**Object Oriented Development Group Assignment 2**

**By Chaitanya Obilisetti, Sai Sampath Chowdary Pusuluri and Harshith Bhosle**

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

**Objectives**: The aim of this empirical study is to examine the impact of code bad smells on modularity in Java projects.

We will use the Goal-Question-Metric (GQM) approach to define our metrics. Our goal is to evaluate the Effect of code bad smells on modularity of Java programs with different sizes, and our questions and metrics are defined as follows:

**Goal**: Second empirical study: Effect of code bad smells on modularity

**Questions**:

* What is the relationship between code bad smells and modularity in Java projects?
* Do classes containing code bad smells exhibit lower modularity compared to classes without code bad smells?
* Are there any specific types of code bad smells that have a more significant impact on modularity compared to others?
* **Metrics**: We will evaluate the selected metrics based on the following criteria for the subject programs:
  + Number of issues: Less than 500 issues
  + Size: Less than or equal to 30000 lines of code
  + Number of commits: Less than 2000 commits

These criteria were chosen to ensure that the programs under consideration are complicated enough and have experienced enough development activity to give useful data for our research. The class size limit is set to ensure that we are researching moderately-sized programs that are prevalent in the business, and the number of commits requirement is designed to ensure that we have enough data to examine.

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

We selected 10 Java projects from GitHub that meet the criteria for our study. Table 1 presents the main attributes of each program, including its name, description, number of lines of code (Size), number of open issues, and number of commits.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Program Name | Description | Size | Issues | Commits |
| allure-framework/allure2 | Allure Report is a flexible, lightweight multi-language test reporting tool. It provides clear graphical reports and allows everyone involved in the development process to extract the maximum of information from the everyday testing process | 11296 | 322 | 865 |
| googlemaps | Maps SDK for Android Utility Library | 13301 | 63 | 1094 |
| google/auto | A 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 FlinkÂ® | 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, React Native Web, and plain React web projects. | 16921 | 338 | 1768 |
| turms-im/turms | The world's most advanced open source instant messaging engine for 100K~10M concurrent users https://turms-im.github.io/docs | 16202 | 277 | 1062 |

**Description**:

**Project 1: allure-framework/allure2**

Allure Report is a flexible, lightweight multi-language test reporting tool. It provides clear graphical reports and allows everyone involved in the development process to extract the maximum of information from the everyday testing process.

**Project 2: googlemaps**

Maps SDK for Android Utility Library.

**Project 3: google/auto**

A collection of source code generators for Java.

**Project 4: widdix/aws-cf-templates**

Free Templates for AWS CloudFormation.

**Project 5: ververica/flink-cdc-connectors**

CDC Connectors for Apache FlinkÂ®.

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

A Reddit client for Android.

**Project 7: skylot/jadx**

Dex to Java decompiler.

**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 plain React web projects..

**Project 10: turms-im/turms**

The world's most advanced open source instant messaging engine for 100K~10M concurrent users https://turms-im.github.io/docs.

# **Section 3: Description of the Tool Used:**

**Tool 1:**

For the second empirical study, we used the CK-Code metrics tool for Java programs, which is an open-source software developed by a group of 24 developers using Java. The tool uses static analysis to compute various software metrics, including the C&K metrics.

