Object Oriented Development Group Assignment 2

Kasoju Mani Rathnam

Hema Bharath KUMAR

Venkata Ramana Patnam

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

Objectives: The main objective of this study is to analyze the code bad smells that impact the modularity of the software's quality.

Our objective is to establish the metrics using the Goal-Question-Metric (GQM) as well as evaluate the impact of code smells on program modularity by considering a variety of sizes, questions, and metrics.

Goal: Empirical study: The Impact of Bad Smells on Modularity

Questions:

- What factor contributes to the occurrence of code-bad smells that affect the modularity?
- What is the frequency of occurrence of different code smells within the subject programs?

Metrics: We will evaluate the impact of code smells on modularity by using the below metrics.

- **Modularity Metrics**:
    - Coupling: It is the measure of dependencies between the classes or methods that are responsible for any operation.
    - Cohesion: It is the measure of the degree to which the functionalities within a single class or method are related and focused on a specific purpose.

- **Code Smell Occurrence Frequency**: We will measure the number and percentage of code smells that are detected in the subject program.

We wanted to evaluate the interdependence between the classes and the method that is responsible for the code's functionality and the software's modularity by using the metrics provided above. Moreover, these criteria were selected to ensure that the programs under consideration are complex enough and have undergone enough development activity to provide significant information for our study. The class size limitation was set up to ensure that we study commonly used, large programs, and the number of commits criterion was created to ensure that we have sufficient data to analyze.

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

We selected 10 Java projects from GitHub as our subject programs. The below table outlines the main attributes of each program, including name, description, size, number of open issues, and number of commits.

Program Name	Description	Size	Issues	Commits
allure-framework/allure2	Allure Report is a flexible, lightweight test report tool with graphical reports.	11296	322	865
googlemaps	Maps SDK for Android Utility Library.	13301	63	1094
google/auto	Collection of source code generators for Java.	18999	78	1470
widdix/aws-cf-templates	Free Templates for AWS CloudFormation.	12181	18	888
ververica/flink-cdc-connectors	CDC Connectors for Apache Flink®	13399	420	340

Docile-Alligator/Infinity-For-Reddit	A Reddit client for Android	20164	201	1755
skylot/jadx	Dex to Java decompiler	18789	220	1766
Netflix/Priam	Co-Process for backup/recovery, Token Management, and Centralized Configuration management for Cassandra.	13291	47	1465
React-native-svg/react-native-svg	SVG library for React Native, and React Native projects.	16921	338	1768
turms-im/turms	Open-source instant messaging engine for concurrent users.	16202	277	1062

Description:

Project 1: allure-framework/allure2

Allure Report is a test reporting tool that provides graphical reports for multi-language testing processes.

Project 2: googlemaps

Google Maps SDK for android utility library for efficient mapping functionality.

Project 3: google/auto

Java source code generator collection for automated code generation.

Project 4: widdix/aws-cf-templates

Free AWS CloudFormation templates for easy deployment.

Project 5: ververica/flink-cdc-connectors

CDC connectors for Apache Flink for seamless data change capture.

Project 6: Docile-Alligator/Infinity-For-Reddit

Android Reddit client for enhanced browsing experience.

Project 7: skylot/jadx

Dex to Java decompiler for analyzing Android applications.

Project 8: Netflix/Priam

Co-process for backup/recovery, token management, and centralized configuration management for Cassandra.

Project 9: react-native-svg/react-native-svg

SVG library for React Native, React Native Web and React web projects.

Project 10: turms-im/turms

Advanced open-source instant messaging engine for high-concurrency user scenarios.

Section 3: Description of the Tool Used:

Tool 1:

For this study, we used the CK-Code metrics tool to gather the C&K metrics values for the chosen Java projects. The CK metrics offer statistics on things like the number of methods, behavior attributes, and imports. These measurements will aid in our examination of the software's modularity.

We used SonaLint and other static code analysis techniques to find code-bad smells in the study programs. As the code is being generated, SonarLint does a real-time analysis and gives feedback on its quality. Additionally, using the data gathered, we examined the frequency of code smells and how they affected modularity. To evaluate the quality and maintainability of our system, we have mostly employed modularity metrics like coupling and cohesion.

The CK-Code metrics tool can be downloaded from GitHub using the link provided by the authors in the ReadMe file. To use the tool, we followed the instructions provided by the authors, which included setting up the required dependencies and running the tool on the selected Java projects.

The tool uses a command-line interface, and it provides a detailed report for each class in the analyzed Java project, including the values for the selected metrics. We used the tool to obtain the values for the chosen metrics, namely the C&K metrics.

Overall, the CK-Code metrics tool was easy to use and provided accurate and reliable results for the analyzed Java projects. The use of an open-source tool also ensured that the results were transparent and reproducible, which is essential for conducting empirical studies.

