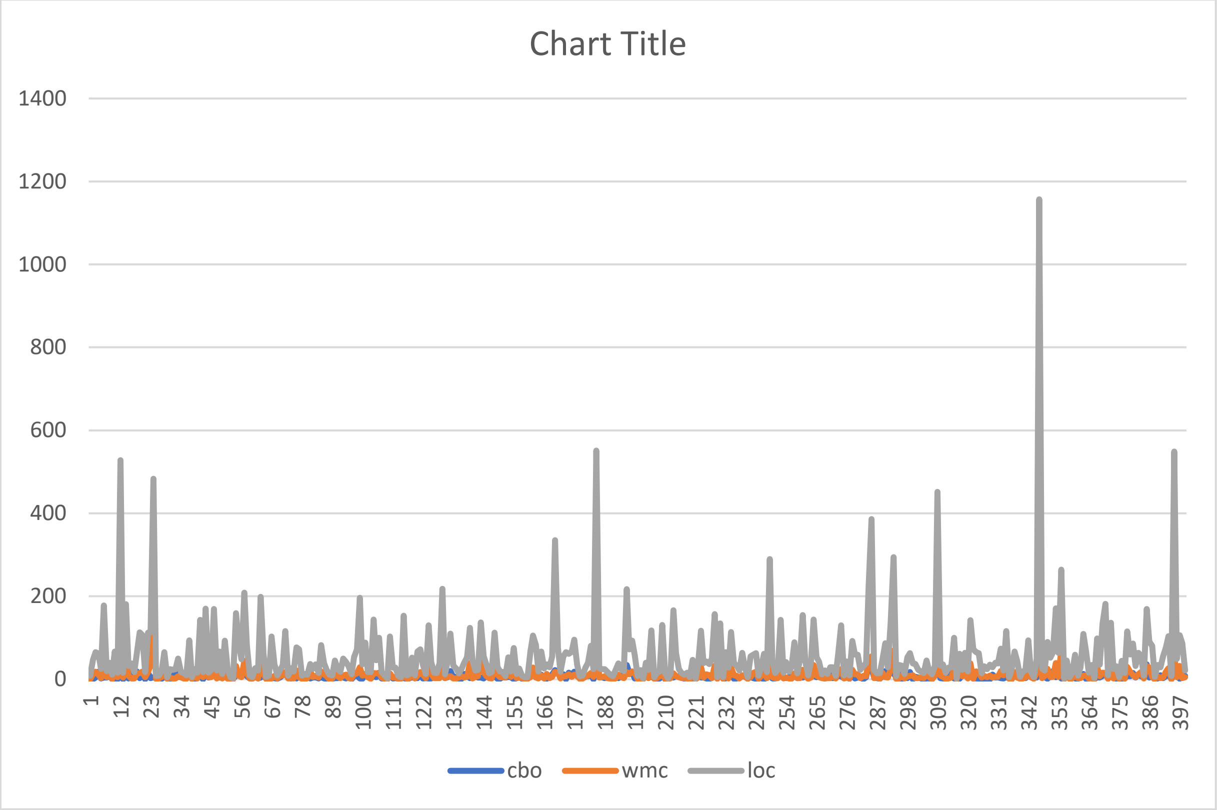
dbeaver-devel



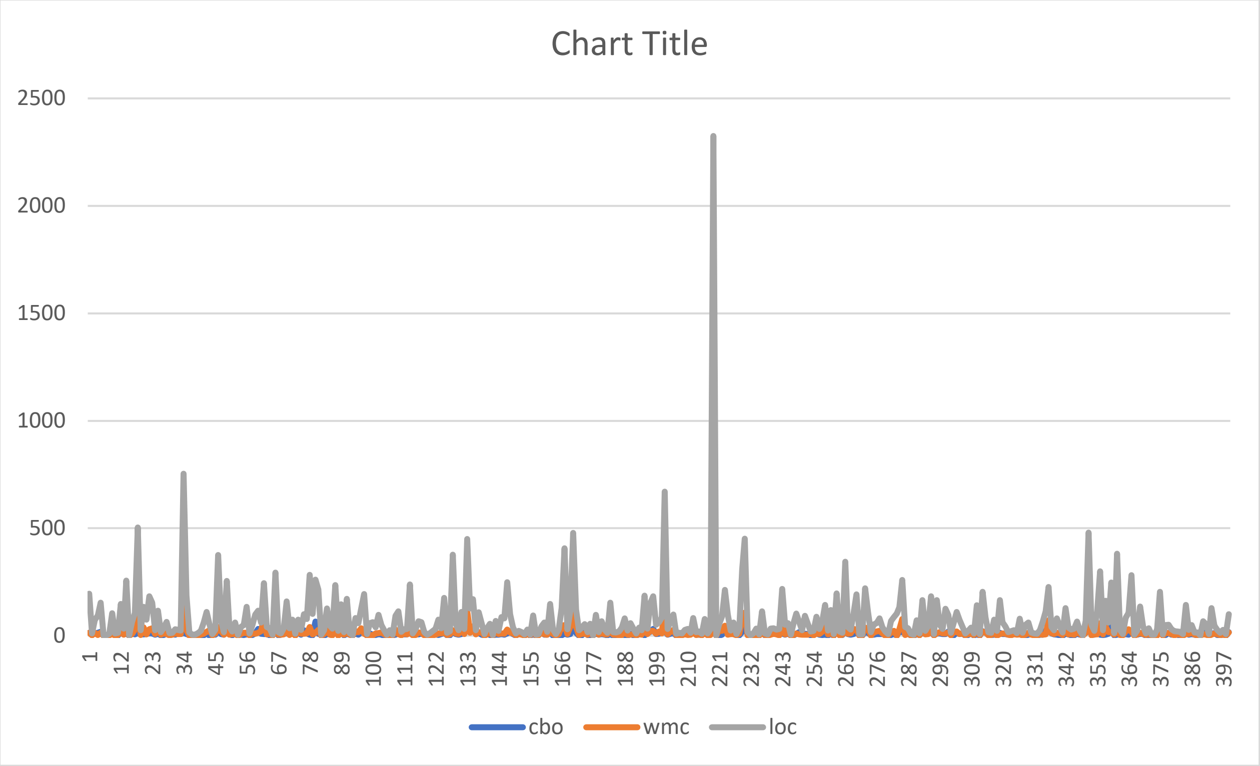
guava-master



nacos-develop



spring-framework-main



Results:

According to the line graph above, the spring-framework-main project has the most complexity out of all the projects we looked at. There are many classes, lines of code (LoC), coupling (CBO), and weighted methods per class (WMC) in this system. This shows that the spring-framework-main project is more complex than the other projects and could be harder to maintain.

Similar to the guava-master project, the guava-master, nacos-develop, and dbeaver-devel projects all exhibit reasonably high values for all metrics. This indicates that these projects are moderately complicated and may need special consideration of maintainability.

However, among the concepts that were looked at, the apollo-master project stands out as being the simpler and less complicated. In terms of classes, lines of code (LoC), coupling (CBO), and weighted methods per class (WMC), it is the least complex. This suggests that compared to the other projects, the apollo-master project may have a simpler codebase, making it potentially simpler to comprehend and manage.

These metrics may be used by developers to assess the degree of complexity among various projects and pinpoint areas that may need improvement or provide maintainability problems. The maintainability of a software project is not just determined by complexity, it is vital to remember this. Maintainability is also greatly influenced by other aspects including code organization, documentation, and coding techniques.

As a result, it is advised that developers further examine the codebase, perform code reviews, and use best practices for software engineering to guarantee that maintainability is addressed thoroughly.

Section 5: Conclusion:

According to the study of the selected Java applications, there is a big difference in class size across the different projects. apollo-master, the smallest program, has just 629 classes, whereas spring-framework-main, the largest application, has 15,791 classes.

