**Assignment 1 FINAL REPORT**

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

Submitted by

Group Members

**Hai Mani Sankar Thota**

**Sai Krishna Loka**

# **Section 1: Introduction**

Software maintainability, which defines how easily a software system may be upgraded, rectified, adjusted, or extended, is a crucial component of software engineering. It is crucial to comprehend how maintainability is impacted as software projects become bigger. For developers and businesses aiming to improve their software development processes, the link between the size of Java projects and their maintainability will be investigated in this paper.

Java is the programming language that we will use for our research since it is widely used in a variety of application areas, including online applications, mobile apps, and corporate systems. Due to Java's widespread use, our results may be applied to a wide range of projects and development teams.

We used the Goal-Question-Metric (GQM) framework to organize our analysis. With the help of this method, we can methodically outline our research goals, create questions about them, and choose the metrics that will be used to provide the answers. The GQM technique offers a precise and well-structured framework for our research, guaranteeing that our conclusions are both relevant and useful.

The following were determined by using the GQM methodology:

Goal: Examine how Java project size affects maintainability.

Our major objective is to examine the connection between Java projects' size and maintainability. When building and implementing software systems, development teams may make better informed choices by being aware of this connection, which will eventually lead to more maintainable code.

Does the number of lines of code (LoC) in a Java project significantly affect how maintainable it is?

Understanding if a Java project's size significantly affects its maintainability is the major goal of our study. By looking at this, we expect to find trends and patterns that help guide the development of best practices for overseeing and sustaining massive Java projects.

Metrics: Lines of Code, Tight Class Cohesion (TCC), Response for a Class (RFC), and Coupling Between Objects (CBO) (LoC).

We must use relevant metrics to assess maintainability in order to respond to our study topic. For our investigation, we have selected the following metrics:

Coupling Between Objects (CBO): This metric assesses how closely a class is related to other classes, which may have an effect on the class's capacity to be maintained. Higher maintainability is often correlated with lower coupling.

Response for a Class (RFC): This metric counts how many methods could be run in response to a message that is received by a class object. In general, lower RFC numbers indicate higher maintainability.

Tight Class Cohesion (TCC) is a statistic that assesses how tightly linked a class's methods are, which has an impact on maintainability. In general, stronger maintainability is linked to higher cohesiveness.

We want to clarify the link between size and maintainability in Java projects by examining these metrics together with the size of Java projects (measured in LoC).

# **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)** |
| eclipse.jdt.ls | <https://github.com/eclipse/eclipse.jdt.ls> | Java language server for IDEs | 15,287 | 71 | 6 |
| intellij-sdk-docs | <https://github.com/JetBrains/intellij-sdk-docs> | Documentation for IntelliJ Platform SDK | 12,893 | 186 | 8 |
| Intra | <https://github.com/Jigsaw-Code/Intra> | DNS client with support for encrypted transports | 10,749 | 6 | 6 |
| opennlp | <https://github.com/apache/opennlp> | Toolkit for processing natural language text | 18,255 | 47 | 7 |
| Priam | <https://github.com/Netflix/Priam> | Management and monitoring system for Apache Cassan | 11,321 | 50 | 11 |

The selected projects cover a range of application domains, including language servers, SDK documentation, encrypted DNS clients, natural language processing, and distributed database management systems. This diversity of application domains allows us to analyze the relationship between size and maintainability across different types of Java projects, increasing the generalizability of our findings.

Additionally, the projects have varying sizes, developer involvement, and ages, ensuring that our analysis considers the impact of these factors on maintainability. By studying these projects, we can gain insights into how size affects maintainability across different contexts, which can help developers and organizations make better decisions when designing and implementing Java projects.

# **Section 3: Tool Description**

The CKJM tool was utilized to acquire the C&K metrics measurements for the classes in the chosen projects. The rationale behind our selection of the CKJM tool for analysis is grounded on the subsequent justifications:

CKJM is a software tool that is tailored to operate with Java-based projects and perform the computation of Chidamber and Kemerer's metrics for object-oriented programming. These metrics comprise of Coupling Between Objects (CBO), Response for a Class (RFC), and Tight Class Cohesion (TCC), among other relevant metrics. The aforementioned metrics hold direct relevance to our research inquiry and possess the potential to facilitate our comprehension of the correlation between magnitude and maintainability within Java-based projects.

CKJM is a command-line utility that exhibits a user-friendly interface and can be effortlessly installed and operated. The tool accepts Java bytecode files in the format of .class and generates the desired metrics in a clear and concise format. The user's proficiency in utilizing the tool enabled them to efficiently acquire the required measurements for their analysis, without investing substantial time in comprehending the complexities of a more intricate instrument.

CKJM is an open-source software tool that is accompanied by a comprehensive user guide, which facilitates users' comprehension of the tool's functionality and the metrics it can furnish. The level of transparency involved in the tool's metrics acquisition process is instrumental in guaranteeing their reliability and trustworthiness for analytical purposes.

