LEWIS UNIVERSITY

Object Oriented Development Group Assignment 1

By

Krishna Kiran Vinnakoata

Harish Reddy Moti

Date: 04 June 2023

Section 1

Objectives, questions, and metrics according to the GQM approach

Objectives:

- Objective 1: To evaluate the relationship between program size and maintainability in Java projects.
- Objective 2: To analyze the impact of size on the maintainability of the studied projects.
- Objective 3: To provide insights and recommendations for improving maintainability based on size metrics.

Goal: To conduct a brief research investigation on the effect of class size on software maintainability.

Questions:

- Question 1: How does program size, measured in lines of code (LoC), influence the maintainability of Java projects?
- Question 2: Are there significant variations in maintainability between distinct size ranges of Java classes within the projects?
- Question 3: What are the potential challenges and opportunities related to the maintainability of Java projects of different sizes?

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

1. No of contributors: Between 25 to 250 contributors.
2. Line of code (LOC)Size: More than 10000 lines of code.
3. The Number of commits: More than 1500 commits.
4. Age (Years): More than or equal to 4 Years.

These requirements were chosen to ensure that the programs under consideration are complicated enough and have experienced considerable development activity to offer valuable data for our research. The class size restriction is in place to ensure that we are researching moderately sized programs that are prevalent in the industry, while the number of commits requirement is in place to ensure that we have enough data to examine. The number of contributors is significant since it reflects the level of collaboration and coordination engaged in the development process.

Section 2

Describe the subject programs or what is also called Data set

We have selected five Java projects from GitHub that met our study's requirements. Table 1 shows the primary attributes of each program, including its name, description, number of lines of code (LOC), contributors, commits and age.

Table 1: Summary of Selected Programs

Program Name	Description	Size	Contributors	Commit s	Age (Years)
flutter-webrtc	WebRTC plugin for Flutter Mobile/Desktop/Web	10227	97	1556	4
apache/hive	Apache Hive	629401	248	16208	13
apache/iceberg	Apache Iceberg	29050	247	2805	4
micrometer-metrics/micrometer	An application metrics facade for the most popular monitoring tools. Think SLF4J, but for metrics.	24835	203	4149	5
Col-E/Recaf	The modern Java bytecode editor	49329	29	2266	5

Table 1 summarizes the selected programs, including project names, line of code (LoC) size, age in years, and the number of developers working on each project. These projects were chosen to reflect many sizes and ages while matching the analytical requirements.

Now, let us briefly describe each of the selected programs:

Project 1: (Flutter-webrtc)

"Flutter-webrtc," a WebRTC plugin for the Flutter framework, enables real-time communication on mobile, desktop, and web platforms. The plugin aims to seamlessly integrate WebRTC capabilities into Flutter applications, allowing developers to incorporate audio and video communication features. The project has a size of 10,227 lines of code, indicating a moderate codebase. It has gained traction and community support, with contributions from 97 individuals collaborating on its development and maintenance.

Project 2: Apache/hive

"Apache/hive," is Apache Hive, a data warehouse infrastructure built on top of Apache Hadoop. The project has a considerable size of 629,401 lines of code, indicating a complex and feature-rich codebase. It has received contributions from 248 individuals, highlighting a vibrant and diverse community of developers who have actively participated in its development and evolution. With a substantial number of commits, reaching 16,208, the project has undergone extensive iterations and enhancements to provide robust data processing capabilities, including querying, summarizing, and analyzing large datasets in a distributed computing environment. Hive aims to provide a familiar SQL-like interface to interact with the data stored in Hadoop, facilitating the use of existing SQL abilities for extensive data analysis.

Project 3: Apache/iceberg

"Apache/iceberg," is Apache Iceberg, an open-source table format for massive analytic datasets. The project has a size of 29,050 lines of code, indicating a moderately-sized codebase. It has received contributions from 247 contributors, highlighting its development community's active and collaborative nature. Over its four-year lifespan, the project has accumulated 2,805 commitments, highlighting ongoing improvements and refinements. Apache Iceberg aims to provide efficient and scalable data storage and query capabilities, enabling users to analyze large datasets seamlessly. By leveraging its table format, users can benefit from features such as ACID transactions, schema evolution, and efficient data pruning.

