# 

# 

# 

# 

# 

# **Assignment 1 FINAL REPORT**

# **Title: Analyzing the Effect of Size on Maintainability in Java Projects**

# 

# 

# **Submitted by**

# **Siva Parimisetty**

# **Section 1: Introduction**

In this group assignment, we are focusing on the concept of software maintainability, which refers to how easily a software system can be upgraded, fixed, adjusted, or extended. It is a very important part in software engineering, and also understanding how maintainability is effected as software projects grow longer is crucial.

To investigate this relationship, this group project focuses on java project and java widely used programming language in various application domains such as online applications, mobile apps, and corporate systems. By studying the connection between the size and java projects and their maintainability, the findings can be applicable to a wide range of projects and development teams.

The Goal-Question-Metric framework is employed to organise the analysis. This framework allows the structure for research goals, generate relevant questions, and select appropriate metrics to provide answers. By using the GQM technique, the research is conducted in a systematic and structured manner, ensuring that the conclusions drawn are both relevant and useful.

The main objective of the research is to extreme how the size of java projects impacts their maintainability. By understanding the relationship, development teams can make informed decisions during the software development process, leading to the creation of the maintainable code. One of the key questions addressed is whether the number of lines of code(LoC) in java projects significantly affects its maintainability. These questions focused on uncovering trends and patterns that can guide the development of best practices for managing and sustaining large java projects.

To assess maintainability, there are several metrics utilised in the study. These metrics include lines of code (LoC), coupling between objects (CBO), response for a class (RFC), and Tight class cohesion (TCC).

Coupling Between Objects (CBO) measures the degree of interdependence between classes.

Response for a Class (RFC) counts the number of methods that can be executed in response to messages received by a class object.

Tight Class Cohesion (TCC) evaluates the level of interrelatedness between class methods.

**Section 2: Data set Description**

To conduct our analysis, we needed a representative sample of Java projects that met specific criteria. The selection criteria were designed to ensure that the chosen projects had a sufficient size, age, and developer involvement to provide meaningful insights into the relationship between size and maintainability. The criteria are as follows:

1. The project must be at least 10K lines of code (LoC) in size. This criterion ensures that the projects included in the analysis are large enough to exhibit potential maintainability challenges associated with size.

2. The project must be at least 3 years old. By selecting projects with a minimum age of 3 years, we can ensure that they have gone through various maintainability tasks, such as bug fixes, refactoring, and feature additions. This age criterion helps us to study the effect of size on maintainability in projects with a more extended development history.

3. The project must have at least 3 developers. This criterion ensures that the projects included in the analysis have experienced collaborative development, which can introduce additional maintainability challenges.

Based on these criteria, we selected five Java projects from GitHub. We also considered the list of 500 Java projects provided with the assignment, ensuring that the selected projects met our criteria. The selected projects are as follows:

| **Project Name** | **Repository URL** | **Description** | **Size (LoC)** | **Developers** | **Age (Years)** |
| --- | --- | --- | --- | --- | --- |
| bumptech/glide | https://github.com/bumptech/glide | An image loading and caching library for Android focused on smooth scrolling | 91776 | 121 | 9 |
| google/guava | https://github.com/google/guava | Google core libraries for Java | 388778 | 273 | 8 |
| apache/rocketmq | https://github.com/apache/rocketmq | Mirror of Apache RocketMQ | 14799 | 357 | 6 |
| OpenAPITools/openapi-generator | https://github.com/OpenAPITools/openapi-generator | OpenAPI Generator allows generation of API client libraries (SDK generation), server stubs, documentation and configuration automatically given an OpenAPI Spec (v2, v3) | 677598 | 402 | 5 |
| neo4j/neo4j | https://github.com/neo4j/neo4j | Graphs for Everyone | 544557 | 221 | 11 |

**Section 3: Tool Description**

To gather measurements for the C&K metrics of the classes in our selected projects, we utilised the CKJM tool. The choice of CKJM is based on several reasons,

CKJM is specifically designed to work with java-based projects and compute chidamber and kemerer’s metrics for object-oriented programming. These all metrics including CBO, RFC, and TCC, are directly relevant to our research inquiry and provide insights into the relationship between size and maintainability in java projects.

CKJM is a command-line utility that offers a user-friendly interface. It can be easily installed and operated without requiring extensive technical knowledge. By accepting java bytecode files in the class format, it efficiently generates the desired metrics in a clear concise format.

CKJM is an open-source tool that comes with a comprehensive user guide. This guide helps users understand the functionality of the tool and the metrics it provides. The transparency in the metrics acquisition process ensures their reliability and trustworthiness for analytical purposes.

CKJM helps users to select the specific C&K metrics they want to compute, giving them the flexibility to choose the relevant metrics for their analysis. In this project, we focused on the CBO, RFC, and TCC metrics which is essential for understanding the maintainability of the java projects.

CKJM has a solid reputation and is frequently referenced in academic investigations within the software engineering research field. Its wisepared use and recognition provide further assurance of the reliability of the metrics it generates.

