Assignment 1 Report

ANALYZE THE EFFECT OF SIZE ON MAINTAINABILITY

IN JAVA PROJECTS

**Group Members:**

1. Nandini Badavath
2. Chaitanya Anurag Turlapati

# Section 1:

## Objective:

The primary goal of this research is to learn how the number of lines of code (LOC) in a Java project affects its ease of maintenance. The software's efficiency, effectiveness, and security must be maintained throughout its existence, hence software maintenance is an essential part of software development. To help developers make more educated decisions during the design and construction phases of their projects, we can examine the correlation between size and maintainability to reveal trends or features that affect software maintainability.

## Question:

In order to accomplish this, we will need to provide an answer to the following question. Can the number of lines of code (LOC) that make up a Java project be used to gauge how easy it will be to keep updated? The elements that contribute to a software project's maintainability can be better understood by examining a wide range of Java projects with varying sizes and characteristics in order to spot patterns, correlations, and trends.

## Metrics:

We have chosen a set of measures to assess maintainability that are based on the well-known and extensively used Chidamber and Kemerer (C&K) Metrics Suite. These measurements are:

1. CBO (Coupling Between Object Classes): This metric measures the number of classes that a class is coupled to. Because modifications in one class may necessitate changes in other classes, high coupling between classes can make a system harder to maintain.
2. RFC (Response For a Class): This statistic counts how many methods an object of a class can execute in response to a message. A higher RFC value indicates a more complex class, which may require more time and effort to test and troubleshoot.
3. LCOM\* (Lack of Cohesion of Methods): This metric assesses how tightly connected a class's methods are to one another. A high LCOM\* number suggests that the class is attempting to accomplish too many things at once, which makes it more difficult to comprehend and maintain.
4. Lines of Code (LOC): This statistic gauges the scope of a project in terms of lines of code. Even though it's a basic statistic, it's helpful for comparing projects and getting a sense of how complex they are.

We can learn a lot about the maintainability of Java projects and how size impacts different aspects of maintainability by using these metrics.

# Section 2:

## Dataset Description:

To find out how project size affects how easy it is to maintain Java projects, we chose five projects that met the following criteria:

* The project has to have at least 10,000 lines: This criterion makes sure that there is a lot of code to look at in the projects. Larger projects usually have more complicated structures and interactions between their parts, which makes them harder to keep up. We can learn more about how size affects maintainability by focusing on projects with at least 10,000 lines of code.
* The project must have been going for at least 3 years: This criterion makes sure that the projects have done things to make them easier to maintain, like fixing bugs, refactoring, and adding new features. Older projects have a longer history of development, which gives us valuable information about how well they can be maintained over time.
* The project needs at least 3 people to work on it: This criterion makes sure that the projects use and contribute to development in different ways. When there are multiple developers working on a project, there are more likely to be different coding styles and ways of doing things, which can make the project harder to maintain. When we look at projects with at least three developers, we can see how different ways of making software affect how easy it is to maintain.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Project** | **GitHub URL** | **Description** | **Lines of Code** | **Developers** |
| Guava | https://github.com/google/guava | A set of core Java libraries providing utility and convenience methods | 150,000 | 284 |
| Dubbo | https://github.com/apache/dubbo | A high-performance, Java-based RPC framework | 180,000 | 492 |
| Ghidra | https://github.com/NationalSecurityAgency/ghidra | A software reverse engineering (SRE) framework and tool suite | 1,000,000 | 229 |
| AndroidUtilCode | https://github.com/Blankj/AndroidUtilCode | A collection of utility classes to simplify Android development | 12,000 | 39 |
| Signal-Android | https://github.com/signalapp/Signal-Android | An open-source, private messenger application for Android devices | 65,000 | 274 |

These projects have a wide range of sizes, ages, and numbers of developers. This lets us look at how size affects maintainability in different situations. By looking at these projects, we can find patterns, trends, and correlations that will help us figure out what makes a software project easy to maintain.