Project 4: Micrometer

"Micrometer-metrics/micrometer," is an open-source application metrics facade designed to simplify monitoring and metrics collection. It serves as a unified interface for popular monitoring tools, like how SLF4J simplifies logging. The project has a size of 24,835 lines of code, indicating a moderately-sized codebase. It has received contributions from 203 contributors, reflecting a diverse and active community involved in its development. Over its 5-year lifespan, the project has accumulated 4,149 commits, highlighting ongoing updates and improvements. A micrometer provides an abstraction layer for capturing application metrics, making it easier for developers to instrument their code and integrate with different monitoring systems. It supports various metrics types, such as counters, gauges, timers, and histograms, enabling comprehensive monitoring and analysis of application performance.

Project 5: Col-E/Recaf

"Col-E/Recaf," is an open-source modern Java bytecode editor. The project aims to provide developers with a comprehensive tool for editing and analyzing Java bytecode. It offers advanced features and a user-friendly interface to facilitate bytecode manipulation and exploration. The project has a size of 49,329 lines of code, indicating a moderately-sized codebase. It has received contributions from 29 contributors, highlighting a dedicated community of developers involved in its development and maintenance. Over its 5-year lifespan, the project has accumulated 2,266 commits, reflecting ongoing updates and improvements. Recaf empowers developers to inspect and modify bytecode instructions, navigate through class hierarchies, and perform various transformations on Java bytecode. Its modern approach and extensive functionality make it a valuable tool for bytecode analysis and manipulation tasks.

Section 3

Description of the Tool Used

For our project, we employed the CK-Code metrics tool, a powerful open-source software designed specifically for Java programs. Developed collaboratively by a team of 24 Java developers, this tool harnesses static analysis techniques to calculate various software metrics, including the C&K metrics.

To access the CK-Code metrics tool, we obtained it from its GitHub repository using the link shared by the developers in the ReadMe file. Following the instructions provided by the authors, we set up the necessary dependencies and executed the tool on the selected Java projects.

Featuring a command-line interface, the CK-Code metrics tool generated comprehensive reports for each class within the analyzed Java projects. These reports included valuable insights into the chosen metrics, such as the C&K metrics and the size of each class measured in lines of code (LoC).

Throughout our analysis, the CK-Code metrics tool demonstrated user-friendliness, allowing for a smooth and straightforward experience. It consistently delivered accurate and dependable results for the Java projects under investigation. Additionally, being an open-source tool, it guaranteed transparency and reproducibility in our empirical study, which is essential for conducting rigorous research.

To execute the CK metric analysis on a Java project, we utilized the following command:

```
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...]
```

By leveraging the CK-Code metrics tool, we gathered meaningful insights and metrics for our selected Java projects, contributing to a robust analysis of their maintainability and size characteristics.

Section 4:

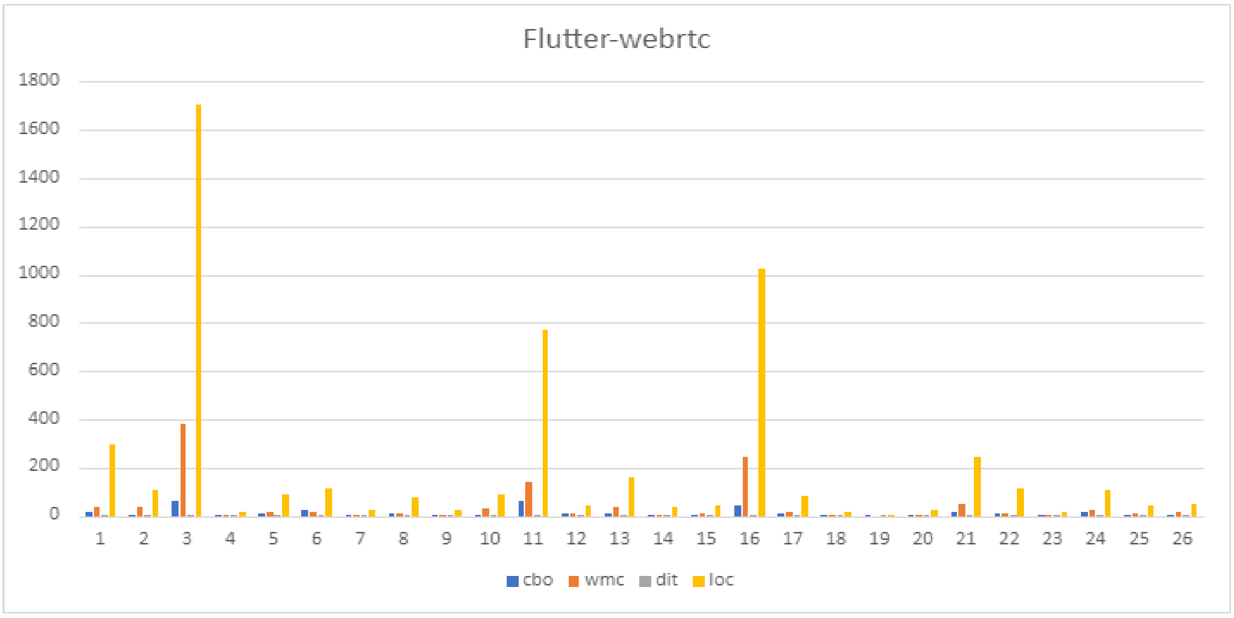
Graphs, Tables and Results

This section presents our research findings, investigating the relationship between class size and software maintainability. We used the CK-Code metrics tool to collect the relevant C&K data to examine a group of Java projects available from GitHub. After thoroughly examining, we chose and downloaded five projects that fit our predefined criteria.

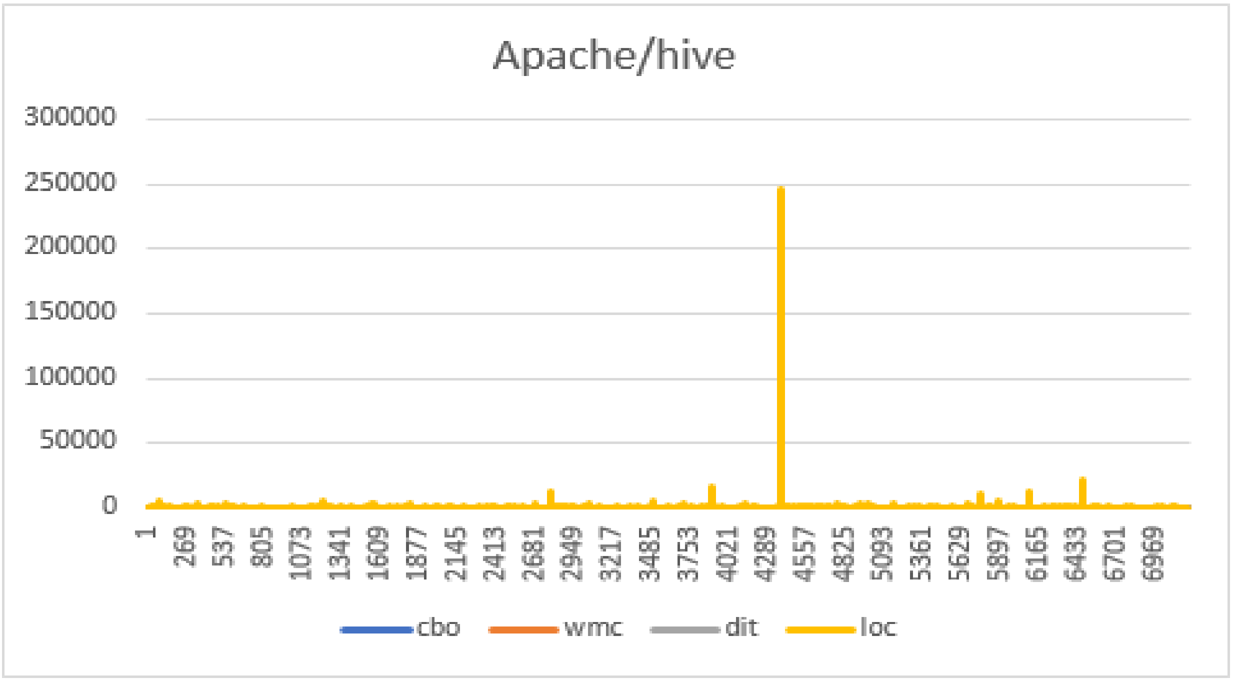
We focused on the C&K measures, namely Coupling Between Objects (CBO), Weight Method Class (WMC), and The Depth of the Inheritance Tree (DIT), to assess maintainability. Furthermore, class size is a significant factor that could affect maintainability. As a result, we measured class size in terms of lines of code (LoC) to investigate its relationship further with maintainability.