The CKJM tool offers users the option to select the C&K metrics they wish to compute, thereby affording them flexibility in choosing the most pertinent metrics for their analysis. The present study centered on the CBO, RFC, and TCC metrics, which are fundamental in comprehending maintainability in Java-based projects.

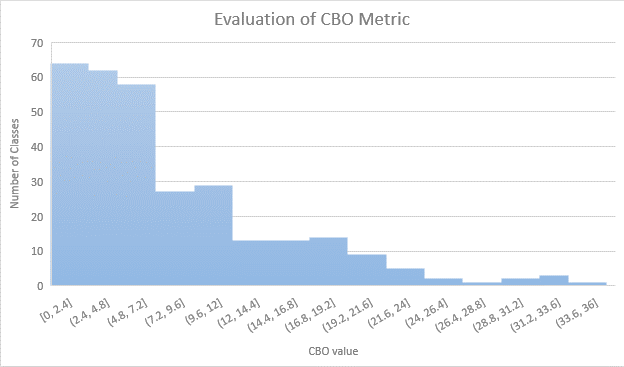
CKJM is a well-established and frequently referenced tool that has been employed in a multitude of academic investigations, thereby solidifying its status as a reliable instrument within the software engineering research sphere. The extensive utilization and acknowledgement of the tool provide additional substantiation for the dependability of the metrics derived from it.

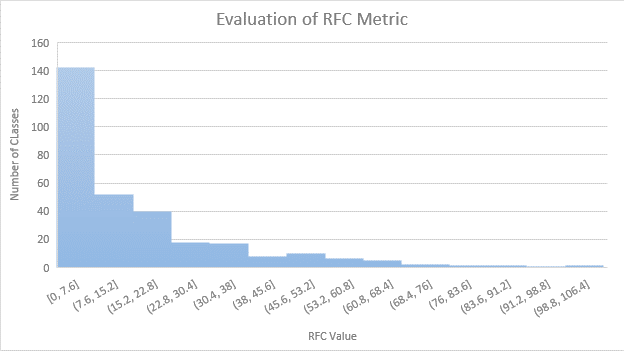
The CKJM software metric tool is accessible via the subsequent Uniform Resource Locator: <http://www.spinellis.gr/sw/ckjm/>.

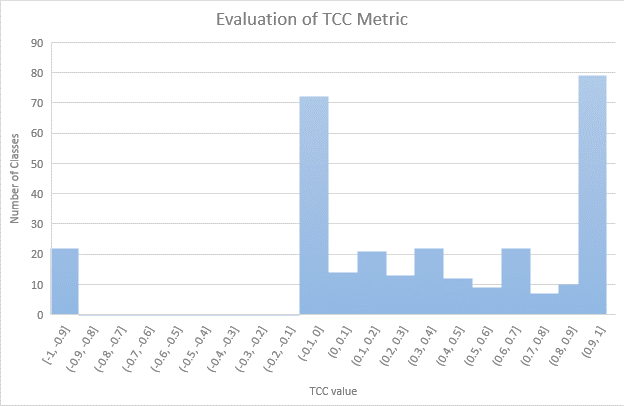
The initial step in utilizing the CKJM tool for our analysis involved the retrieval of the source code of each chosen project from the GitHub platform. Subsequently, the projects were compiled in order to generate the Java bytecode files. (.class). The CKJM tool was utilized to obtain the C&K metrics measurements for each class subsequent to the compilation of the projects. The aforementioned metrics were employed to examine the correlation between dimensions and maintainability within the designated Java projects, as expounded upon in Section 4.