# Section 3:

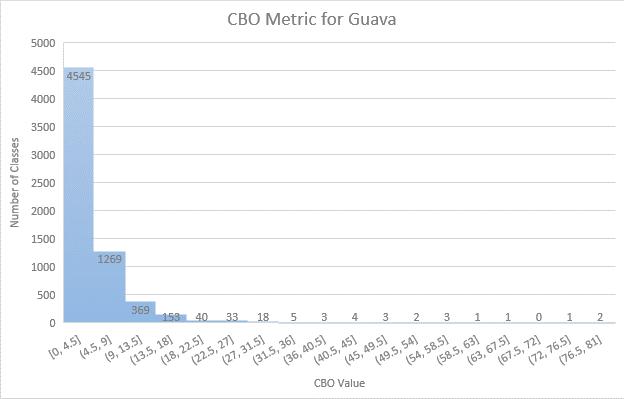
## Tool Used:

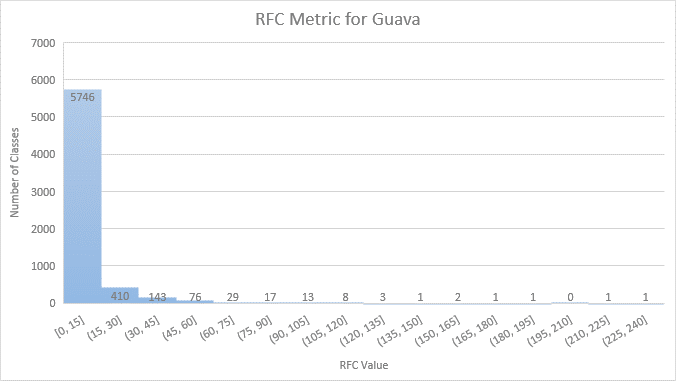
To obtain the C&K metrics measurements for the selected Java projects, we used the CK (Chidamber and Kemerer) metrics tool, specifically the CKJM-Extended tool. CKJM-Extended is an open-source Java-based tool that calculates a variety of software metrics, including CBO, RFC, LCOM\*, and LOC. The CKJM-Extended tool calculates the C&K metrics for each class in a Java project and exports the results in an XML format. The tool can also generate reports in HTML and CSV formats, making it easy to analyze and visualize the results. One advantage of using the CKJM-Extended tool is that it is easy to use and requires no prior knowledge of the C&K metrics suite. The tool provides a simple command-line interface and can be run on any machine with a Java runtime environment installed. Additionally, the CKJM-Extended tool is open-source, which means that users can view and modify the source code as needed.

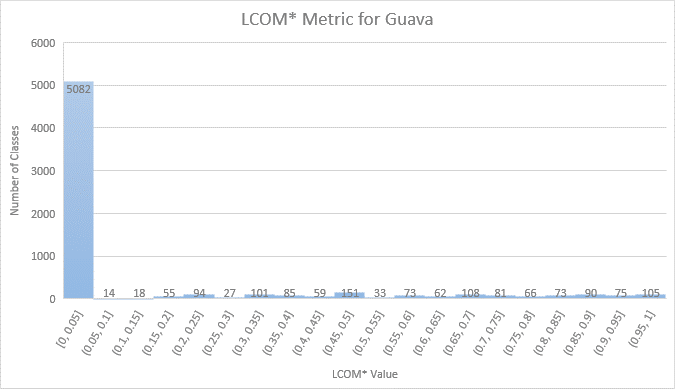
# Section 4:

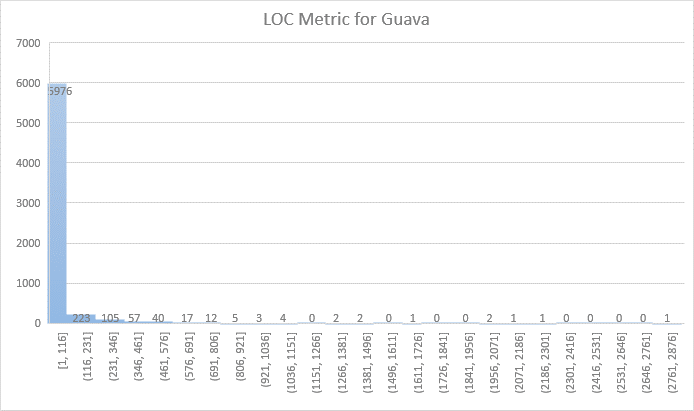
## Results:

### Project 1 Guava:



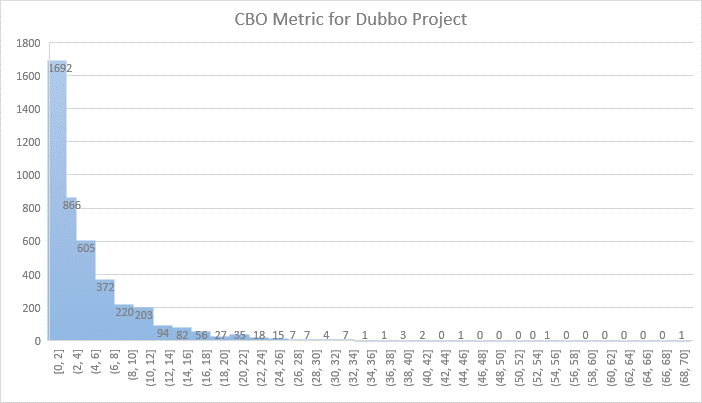


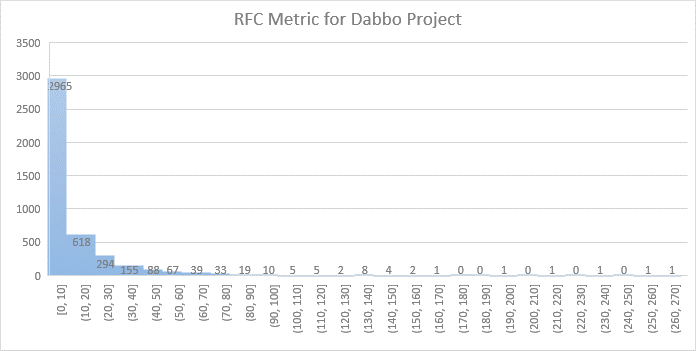


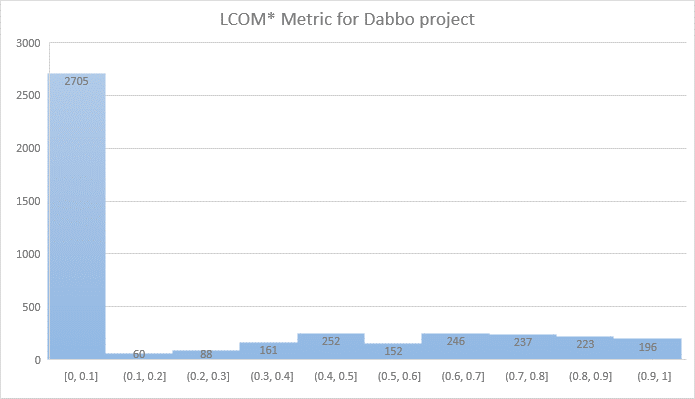


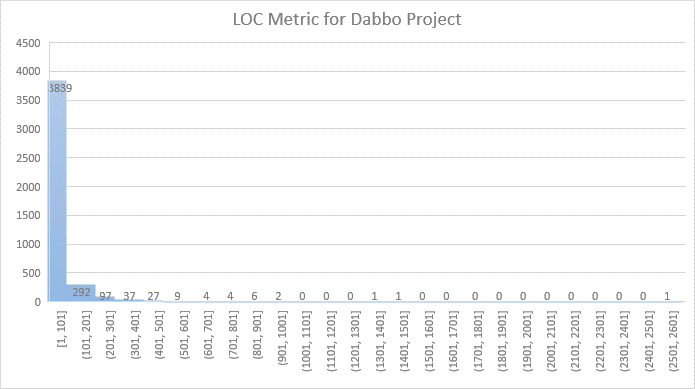
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CK Metric** | **Max** | **Mode** | **Median** | **Average** | **Std Deviation** |
| *CBO* | 78 | 0 | 2 | 3.74938 | 5.3381 |
| *RFC* | 228 | 0 | 2 | 6.375697 | 13.64862 |
| *LCOM\** | 1 | 0 | 0 | 0.125601 | 0.267061 |
| *LOC* | |  | | --- | | 2828 | | |  | | --- | | 5 | | |  | | --- | | 40.74985 | | |  | | --- | | 9 | | |  | | --- | | 117.939 | |