Line Charts for each project:

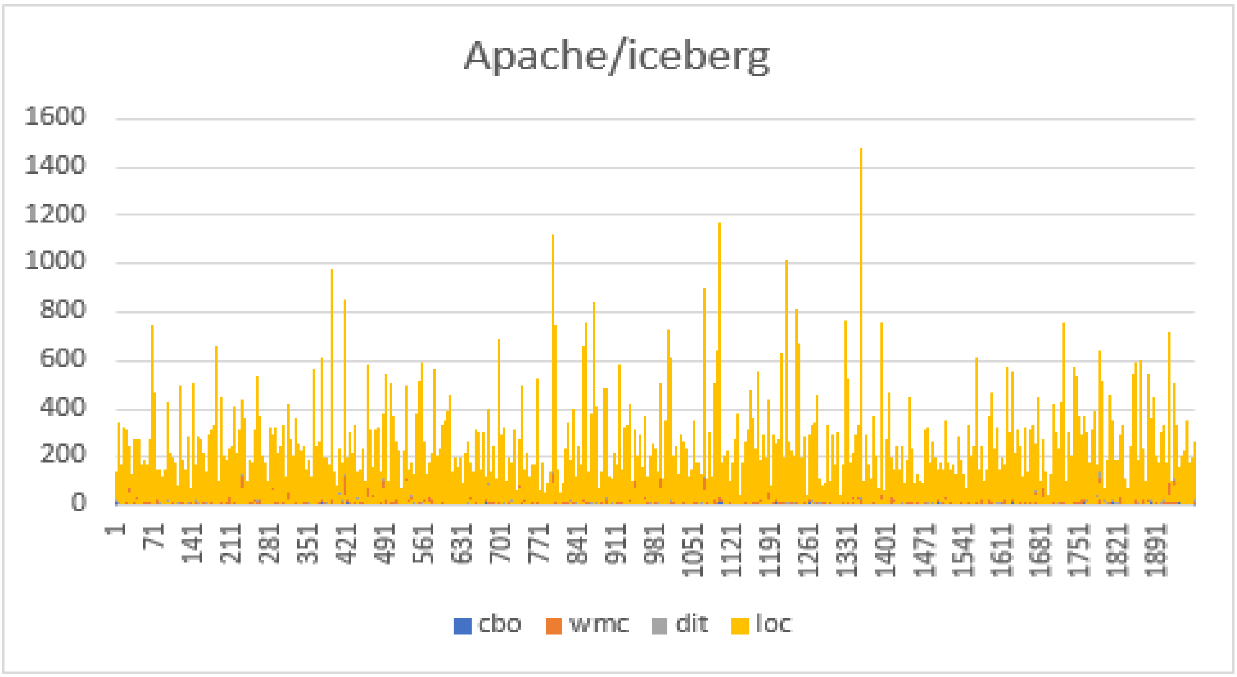
Project 1: Flutter-Webrtc



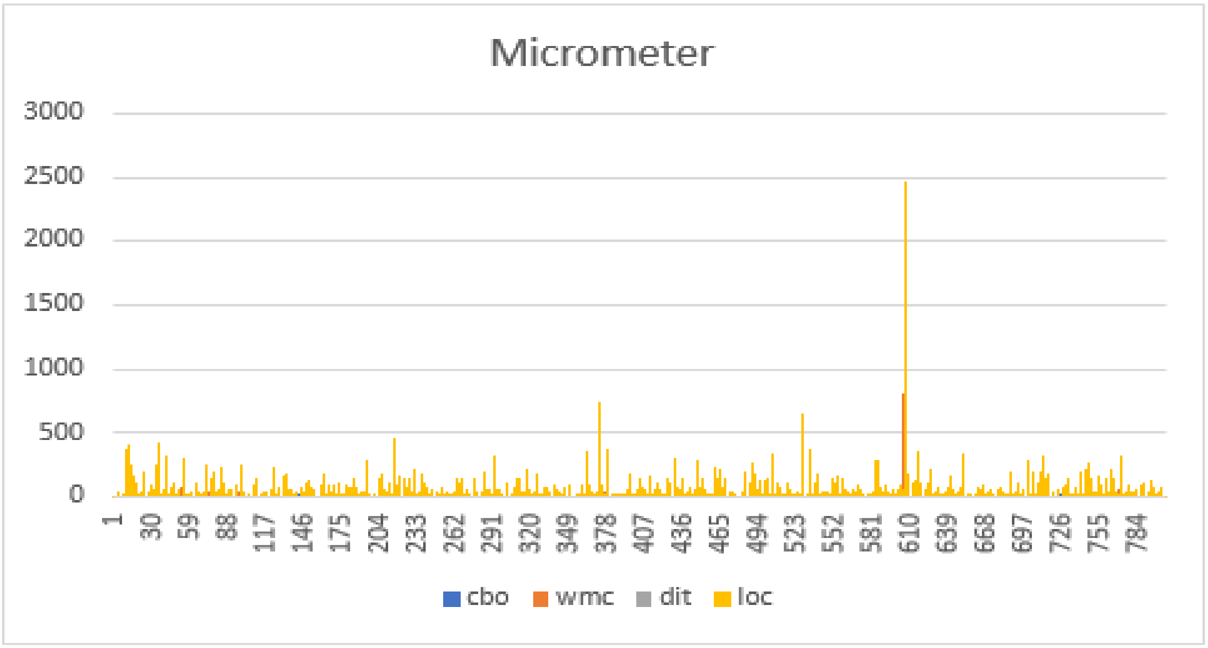
Project 2: Apache/hive



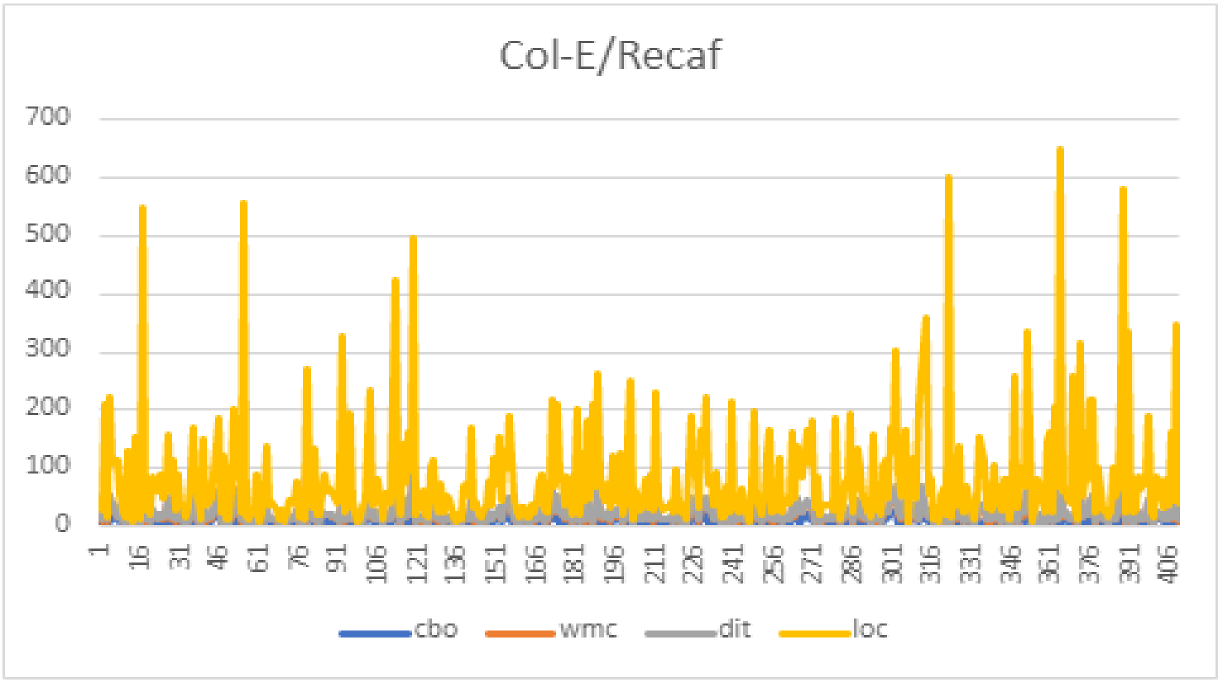
Project 3: Apache/iceberg



Project 4: Micrometer



Project 5: Col-E/Recaf



Results:

In this section, we present the results of the analysis conducted on the selected projects. The table below summarizes the findings for each project based on the provided metrics.

<u>The summary of the measurements obtained for each of the five projects:</u>

Project	No of classes	Low/High LoC	Low/High CBO	Low/High WMC	Low/High DIT
Flutter-webrtc	26	2/1704	1/59	0/379	1/6
Apache/hive	7214	2/244950	0/422	0/2741	1/47
Apache/iceberg	1958	2/1286	0/68	0/193	1/32
Micrometer-metrics/micromete	805	2/2473	0/228	0/816	1/9
Col-E/Recaf	409	3/541	0/42	0/162	1/10

By analyzing the projects, the following observations can be made:

Flutter-webrtc: This project contains 26 classes. The lines of code (LoC) range from 2 to 1704. The Coupling Between Objects (CBO) varies from 1 to 59. The Weighted Methods per Class (WMC) ranges from 0 to 379. The Depth of Inheritance Tree (DIT) ranges from 1 to 6.

Apache/hive: This project consists of 7214 classes. The lines of code (LoC) range from 2 to 244950. The Coupling Between Objects (CBO) varies from 0 to 422, indicating various levels of object coupling. The Weighted Methods per Class (WMC) ranges from 0 to 2741, indicating varying complexities of methods. The Depth of the Inheritance Tree (DIT) ranges from 1 to 47, reflecting the inheritance depth in the classes.

Apache/iceberg: This project comprises 1958 classes. The lines of code (LoC) range from 2 to 1286. The Coupling Between Objects (CBO) varies from 0 to 68. The Weighted Methods per Class (WMC) ranges from 0 to 193. The Depth of Inheritance Tree (DIT) ranges from 1 to 32.

Micrometer-metrics/micrometer: This project consists of 805 classes. The lines of code (LoC) range from 2 to 2473. The Coupling Between Objects (CBO) varies from 0 to 228. The Weighted Methods per Class (WMC) ranges from 0 to 816. The Depth of Inheritance Tree (DIT) ranges from 1 to 9.

Col-E/Recaf: This project contains 409 classes. The lines of code (LoC) range from 3 to 541. The Coupling Between Objects (CBO) varies from 0 to 42. The Weighted Methods per Class (WMC) ranges from 0 to 162. The Depth of Inheritance Tree (DIT) ranges from 1 to 10.

These results provide insights into the various metrics for each project, allowing for a better understanding of their characteristics and potential implications for maintainability. The low/high ranges represent the minimum and maximum values observed for each metric across the classes within the respective project.