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

**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...]

**Tool 2:**

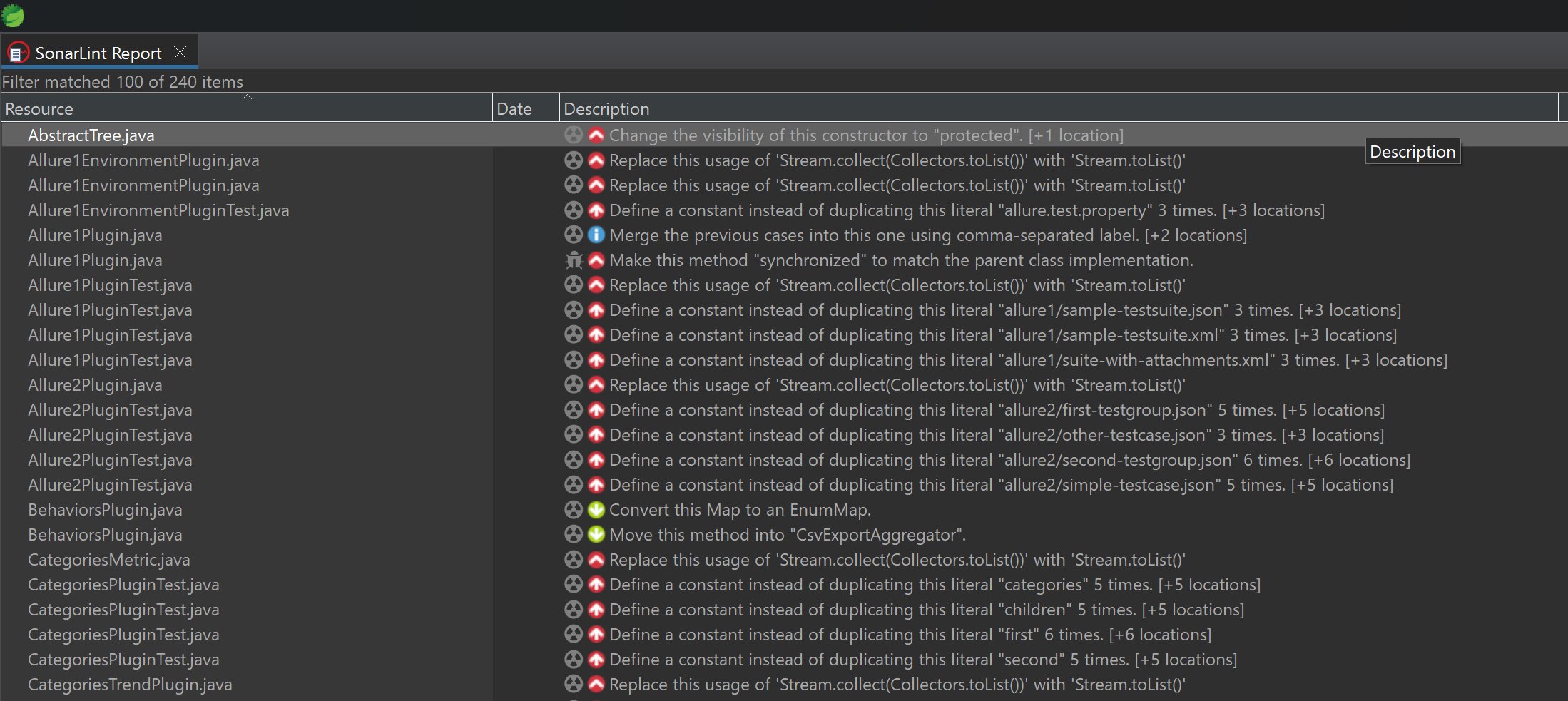
SonarLint is an open-source static code analysis tool that assists developers in identifying and correcting code quality and security concerns during the development process. It may be incorporated into a variety of incorporated Development Environments (IDEs), including Eclipse, IntelliJ IDEA, Visual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