### Project 2 Dabbo:



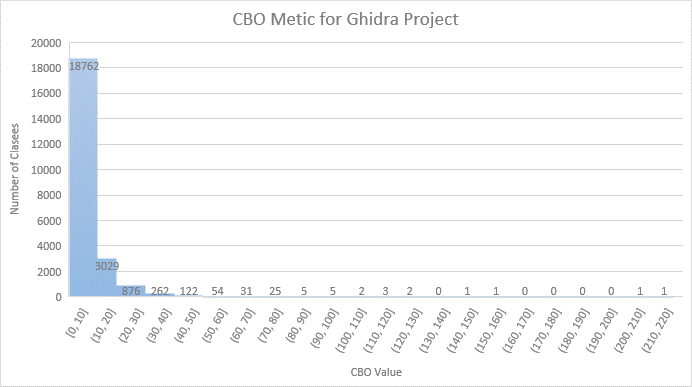


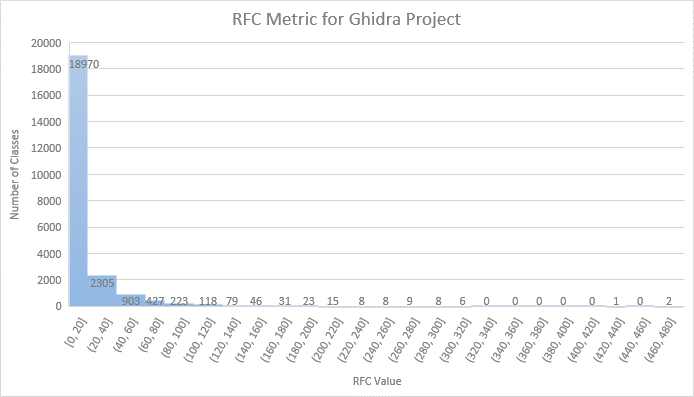


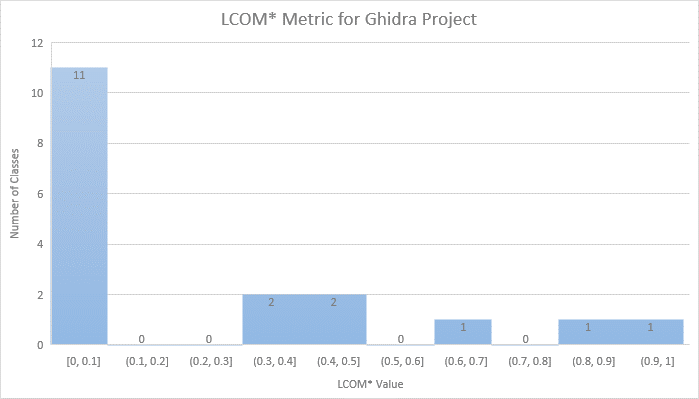


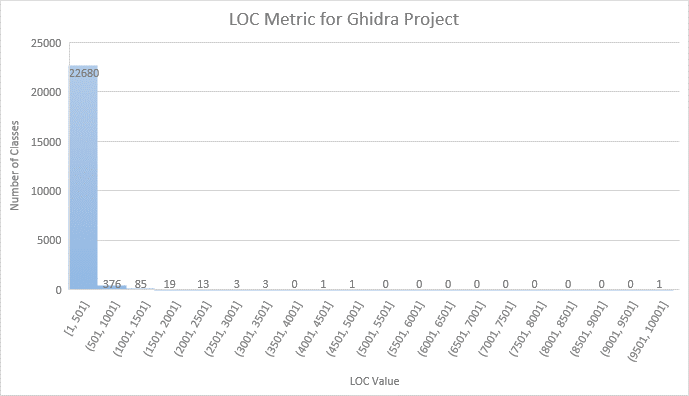
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CK Metric** | **Max** | **Mode** | **Median** | **Average** | **Std Deviation** |
| *CBO* | 70 | 1 | 4 | 5.109722222 | 5.488793121 |
| *RFC* | 264 | 0 | 4 | 11.29722222 | 19.88989581 |
| *LCOM\** | 1 | 0 | 0 | 0.240201836 | 0.335151108 |
| *LOC* | |  | | --- | | 2589 | | |  | | --- | | 5 | | |  | | --- | | 47.41064815 | | |  | | --- | | 18 | | |  | | --- | | 95.73662453 | |