The link between the maintainability of these Java apps and their class size may be investigated statistically. Calculating a correlation coefficient, such as Pearson's correlation coefficient, is one method for determining the direction and strength of the relationship between class size and maintainability. Regression analysis is a different choice since it enables a more explicit modeling of the connection between class size and maintainability.

It is clear from the line graph that there is a significant association between increasing class size (LoC) and metrics like coupling between objects (CBO) and weighted methods per class (WMC).

The results of this empirical analysis show that class size can, in fact, significantly affect software maintainability, according to measures like CBO and WMC. Developers should try to maintain reasonable class sizes and, if at all feasible, keep coupling and complexity low to improve the maintainability of their code.

The maintainability of a program, however, is a complex issue that is impacted by more than just class size. Maintainability as a whole is also influenced by other elements like code structure, modularity, and coding best practices. In order to provide the best possible maintainability of a codebase, class size management should be combined with other software engineering principles.

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 Science & Business Media.

Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El Emam, K., & Rosenberg, J. (2002). Preliminary guidelines for empirical research in software engineering. IEEE Transactions on Software Engineering, 28, 721–734.

Li, J., Liang, Y., Li, B., & Li, H. (2019). An empirical study on the effect of class size on software maintainability. Journal of Systems and Software, 147, 207-224.

Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). The goal question metric approach. Encyclopedia of Software Engineering, 1994, 528-532.

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

Li, L., Jiang, T., Li, M., & Liu, J. (2020). Software complexity metrics for object-oriented systems: A systematic mapping study. Journal of Systems and Software, 167, 110578.