Tool 2:
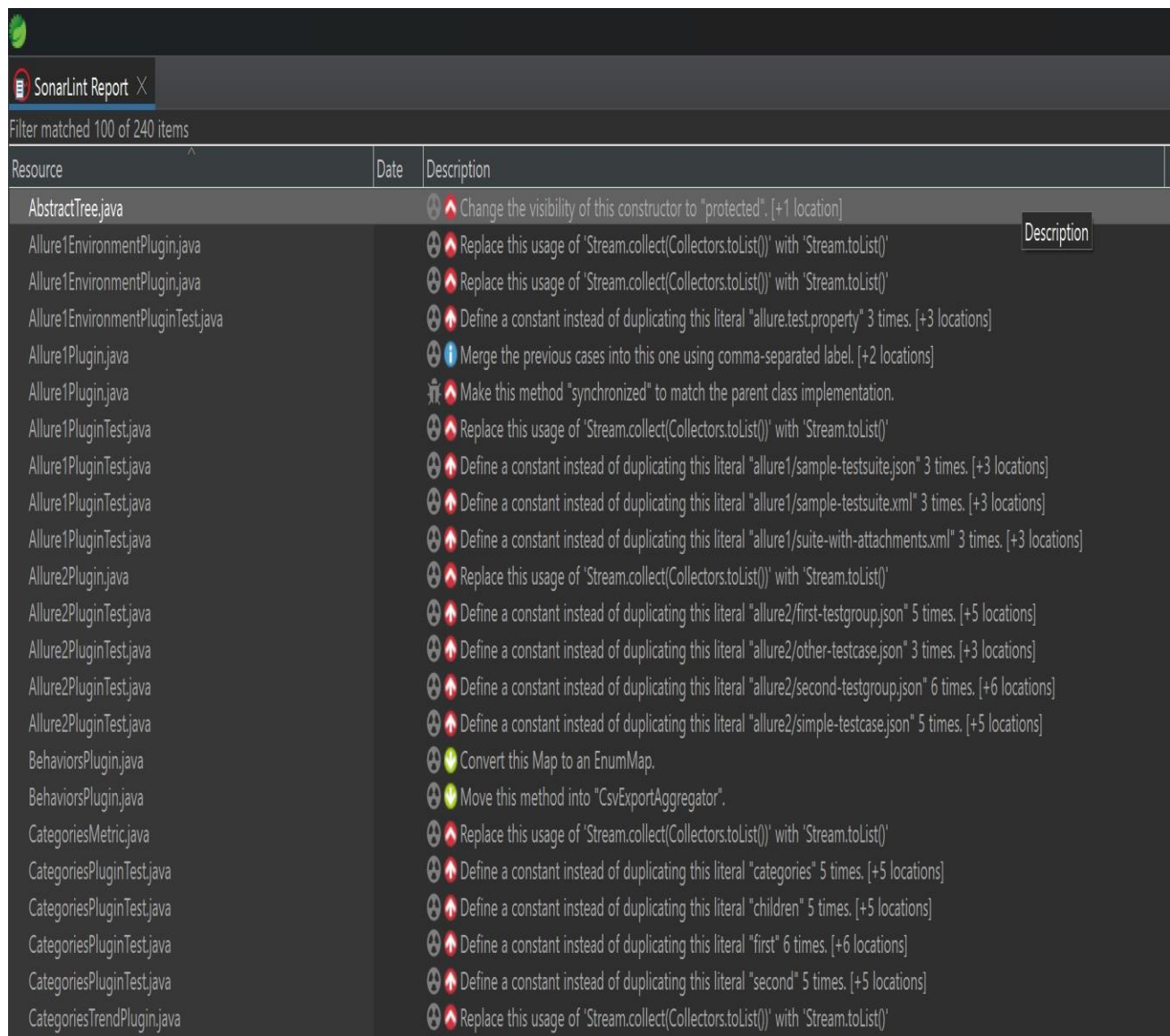
SonarLint is an open-source static code analysis tool that assists developers in identifying and correcting code quality and security concerns during the development process. It may be incorporated into a variety of integrated Development Environments (IDEs), including Eclipse, IntelliJ IDEA, Visual Studio, and others.

 SonarLint analyzes code using a set of criteria and gives real-time feedback to developers on possible issues such as code smells, security vulnerabilities, and other quality-related concerns. The program also gives thorough information about the problem as well as recommendations for how to resolve it.

 One of the key advantages of SonarLint is that it can evaluate code while it is being created, delivering instant feedback to the developer. This can help decrease the time and effort necessary to correct errors later in the development process.

SonarLint also interfaces with SonarQube, an open-source platform for continuous code quality and security analysis. This enables developers to watch the growth of code quality over time, configure quality gates, and guarantee that code quality is maintained throughout the development process.