### Project 3 Ghidra:



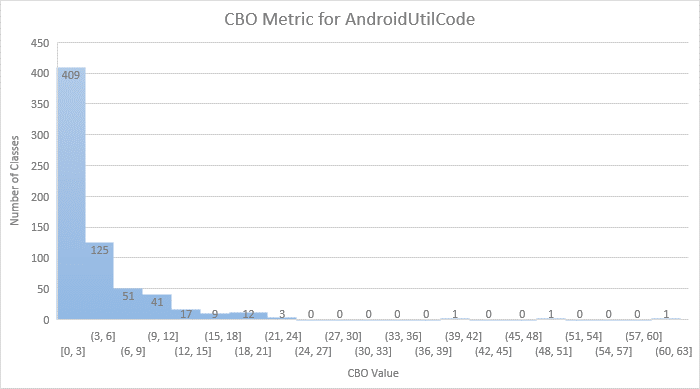


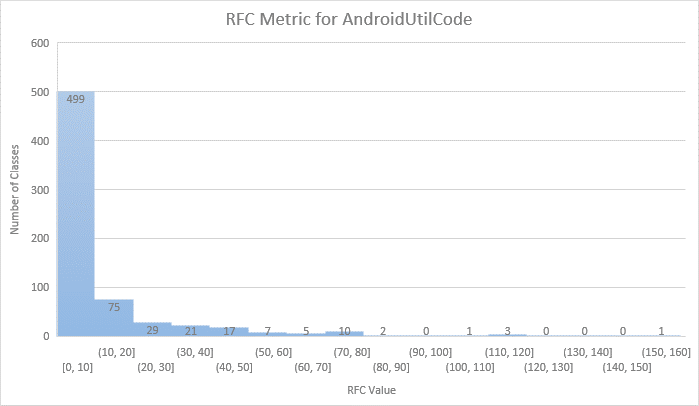


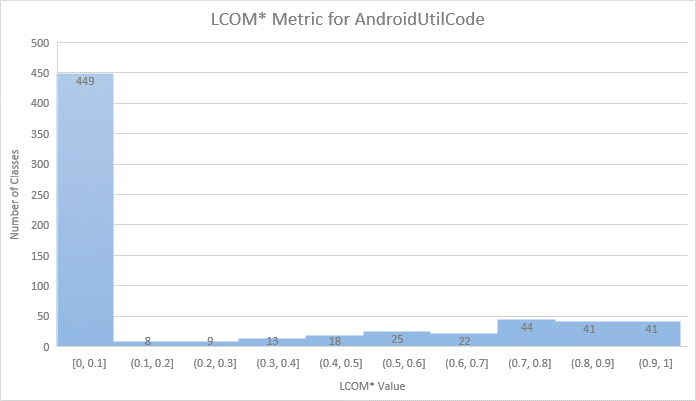


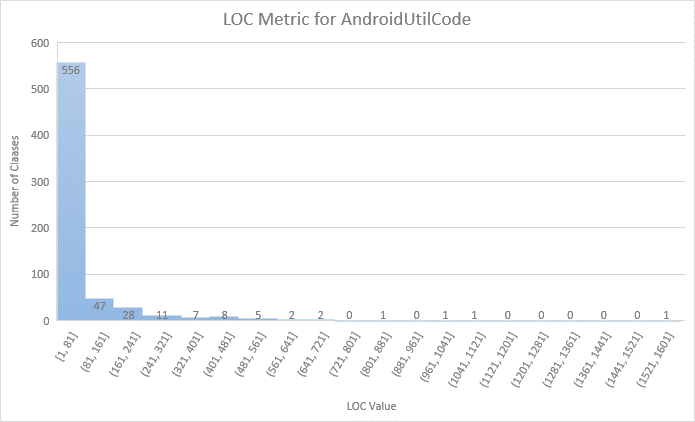
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CK Metric** | **Max** | **Mode** | **Median** | **Average** | **Std Deviation** |
| *CBO* | 213 | 2 | 4 | 6.977568803 | 8.768286869 |
| *RFC* | 479 | 0 | 5 | 13.36834613 | 25.05409529 |
| *LCOM\** | 1 | 0 | 0 | 0.302777 | 0.354753 |
| *LOC* | |  | | --- | | 9774 | | |  | | --- | | 5 | | |  | | --- | | 26 | | |  | | --- | | 72.46229833 | | |  | | --- | | 174.1953656 | |