The CKJM tool can be accessed through the following URL,

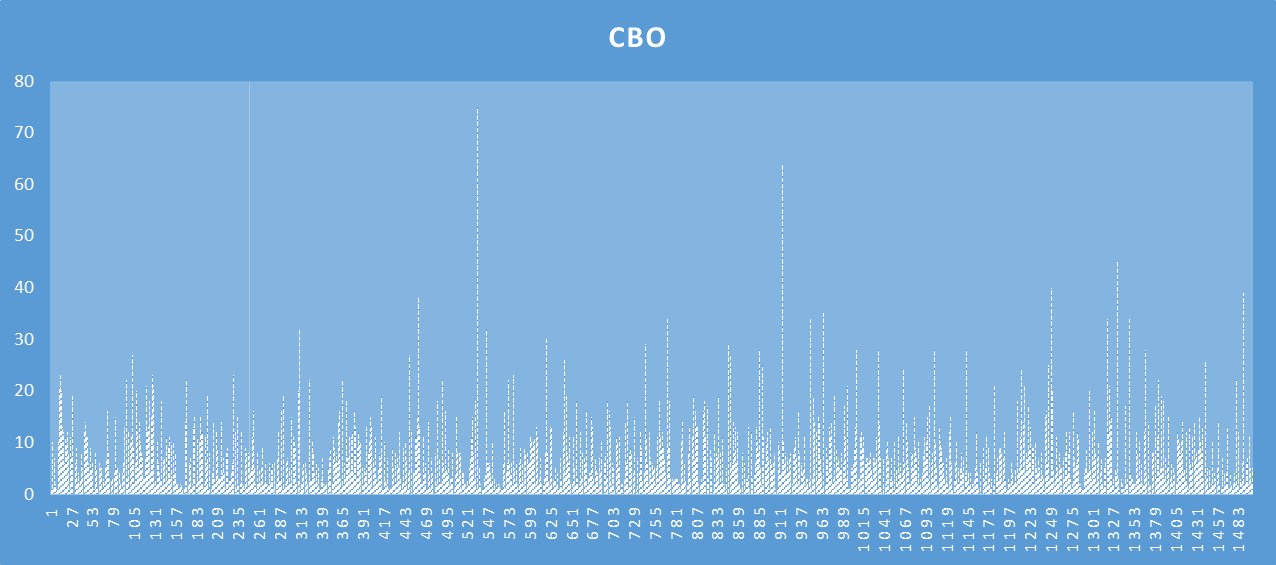
https://github.com/mauricioaniche/ck

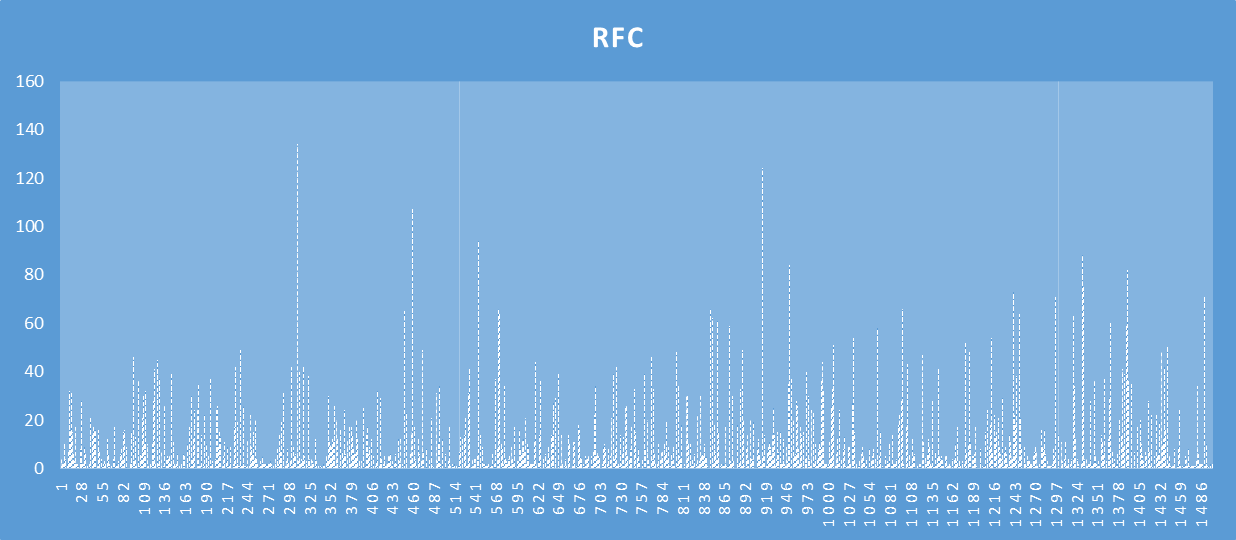
In our analysis, the initial step involved retrieving the source code of each project from the GitHub platform. After that, the projects were compiled to generate java bytecode files. Using the CKJM tool, we obtained the C&K metrics measurements for each class based on the compiled projects. These metrics were then used to extreme the relationship between size and maintainability in the selected java projects, as discussed in the section 4 of our research.