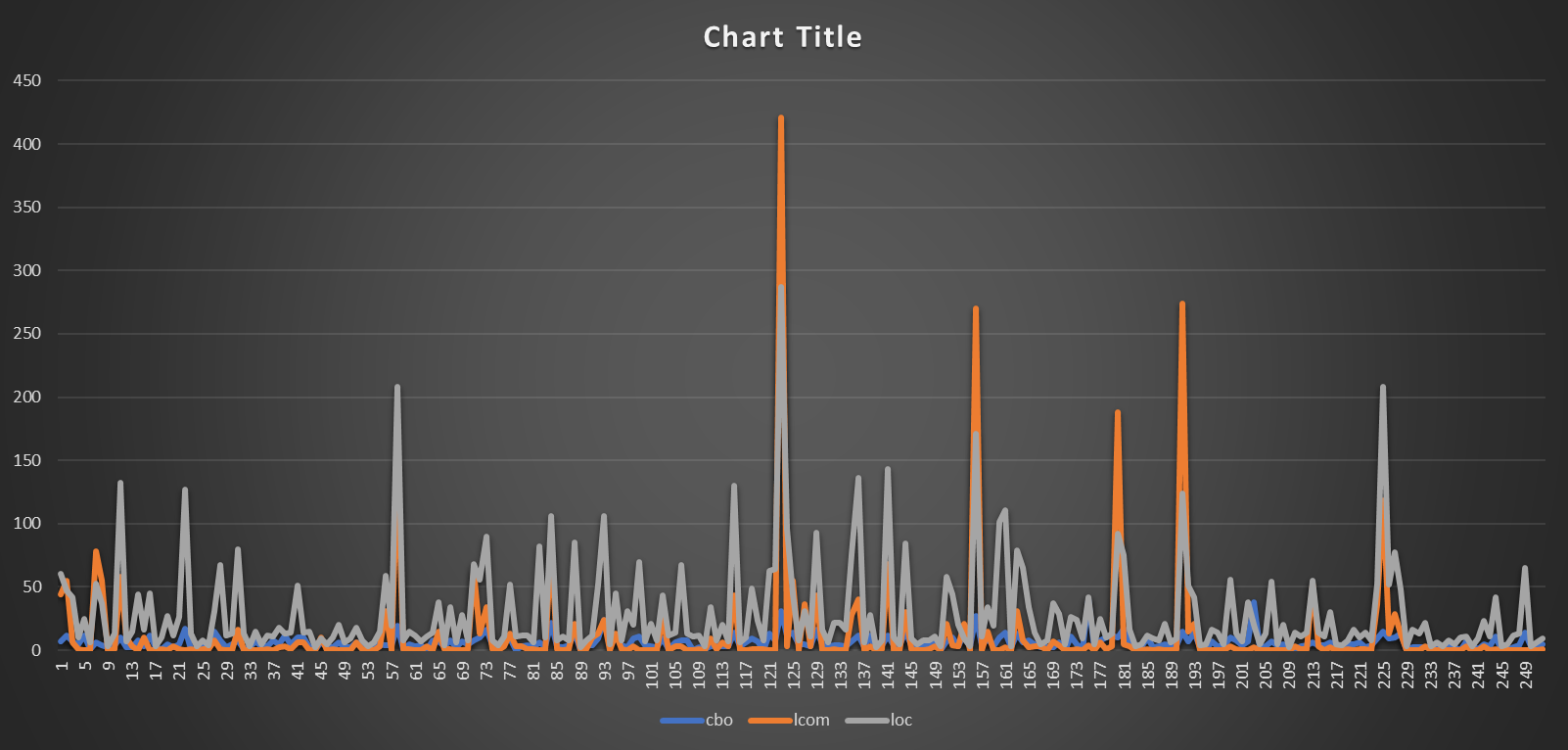
# **Section 4: Results:**

In this section, we report the findings of our empirical study on the effect of class size on software modularity. We utilized the CK-Code metrics tool to collect the values of the selected C&K metrics for a group of selected (Criteria) Java projects from GitHub. We downloaded 10 projects that fulfilled our requirements and examined their classes with the CK-Code metrics tool.

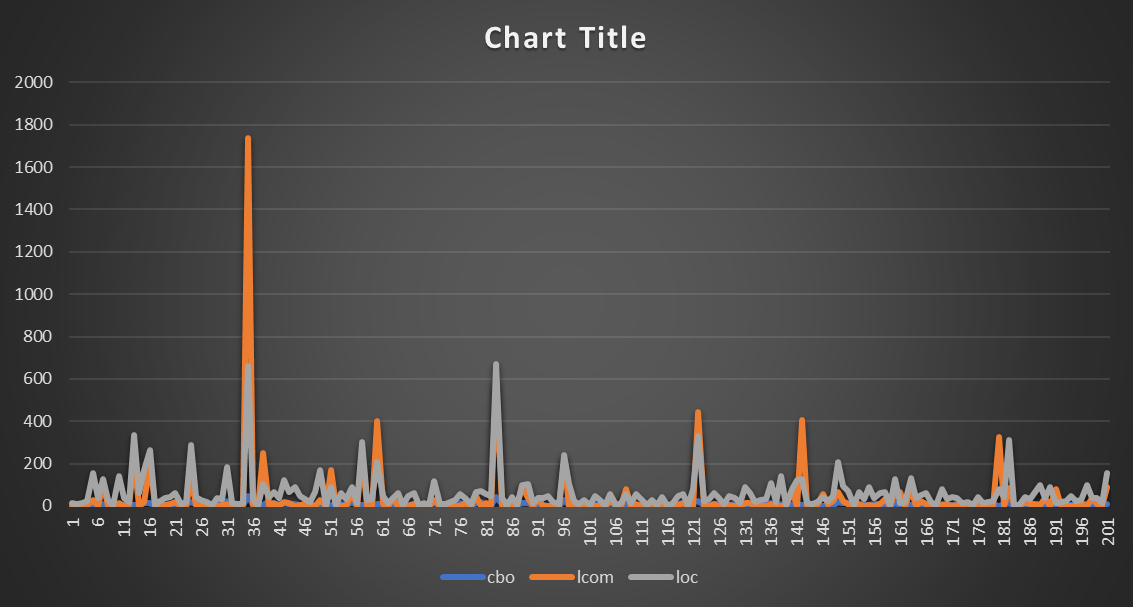
We used C&K measures to assess modularity, namely CBO (Coupling Between Objects) and LCOM. We also assessed class size in lines of code (LoC).