### Project 4 AndroidUtilCode:



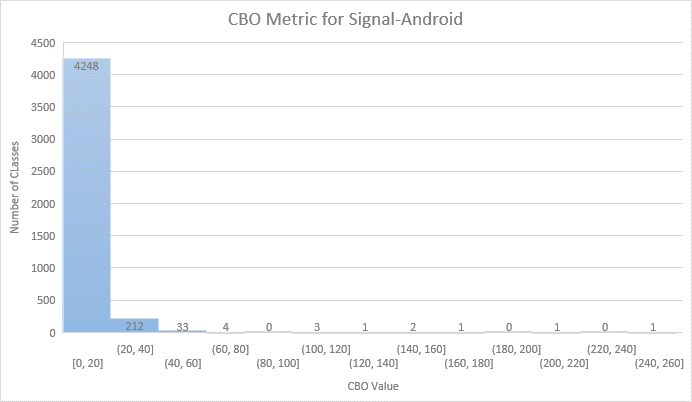


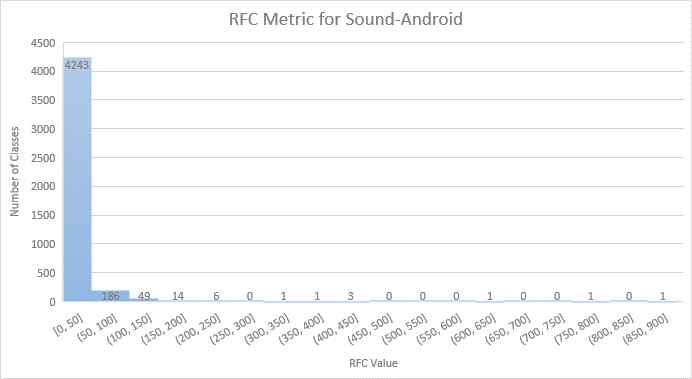


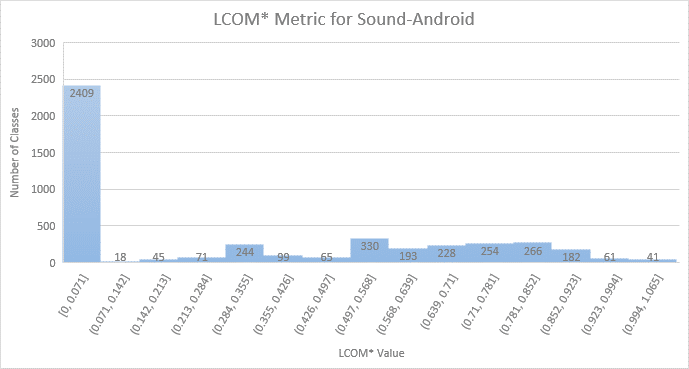


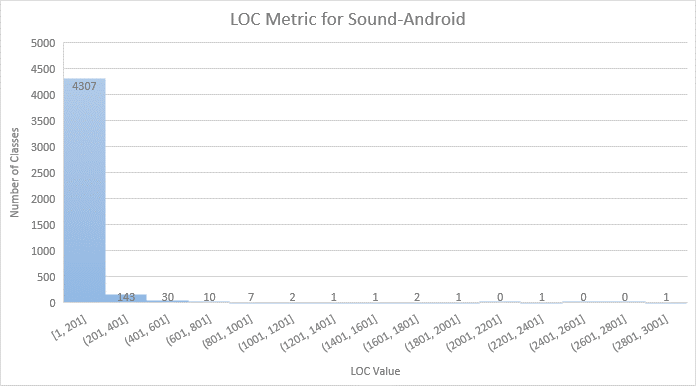
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CK Metric** | **Max** | **Mode** | **Median** | **Average** | **Std Deviation** |
| *CBO* | 63 | 0 | 2 | 4.165671642 | 5.557113291 |
| *RFC* | 155 | 0 | 3 | 9.908955224 | 18.17162978 |
| *LCOM\** | 1 | 0 | 0 | 0.23633425 | 0.353032099 |
| *LOC* | |  | | --- | | 1541 | | |  | | --- | | 5 | | |  | | --- | | 14 | | |  | | --- | | 59.65074627 | | |  | | --- | | 129.6204463 | |

### Project 5 Signal-Android:









|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CK Metric** | **Max** | **Mode** | **Median** | **Average** | **Std Deviation** |
| *CBO* | 248 | 2 | 4 | 6.801154017 | 10.34463645 |
| *RFC* | 867 | 0 | 4 | 12.97092765 | 32.67990662 |
| *LCOM\** | 1 | 0 | 0 | 0.286117 | 0.337656 |
| *LOC* | |  | | --- | | 2982 | | |  | | --- | | 5 | | |  | | --- | | 19 | | |  | | --- | | 49.87483356 | | |  | | --- | | 114.5415662 | |

# Section 5:

## Conclusion:

After looking at the C&K metrics for the five Java projects, we can say a few things about how size affects how easy it is to maintain. Overall, the results suggest that there is a moderate link between size and maintainability, as measured by the chosen metrics. First of all, the results show that the CBO, RFC, and LCOM\* metrics are very different between the projects. The highest possible values for these metrics range from 63 to 867. This means that some classes in these projects are tightly linked, have a lot of methods, and don't work well together. On the other hand, the mode, median, and average values for these metrics are all pretty low. This means that most classes in these projects have low to moderate coupling, a small number of methods, and a high level of cohesion. Based on these results, it looks like some classes in a project might be hard to maintain, but most classes are well-designed and easy to maintain.