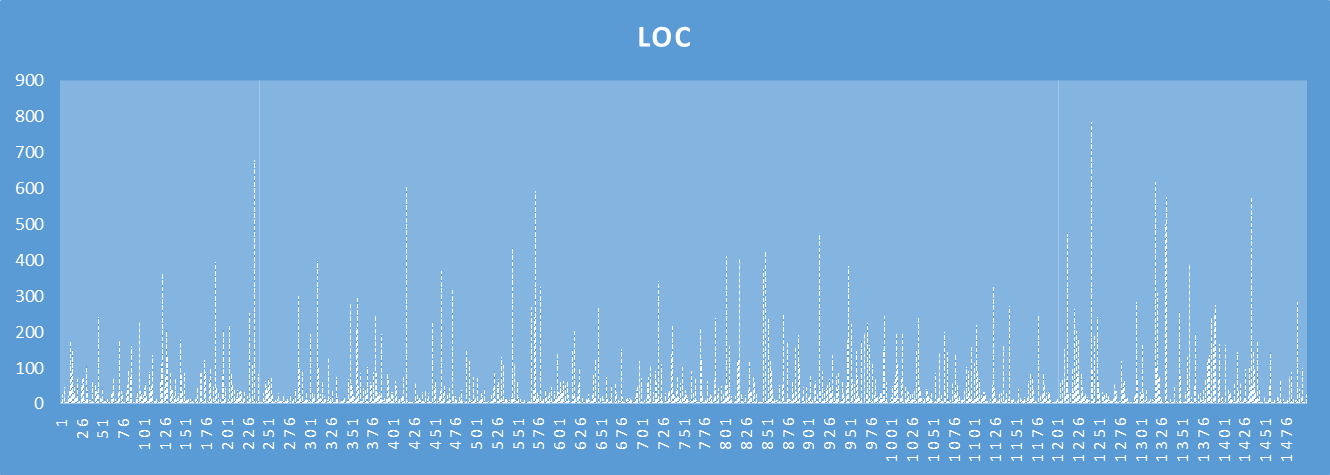
**Section 4: Projects**

**Project1 : glide**

|  | **CBO** | **RFC** | **TCC** | **LOC** |
| --- | --- | --- | --- | --- |
| **Max** | 75 | 134 | 1 | 785 |
| **Mode** | 2 | 0 | 0 | 5 |
| **Median** | 4 | 2 | 0.2 | 13 |
| **Std Deviation** | 6.483364 | 13.22057 | 0.536141 | 75.40273 |
| **Average** | 6.003333 | 7.316667 | 0.302462 | 39.88733 |