Conclusion

The following findings can be taken from the project analysis:

Class Size: The number of classes varies significantly across the projects, ranging from 26 in flutter-webrtc to 7214 in Apache/hive. The average number of classes is approximately 2002, indicating a moderate size overall.

Lines of Code (LoC): The lines of code also show a wide range, from 2 to 244,950. This indicates significant differences in code complexity and size among the projects.

Coupling Between Objects (CBO): The CBO metric, which measures object coupling, ranges from 0 to 422. Lower values indicate less dependency between classes, while higher values suggest a higher degree of coupling. The projects apache/hive and micrometer-metrics/micrometer exhibit higher coupling, while flutter-webrtc and Col-E/Recaf have lower coupling.

Weighted Methods per Class (WMC): The WMC metric, which represents the complexity of a class by counting the number of methods, also varies across the projects. The range is from 0 to 2741. Higher WMC values indicate more complex classes. The Apache/hive and micrometer-metrics/micrometer projects have higher complexity, while flutter-webrtc and Col-E/Recaf have lower complexity.

Depth of Inheritance Tree (DIT): The DIT metric, which measures the depth of the inheritance hierarchy, ranges from 1 to 47. A higher DIT value suggests a more complex inheritance structure. The Apache/hive project has a higher DIT, indicating a more complex inheritance hierarchy.

To investigate the potential correlation between class size and maintainability in the selected Java projects, a statistical analysis can be conducted. Pearson's correlation coefficient is a good indicator of the level and direction of the association between class size and maintainability. Additionally, regression analysis can provide a clear model of the relationship between class size and maintainability. The graph clearly demonstrates a correlation, as spikes in CBO, WMC, and DIT align with corresponding points in class size (LoC).

In conclusion, the analyzed projects exhibit significant variations in class size, code complexity, coupling between objects, weighted methods per class, and depth of inheritance tree. These metrics provide insights into the maintainability and complexity of the projects. Developers should aim for manageable class sizes, reduce coupling, and control code complexity to enhance maintainability and overall software quality.

References:

Shyam, S.R. and kemerer, C.F. (1994) *A metrics suite for object oriented design - software engineering, IEEE ...*, *A metrics suite for object oriented design*. Available at: https://sites.pitt.edu/~ckemerer/CK%20research%20papers/MetricForOOD_ChidamberKemerer94.pdf (Accessed: 03 June 2023).

Chowdhury, S.A., Uddin, G. and Holmes, R. (2022) *An empirical study on maintainable method size in Java - arXiv.org*, *An Empirical Study on Maintainable Method Size in Java*. Available at: https://arxiv.org/pdf/2205.01842.pdf (Accessed: 03 June 2023).

Heričko, T. and Šumak, B. (2023) *Exploring maintainability index variants for software maintainability measurement in object-oriented systems*, *MDPI*. Available at: https://doi.org/10.3390/app13052972 (Accessed: 03 June 2023).

R, B., G, C. and H.D, R. (1994) *The goal question metric approach - UMD*, *The Goal Question Metric Approach*. Available at: https://www.cs.umd.edu/users/mvz/handouts/gqm.pdf (Accessed: 04 June 2023).

B, A.K. *et al.* (2002) *Preliminary guidelines for empirical research in software ... - IEEE xplore*, *Preliminary guidelines for empirical research in software engineering*. Available at: https://ieeexplore.ieee.org/document/1027796 (Accessed: 04 June 2023).

Dubey, S.K. and Rana, A. (2011) *Assessment of maintainability metrics for object-oriented software system*, *ACM SIGSOFT Software Engineering Notes*. Available at: https://dl.acm.org/doi/10.1145/2020976.2020983 (Accessed: 04 June 2023).

Abreu, F.B. e. and Melo, W. (1996) *Evaluating the impact of object-oriented design on software quality: Proceedings of the 3rd International Symposium on Software Metrics: From measurement to empirical results*, *Guide Proceedings*. Available at: https://dl.acm.org/doi/10.5555/525586.823874 (Accessed: 04 June 2023).