A Sample SonarLint report
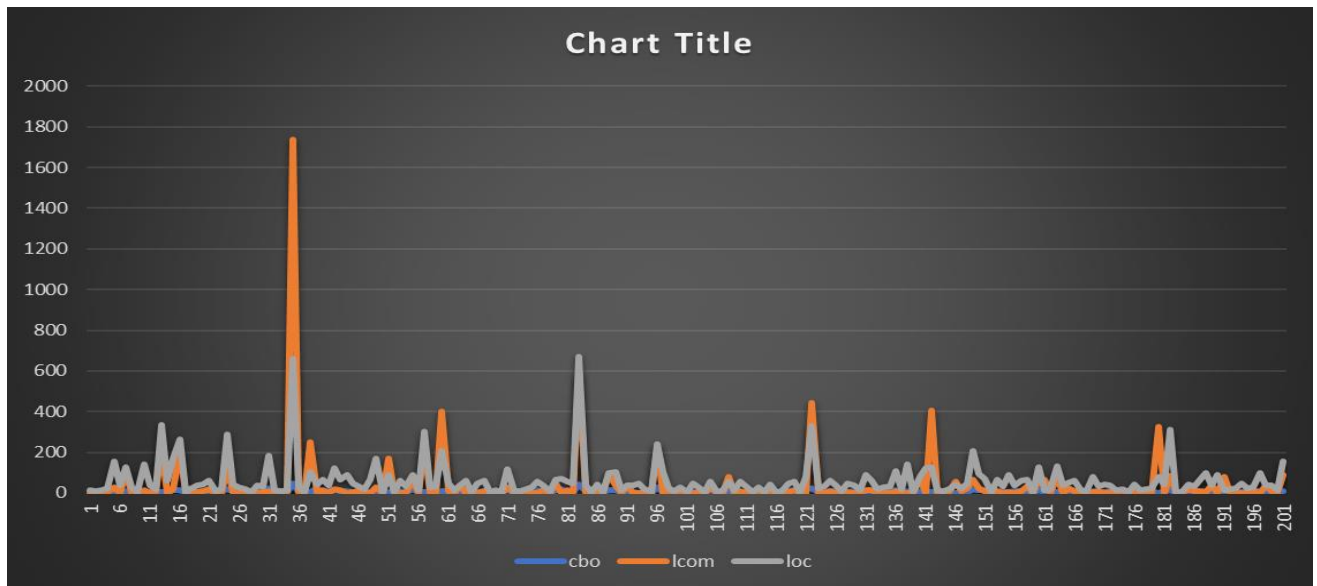
Section 4: Results:

In our empirical study, we used the CK-Code metrics tool to collect the values of C&K metrics for the selected Java projects. The following line charts illustrate the modularity metrics for each project.

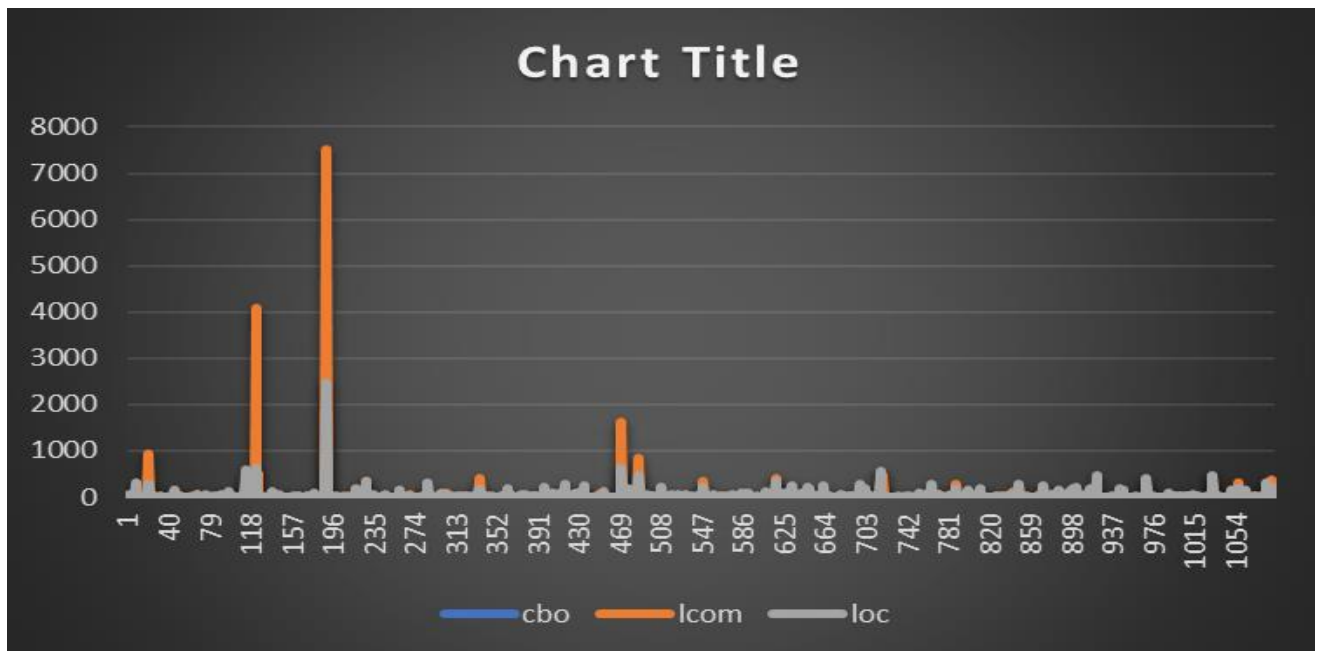## Line Charts for each project:

1. allure-framework/allure2:
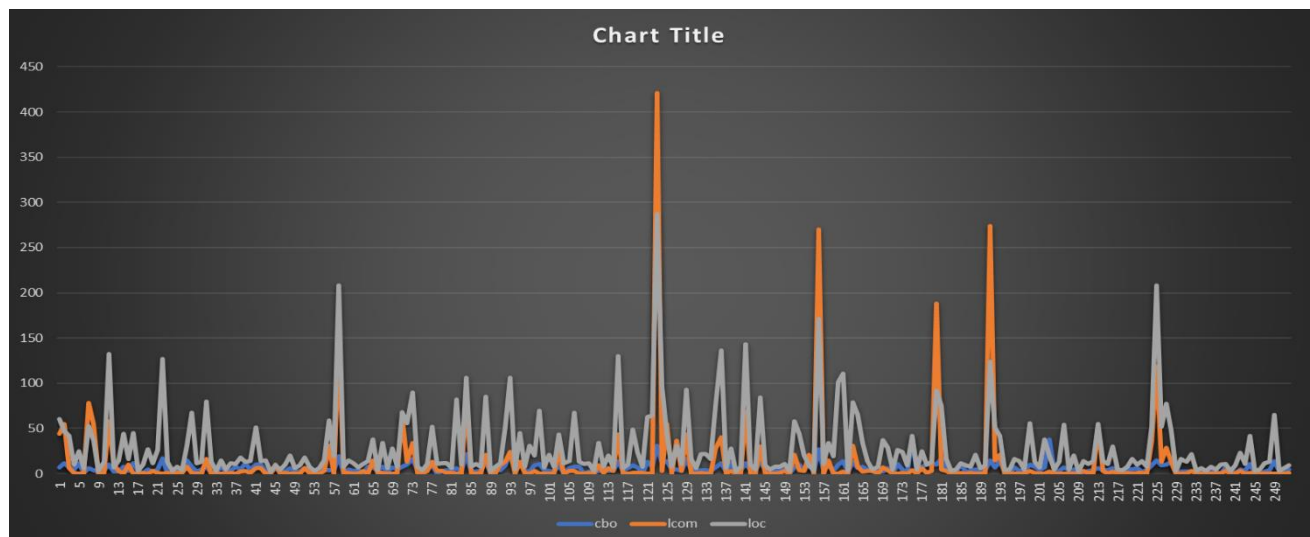
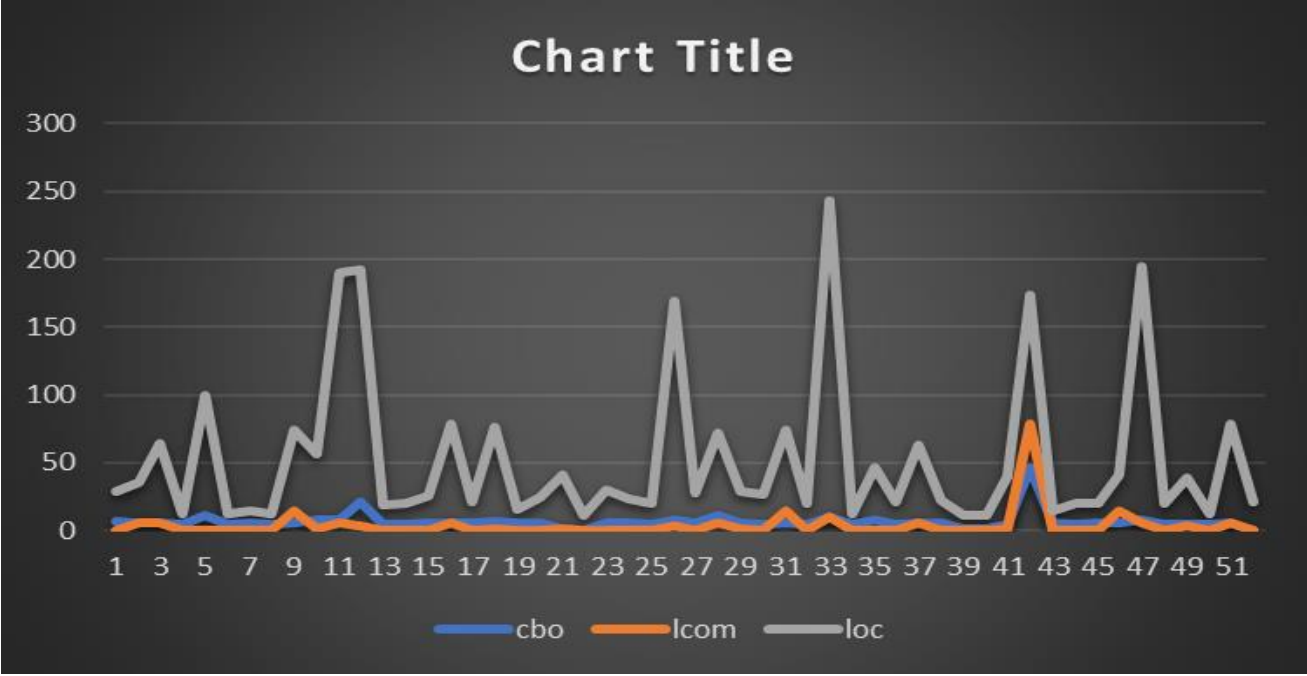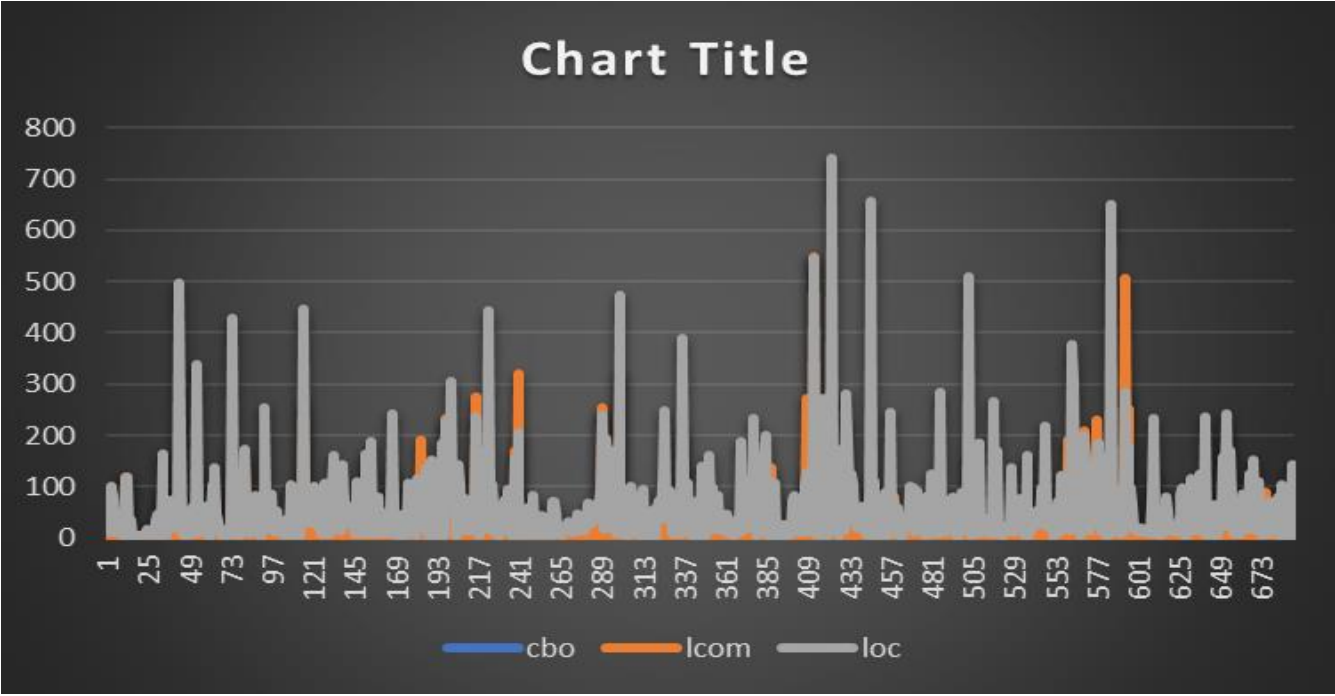### 2. googlemaps :