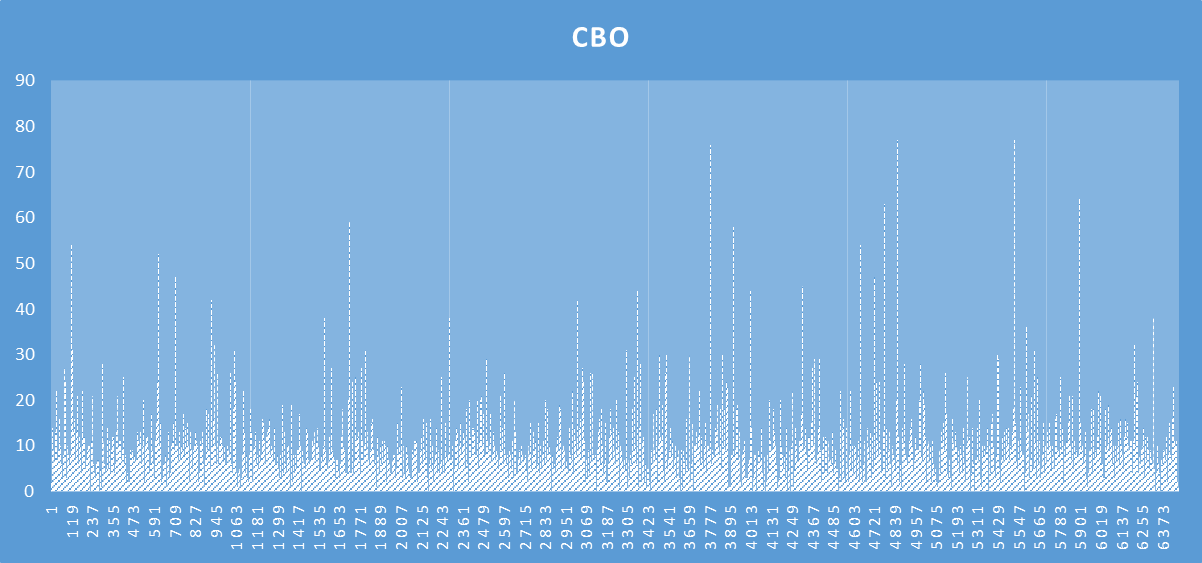
****

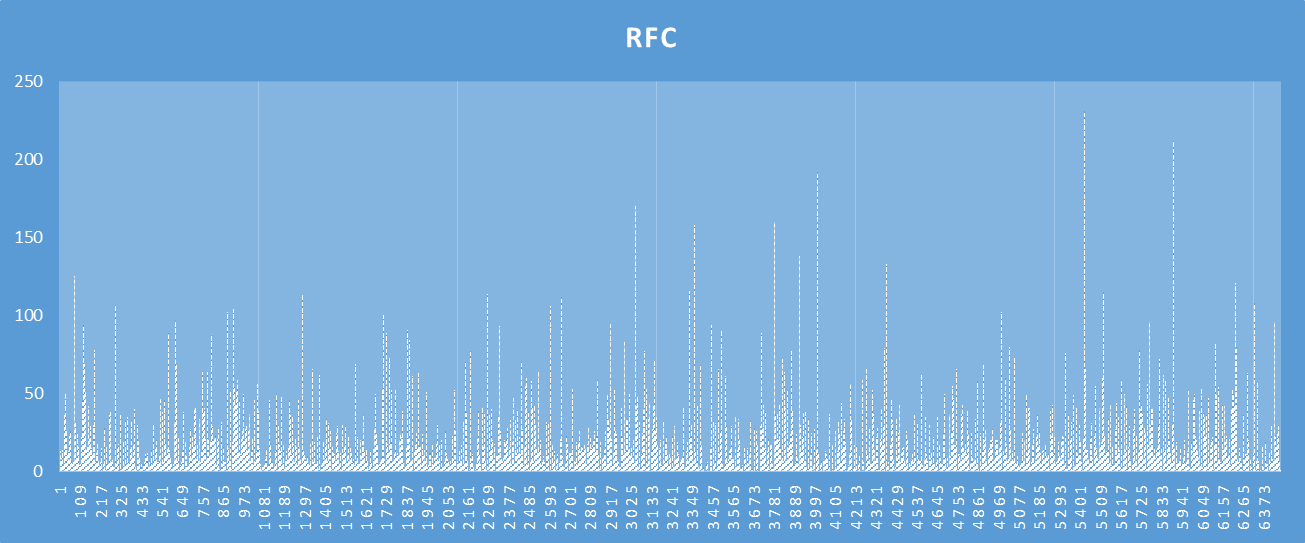
****

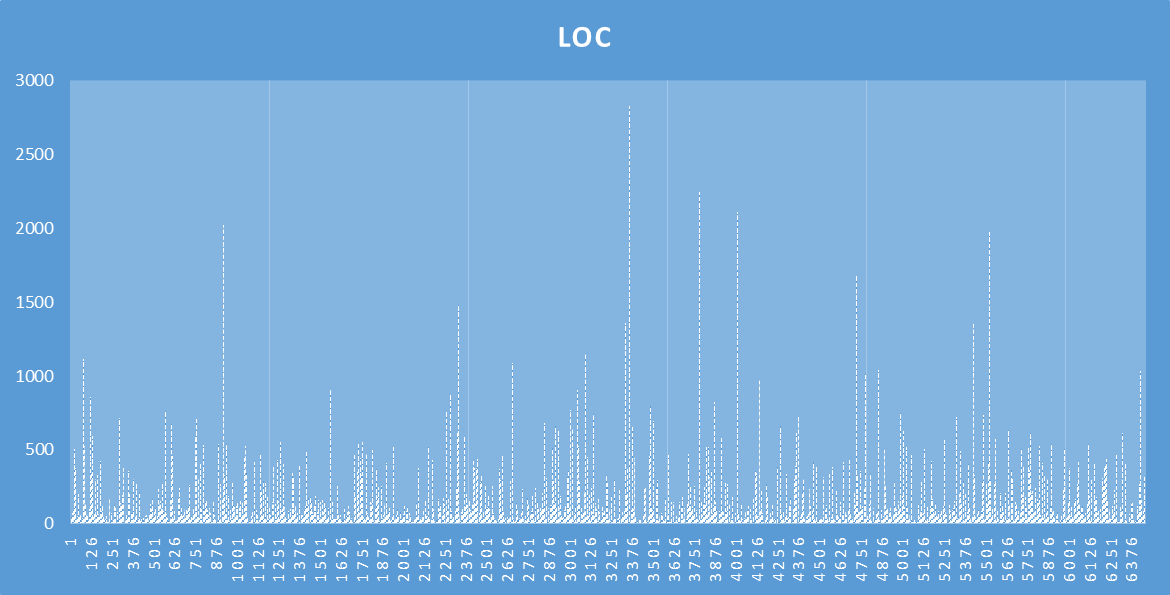
****

**Project2 : guava**

|  | **CBO** | **RFC** | **TCC** | **LOC** |
| --- | --- | --- | --- | --- |
| **Max** | 77 | 231 | 1 | 2828 |
| **Mode** | 0 | 0 | 0 | 5 |
| **Median** | 2 | 2 | 0 | 9 |
| **Std Deviation** | 5.318617 | 13.65527 | 0.650696 | 117.855 |
| **Average** | 3.767064 | 6.370221 | -0.07887 | 40.68782 |