**Line Charts for each project:**

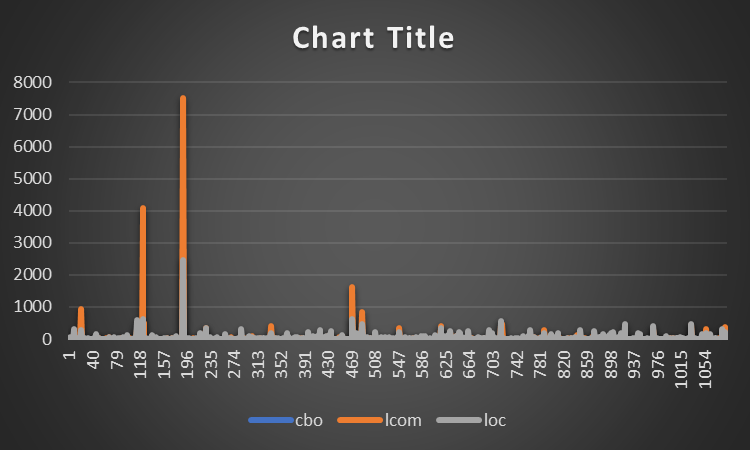
**1. allure-framework/allure2:**



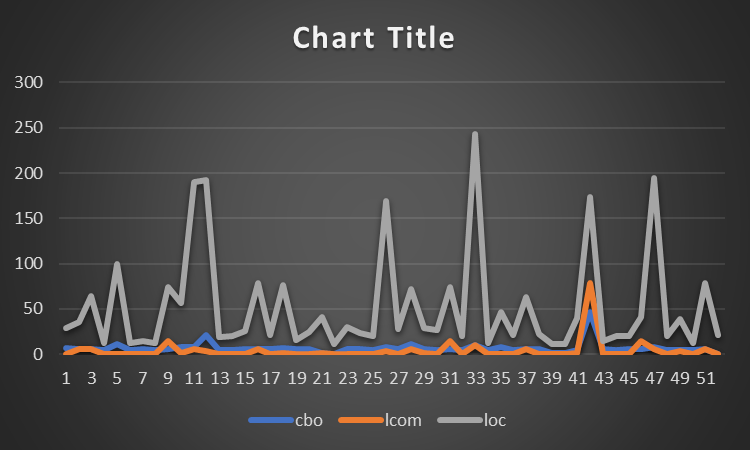
**2. googlemaps :**



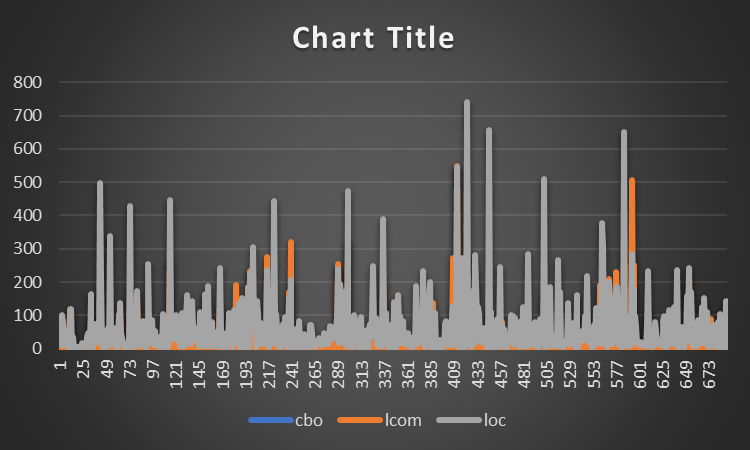