3. google/auto:

4. widdix/aws-cf-templates:

5. ververica/flink-cdc-connectors:

6. Docile-Alligator/Infinity-For-Reddit:
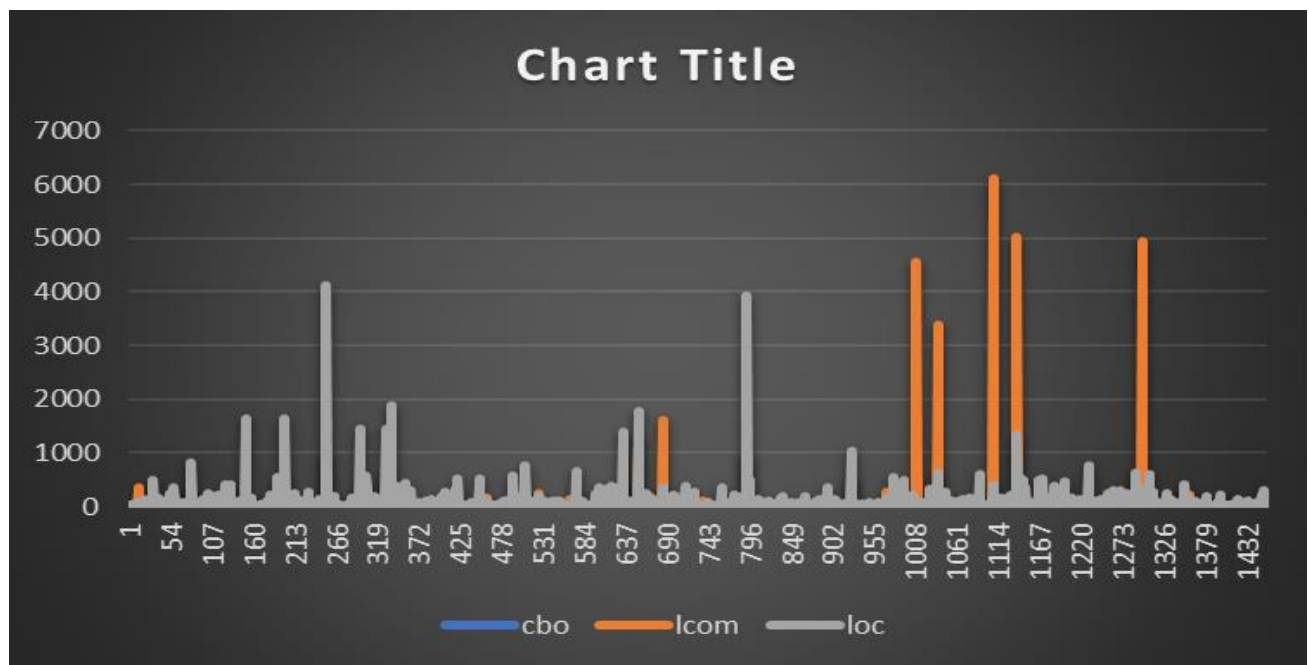
7. skylot/jadx:
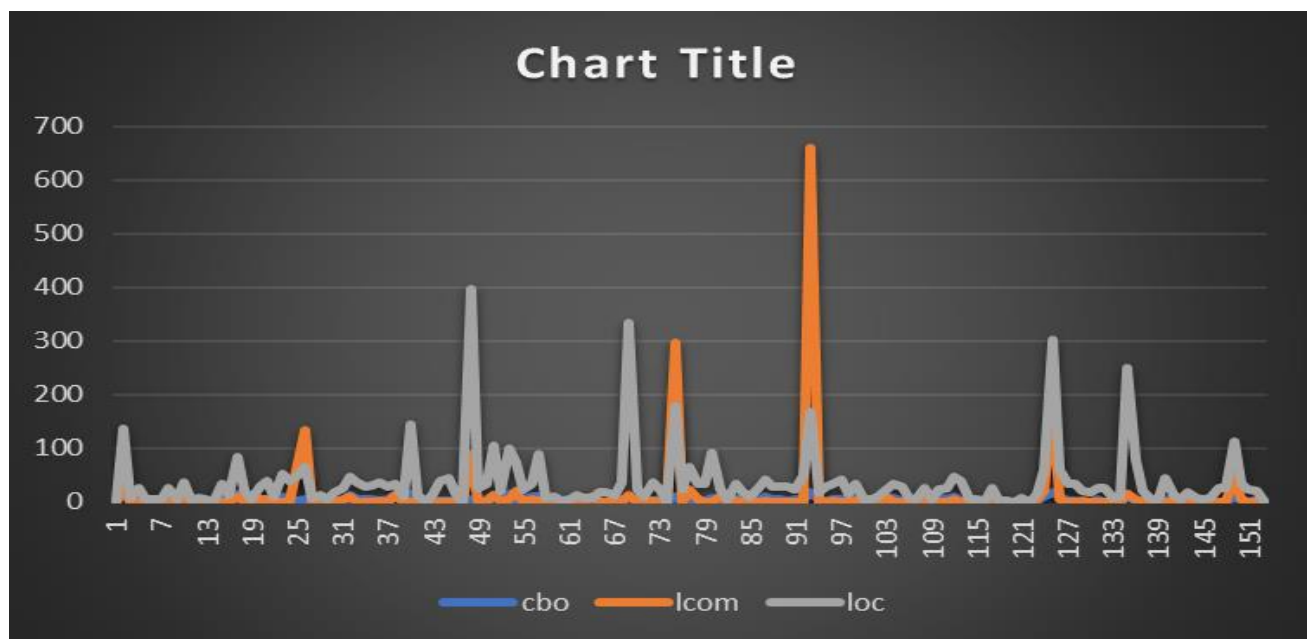
8. Netflix/Priam:
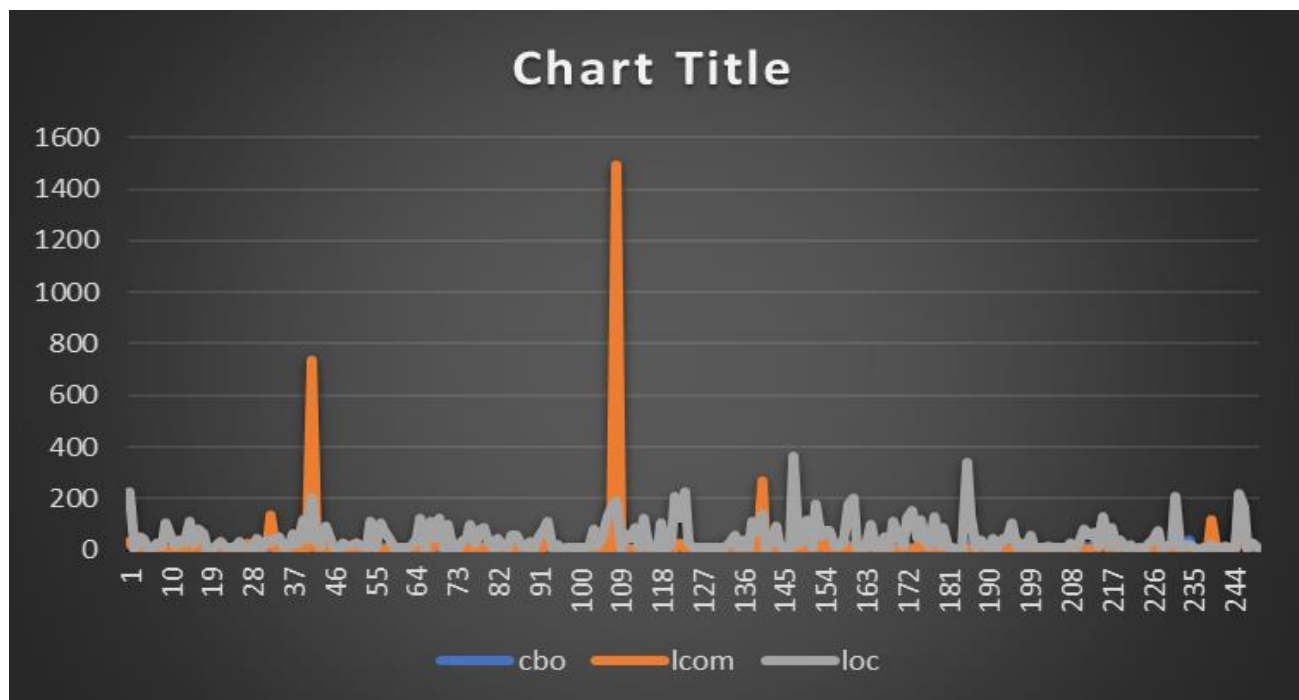
9. react-native-svg/react-native-svg:
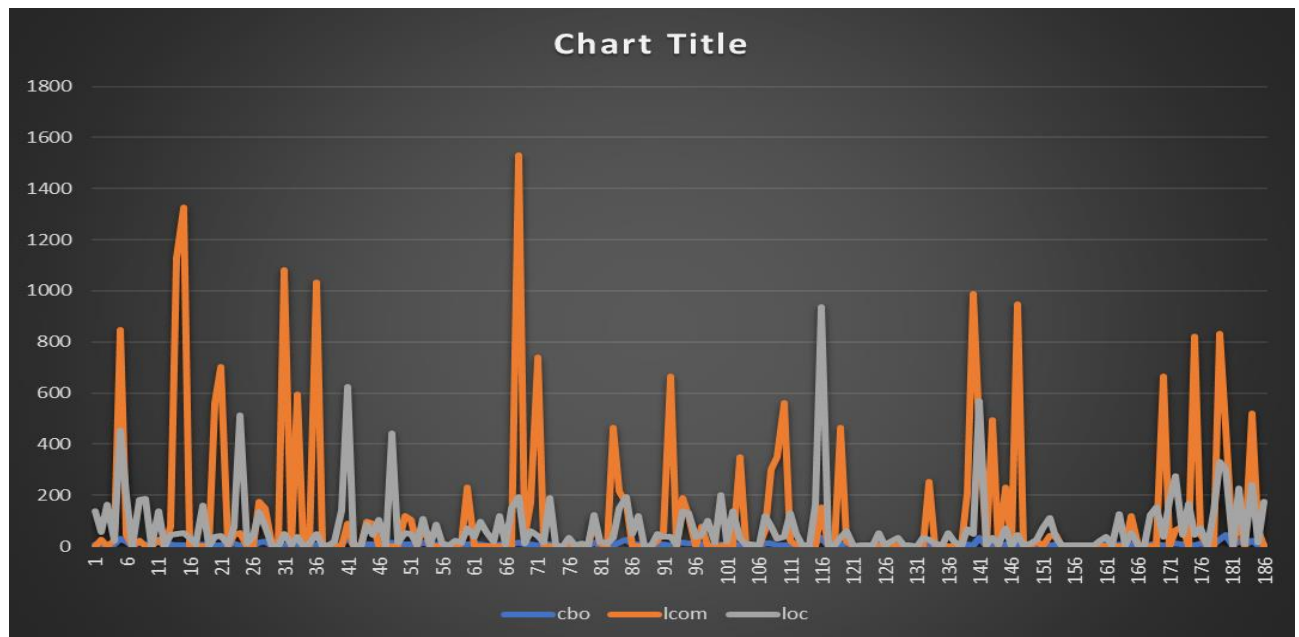
10. turms-im/turms:
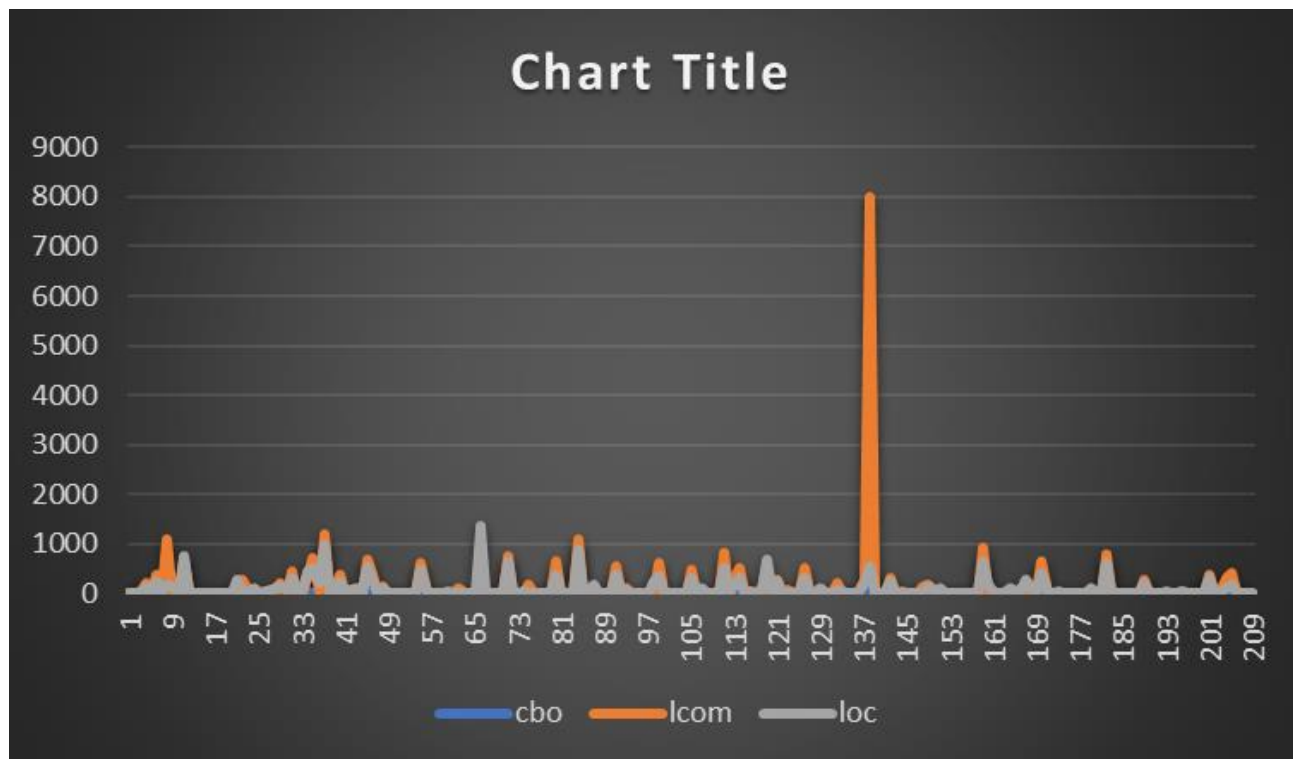
Based on the collected data, we analyzed the occurrence of code smells and their impact on modularity. The results for each project are as follows:

1. allure-framework/allure2:

- Number of bad smells: 89
- Percentage of bad smells: 35.17%

2. googlemaps:

- Number of bad smells: 101
- Percentage of bad smells: 50%

3. google/auto:

- Number of bad smells: 0

- Percentage of bad smells: 0%

4. widdix/aws-cf-templates:

- Number of bad smells: 0
- Percentage of bad smells: 0%

5. ververica/flink-cdc-connectors:

- Number of bad smells: 0
- Percentage of bad smells: 0%

6. Docile-Alligator/Infinity-For-Reddit:

- Number of bad smells: 101
- Percentage of bad smells: 6.9178%

7. skylot/jadx:

- Number of bad smells: 101
- Percentage of bad smells: 4.17%

8. Netflix/Priam:

- Number of bad smells: 101
- Percentage of bad smells: 21.18%

9. react-native-svg/react-native-svg:

- Number of bad smells: 101
- Percentage of bad smells: 54.01%

10. turms-im/turms:

- Number of bad smells: 0
- Percentage of bad smells: 0%

Section 5: Conclusion

Based on the analysis of the selected Java projects, we found variations in the occurrence of code smell and their impact on modularity. Projects like react-native-svg/react-native-svg and Google Maps showed a high percentage of code smells, suggesting the need for attention and effort to improve code quality and maintain modularity.

The projects assessed had varying levels of bad smells. Some projects, such as Docile-Alligator/Infinity-For-Reddit and Skylot/Jadx, have a low percentage of bad-smelling classes, whereas others, such as react-native-svg/react-native-svg and Google Maps, have a significantly higher percentage.

The projects with the highest percentage of bad-smelling classes, such as react-native-svg/react-native-svg and Google Maps, this implies that these projects may require more attention and effort to eliminate bad smells and enhance overall code quality; otherwise, it may have an impact on modularity.

On the other hand, projects such as Docile-Alligator/Infinity-For-Reddit and Skylot/Jadx probably have a low bad smell, so most of these projects are clean for some percent, and because of this low bad smell, it may not have any effect on modularity, and turms-im/turms, ververica/flink-CDC-connectors, widdix/aws-cf-templates, and Google/Auto don't have any code smell, so that their codebase is reasonably clean and well-structured.

Overall, the results emphasize the importance of regular monitoring and addressing code smells to maintain software quality and modularity. By reducing code smells, developers can enhance maintainability, readability, and scalability, contributing to the improved modularity of the software.

References:

Brown, W. J., Malveau, R. C., McCormick, H. W., & Mowbray, T. J. (1998). AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. John Wiley & Sons.

Oizumi, W., Kamei, Y., Ubayashi, N., & Matsumoto, K. (2010). An Empirical Study on the Bad Smell Characteristics. In Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement (ESEM'11).

Nagappan, M., Murphy, B., & Basili, V. (2006). The Influence of Organizational Structure on Software Quality: An Empirical Case Study. In Proceedings of the 28th International Conference on Software Engineering (ICSE'06).

Arisholm, E., Briand, L. C., & Foyen, A. (2006). Dynamic Coupling Measurement for Object-Oriented Software. IEEE Transactions on Software Engineering, 32(6), 391-407.

C&K Github: https://github.com/mauricioaniche/ck/blob/master/README.md