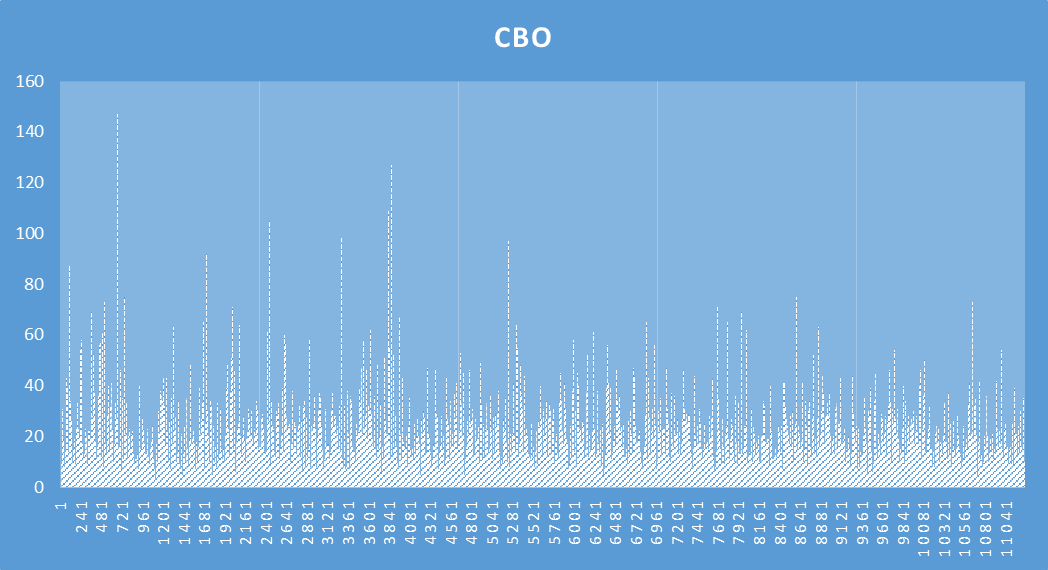
****

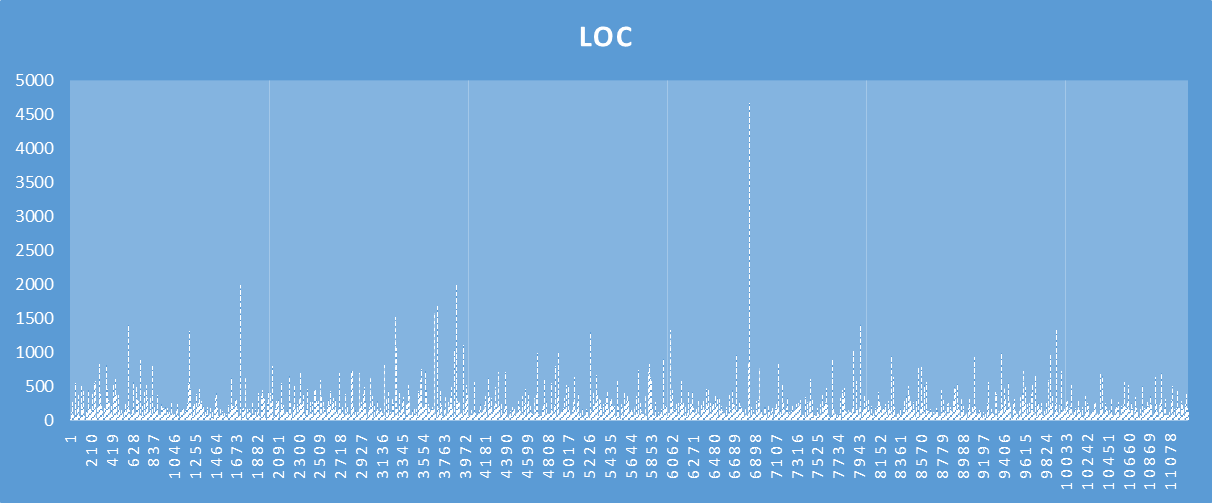
****

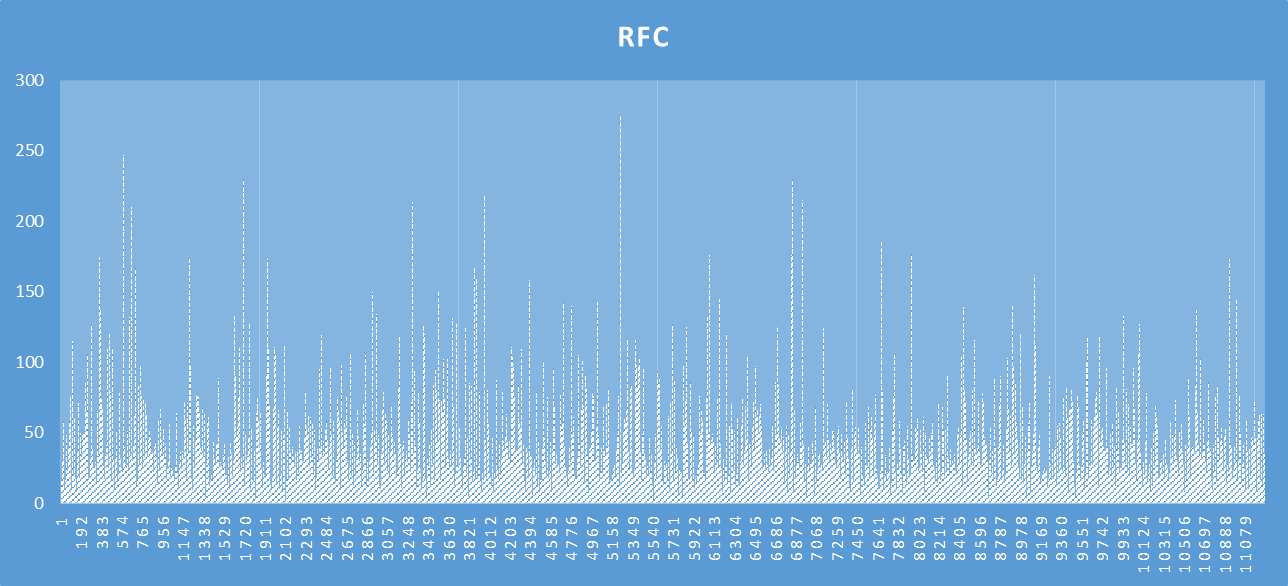
****

**Project3: neo4j**

|  | **CBO** | **RFC** | **TCC** | **LOC** |
| --- | --- | --- | --- | --- |
| **Max** | 147 | 275 | 1 | 4670 |
| **Mode** | 2 | 0 | 0 | 5 |
| **Median** | 4 | 3 | 0 | 19 |
| **Std Deviation** | 8.30141 | 19.00052 | 0.496931 | 108.6678 |
| **Average** | 6.452292 | 10.5677 | 0.25016 | 50.03962 |