Second, the results show that there is a moderately positive correlation between size (measured by LOC) and the CBO, RFC, and LCOM\* metrics. In general, CBO and RFC values tend to be higher for bigger projects, while LCOM\* values tend to be lower. This means that bigger projects are more complicated and harder to maintain. This connection is not always easy to see, though. For example, the AndroidUtilCode project only has 12,000 lines of code, but its CBO and RFC values are high. This means that some of the classes in this project are tightly connected and have a lot of methods.

Third, the results show that the Guava project is the easiest to maintain compared to the others. The CBO, RFC, and LCOM\* metrics for the Guava project have relatively low maximum values. This means that the classes in this project are well-designed and easy to keep up to date. This result might not be too surprising, since Guava is a set of core Java libraries made by Google, which is known for its strong engineering practices and focus on code quality.

In conclusion, the results show that there is a moderate relationship between the size of a Java project and how easy it is to maintain, but this relationship is not simple, and there are many other factors that affect how easy software is to maintain. No matter how big a project is, developers should focus on making classes with low coupling, a small number of methods, and a high level of cohesion. Also, it is important to put code quality and maintainability at the top of the development process, no matter how big or complicated the project is.

# References:

1. Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 20(6), 476–493. <https://doi.org/10.1109/32.295895>
2. Basili, V., Caldiera, G., & Rombach, H. (n.d.). THE GOAL QUESTION METRIC APPROACH. <https://www.cs.umd.edu/users/mvz/handouts/gqm.pdf>