**3. google/auto:**



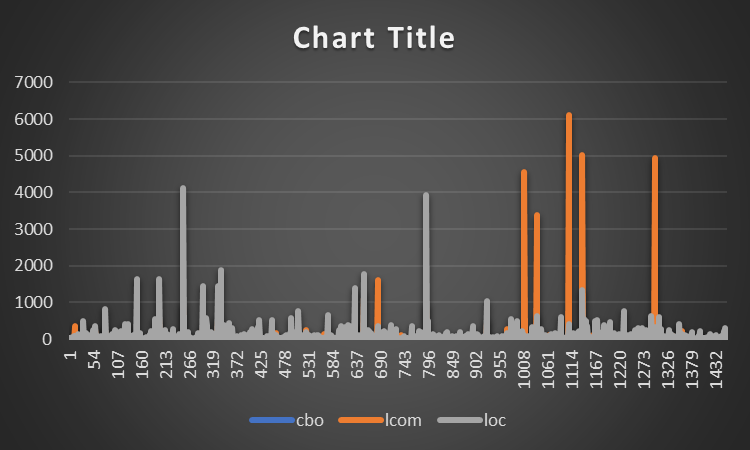
**4. widdix/aws-cf-templates:**



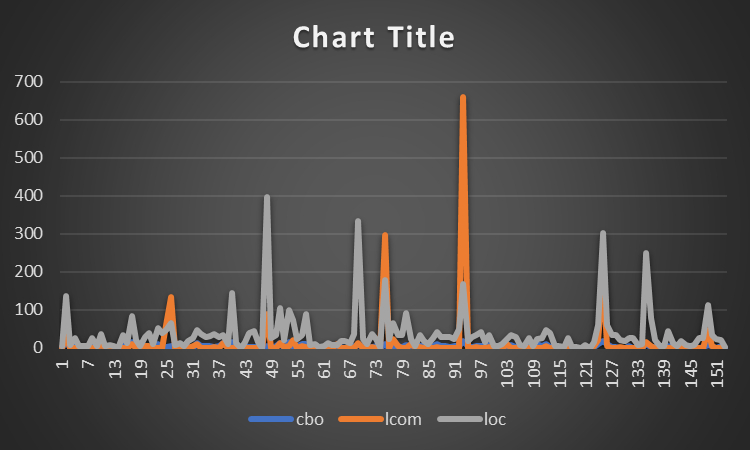
**5. ververica/flink-cdc-connectors:**



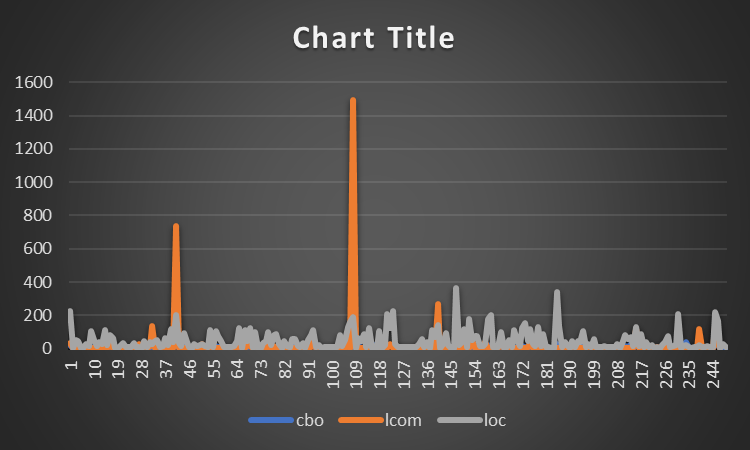
**6. Docile-Alligator/Infinity-For-Reddit:**