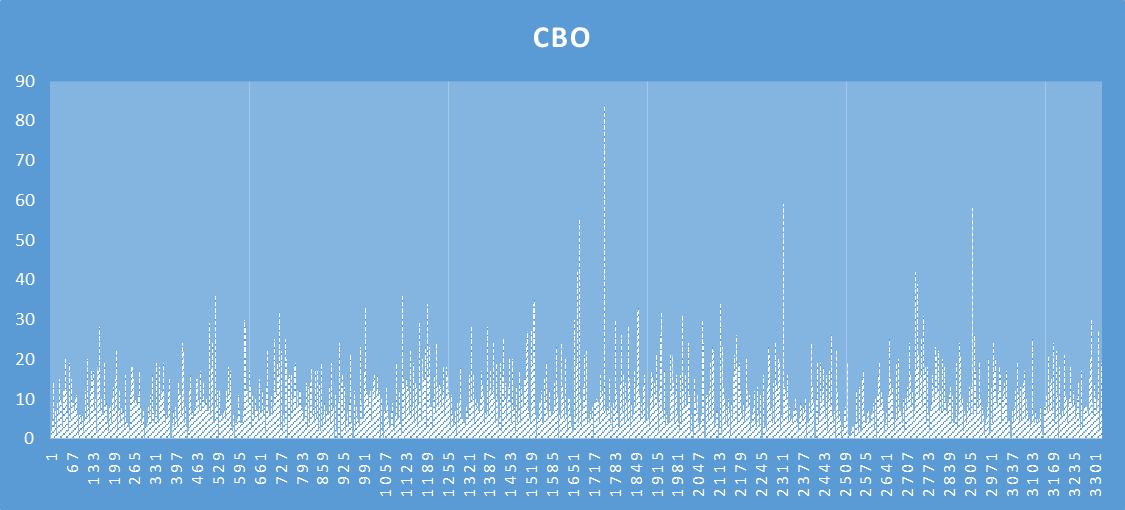
****

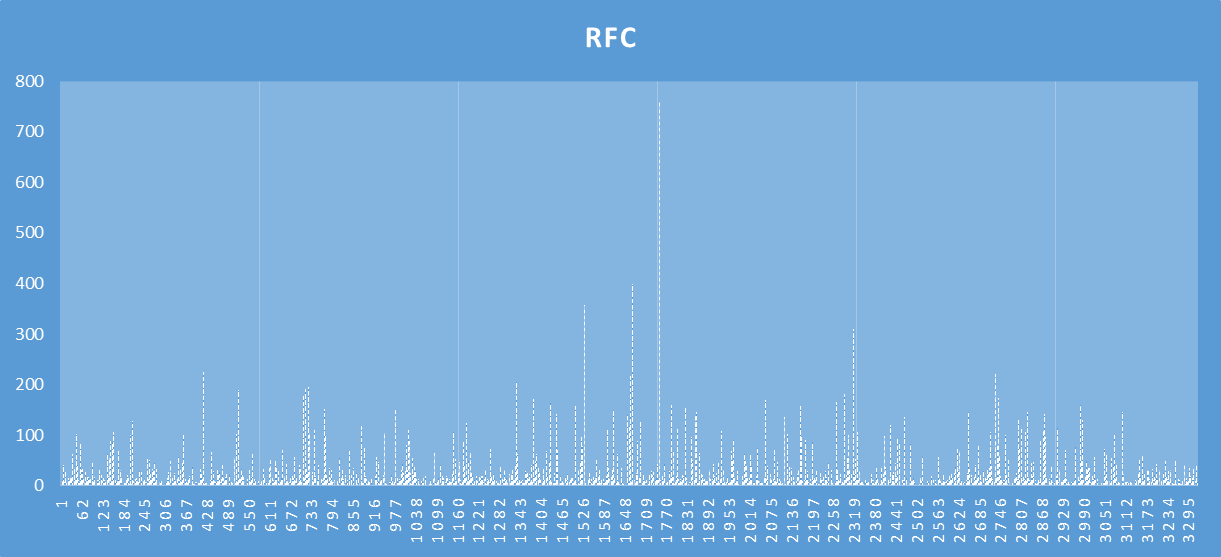
****

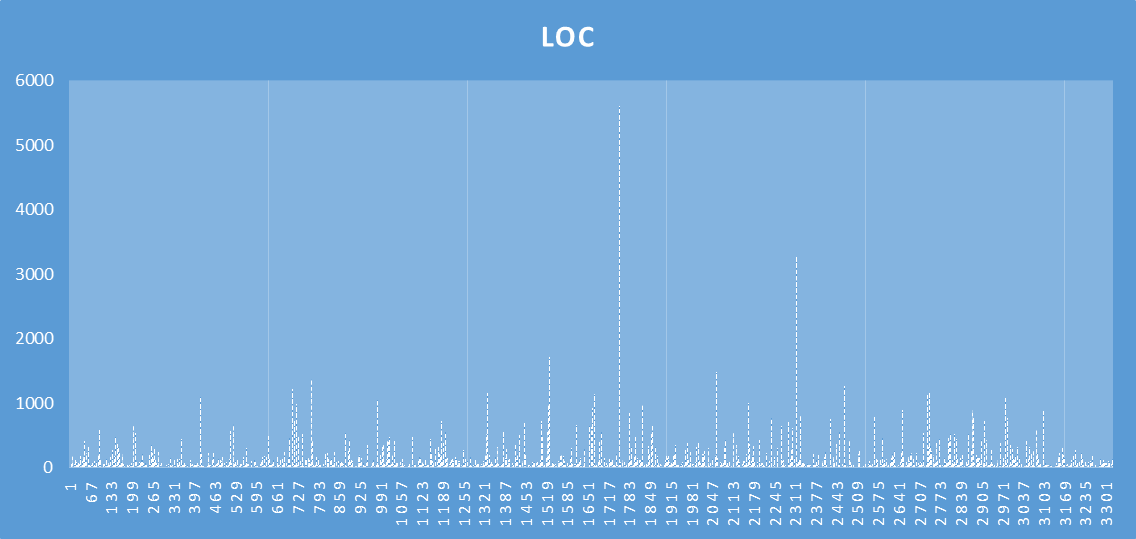
****

**project 4: openapi-generator**

|  | **CBO** | **RFC** | **TCC** | **LOC** |
| --- | --- | --- | --- | --- |
| **Max** | 84 | 758 | 1 | 5603 |
| **Mode** | 0 | 0 | 0 | 2 |
| **Median** | 3 | 3 | 0 | 19 |
| **Std Deviation** | 5.99679 | 28.17365 | 0.577237 | 169.0862 |
| **Average** | 5.162651 | 10.87108 | 0.019737 | 58.59157 |

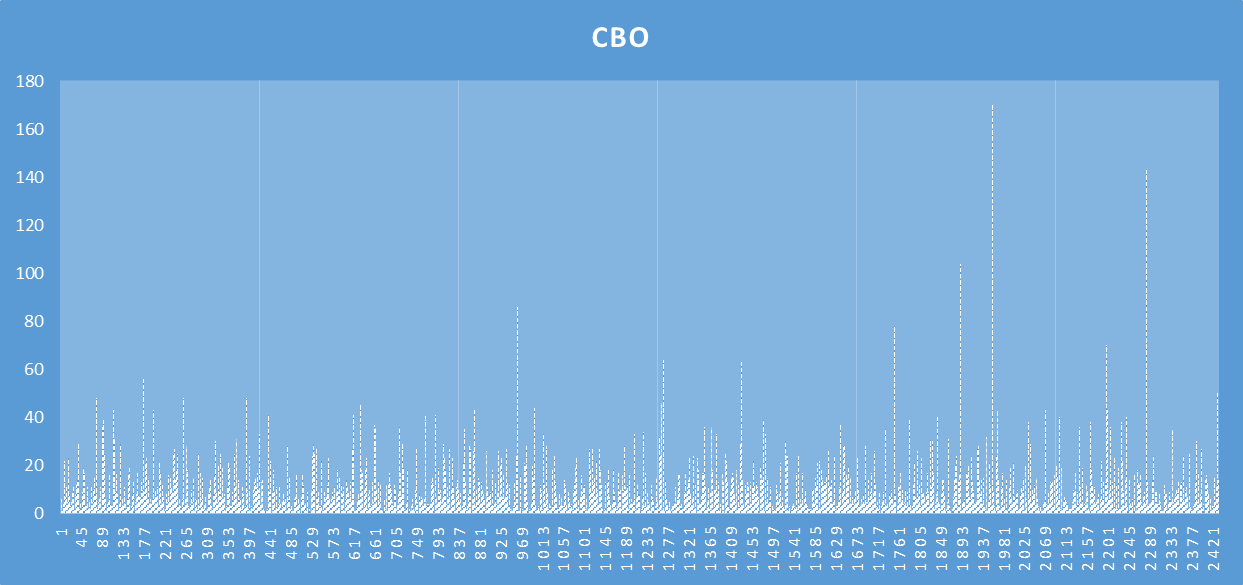
****

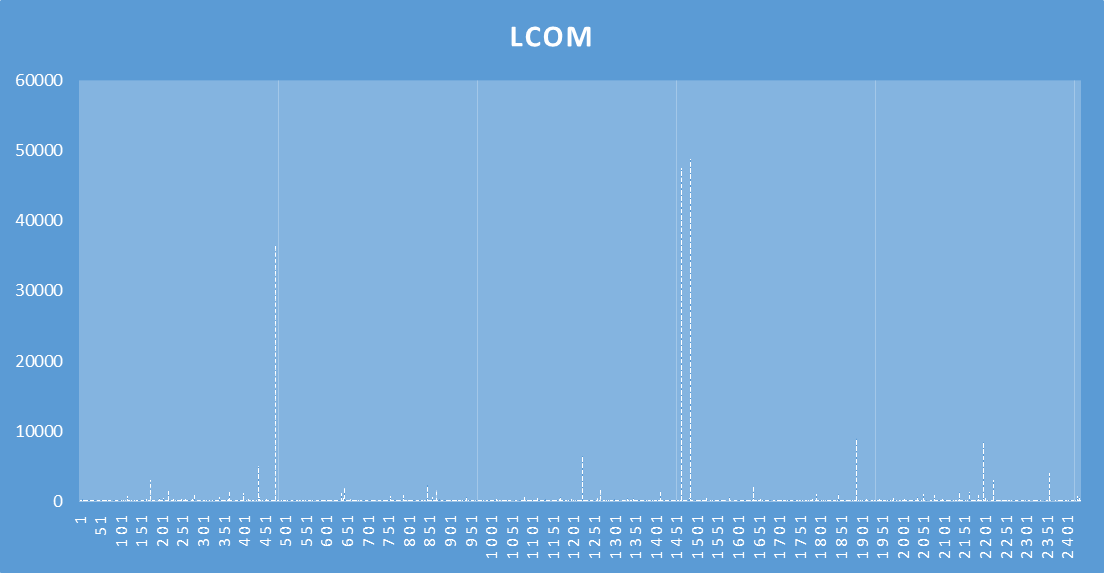
****

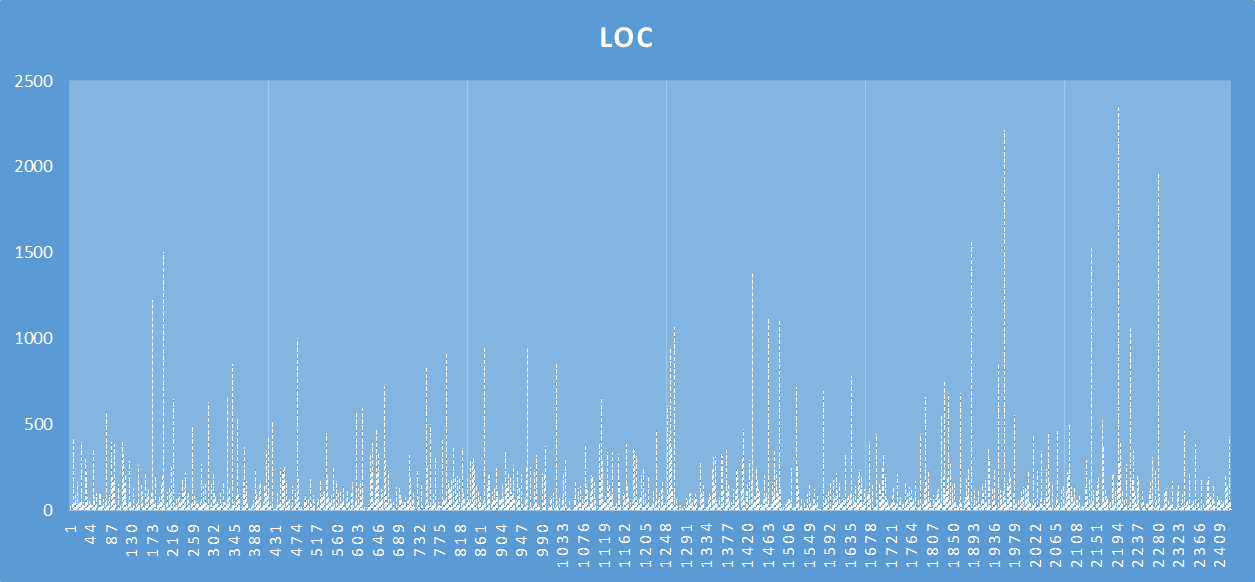
****

**Project5: rocketmq**

|  | **CBO** | **RFC** | **TCC** | **LOC** |
| --- | --- | --- | --- | --- |
| **Max** | 170 | 552 | 1 | 2354 |
| **Mode** | 1 | 0 | 0 | 5 |
| **Median** | 3 | 6 | 0.2 | 28 |
| **Std Deviation** | 9.640308 | 32.60403 | 0.460953 | 151.0807 |
| **Average** | 6.526316 | 17.80633 | 0.272992 | 72.63898 |

****

****

****

**Section 5: Conclusion**

Based on our analysis of C&K metrics measurements from a set of java projects, we have reached the relationships between the size and maintainability in java projects.

Larger classes, measured by lines of code(LoC), tend to exhibit higher coupling (CBO) and increased response for a class (RFC). This suggests a positive correlation between class size and interconnectivity with other classes, as well as the number of methods within a class. Higher coupling and RFC levels can negatively impact maintainability, as modifications to one class may require changes in multiple other classes, and managing a larger number of methods can lead to increased complexity.

The relationship between class size and tight class cohesion (TCC) is not consistently observed. In some cases, there is positive collaboration, while in others, a negative correlation is observed. The impact of maintainability varies across different projects. Some projects. Some projects demonstrate proficiency in handling maintainability challenges associated with size, while others struggle to maintain code quality as the project size increases. Developers and organisations should be cautious about the size of their java projects