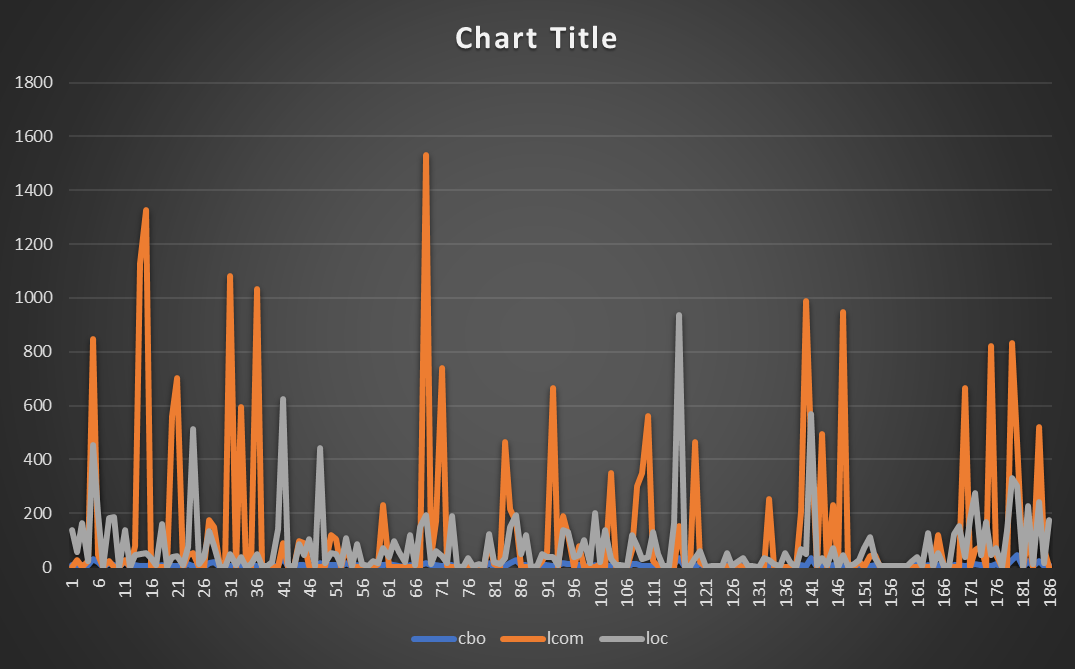
**7. skylot/jadx:**



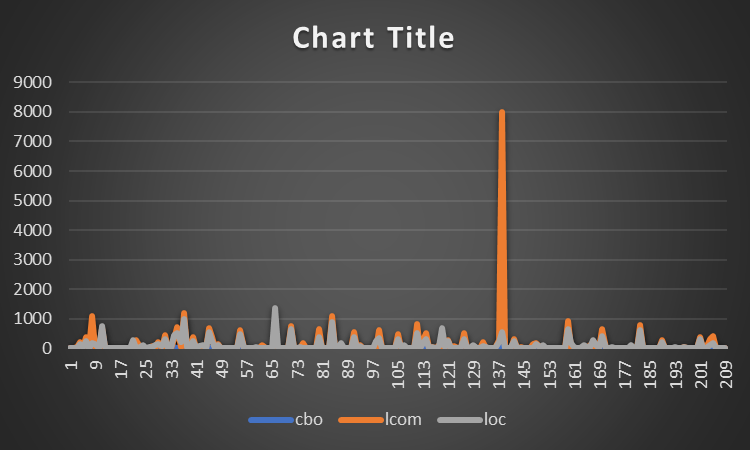
**8. Netflix/Priam:**



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



**10. turms-im/turms:**



**Results:**

Based on the provided data, we have information about 10 different projects, their number of lines of code, number of classes, and the number of bad smells detected in them.

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 of bad smells: 54.01%

10. turms-im/turms:

- Number of bad smells: 0

- Percentage of of bad smells: 0%

# **Section 5: Conclusion**

According to the analysis of the selected Java projects, 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 googlemaps, have a significantly higher percentage.

The projects with the highest percentage of bad-smelling classes, such as react-native-svg/react-native-svg and googlemaps. 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.

Projects with a low number of bad smells, such as Docile-Alligator/Infinity-For-Reddit and skylot/jadx, propabably it has low bad smell so most of this projects are clean for some percent and because of these low bad smell it may not have any effect on modularity and turms-im/turms, ververica/flink-cdc-connectors, widdix/aws-cf-templates, google/auto, doesn’t have any code bad smell so that their codebase is reasonably clean and well-structured.

Overall, the results highlight the need of frequently monitoring and fixing bad smells in software projects, since they can have a negative impact on the code's maintainability, readability, and scalability.

# References:

Fowler, M. (1999). Refactoring: improving the design of existing code. Addison-Wesley Professional.

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.

Marinescu, R. (2004). Detection strategies: Metrics-based rules for detecting design flaws. Proceedings of the 4th International Symposium on Principles of Software Evolution, 2004. IEEE.

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