and consider the potential effect on maintainability. Based on our analysis, it can be inferred that maintaining larger projects, especially those with high coupling and high RFC, can be more challenging. Our analysis suggests a correlation between size and maintainability in java projects. Larger projects tend to exhibit higher coupling RFC. The relationship between size and cohesion is not straightforward, and the impact of size on maintainability varies across projects. Developers and organisations should consider three findings to inform their approach to java projects, leading to the adoption of improved development methodologies and long-term project success. Future studies could explore the impact of development practices, code organisation, and application complexity on maintainability. Additionally, identifying optimal strategies for overcoming size related obstacles in java projects would be beneficial.

**References:**

1. Software Metrics: A Rigorous and Practical Approach, Third Edition. (n.d.). Routledge & CRC Press. Retrieved April 10, 2023, from<https://www.routledge.com/Software-Metrics-A-Rigorous-and-Practical-Approach-Third-Edition/Fenton-Bieman/p/book/9780367659028>

2. Martin, R. (n.d.). Agile Software Development, Principles, Patterns, and Practices.<https://dl.ebooksworld.ir/motoman/Pearson.Agile.Software.Development.Principles.Patterns.and.Practices.www.EBooksWorld.ir.pdf>

3. Design Patterns: Elements of Reusable Object-Oriented Software. (2019). O’Reilly | Safari.<https://learning.oreilly.com/library/view/design-patterns-elements/0201633612/>

4. Object-Oriented Metrics in Practice. (2006). Springer Berlin Heidelberg.<https://doi.org/10.1007/3-540-39538-5>