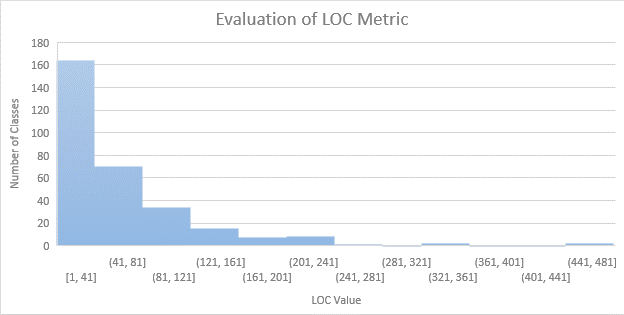
# **Section 4: Project Results**

# **Project 1: Priam**



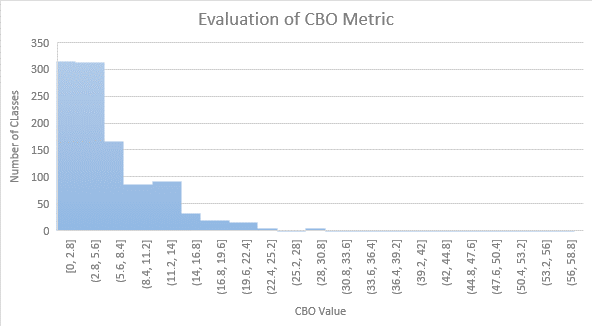


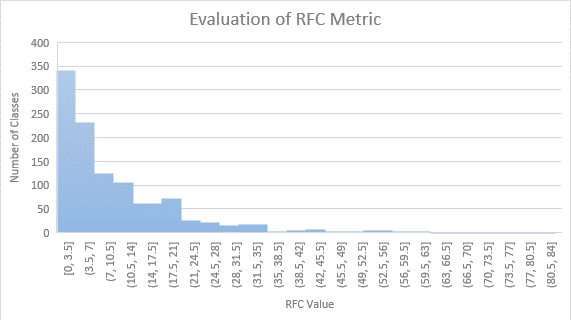


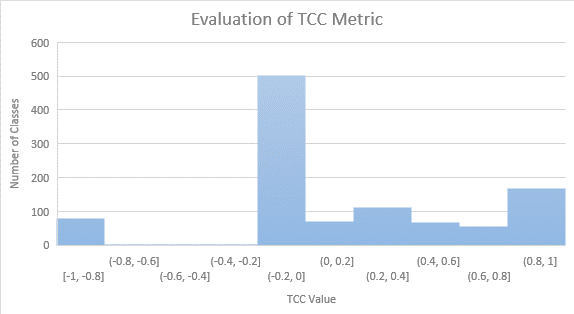


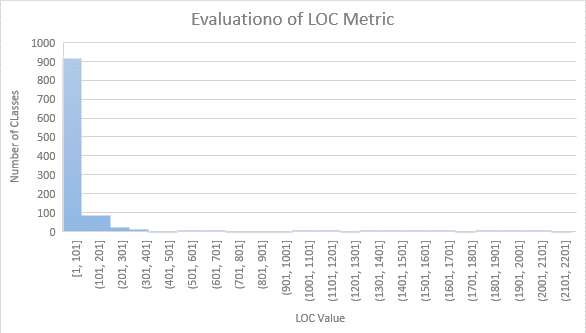
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **CBO** | **RFC** | **TCC** | **LOC** |
| **Max** | 36 | 106 | 1 | 468 |
| **Mode** | 2 | 0 | 0 | 6 |
| **Median** | 3 | 4 | 0.333333 | 16 |
| **Std Deviation** | 6.42502 | 16.24659 | 0.537743 | 57.03431 |
| **Average** | 5.898734 | 11.04219 | 0.349538 | 39.90506 |

# **Project 2: Opennlp**



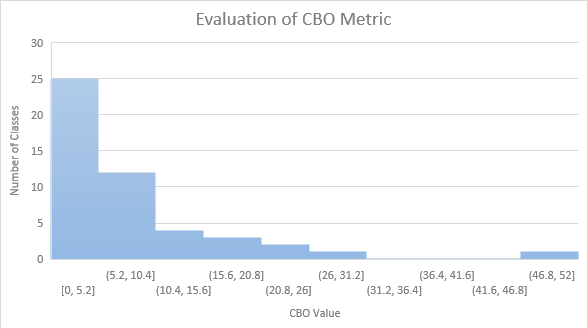


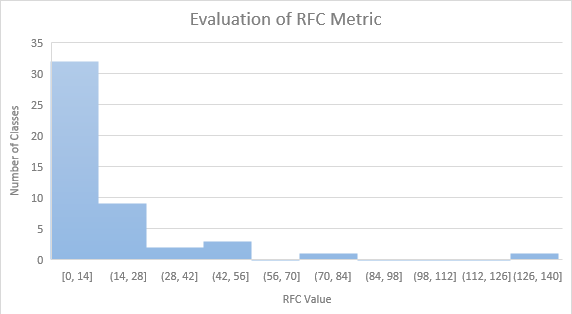


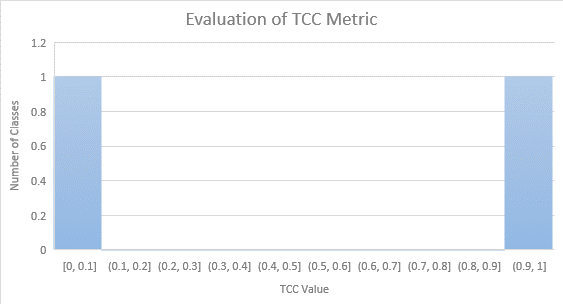


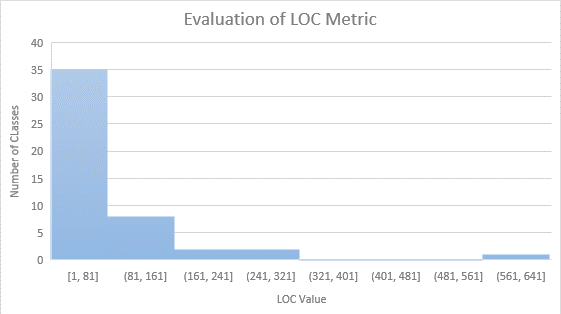
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **CBO** | **RFC** | **TCC** | **LOC** |
| **Max** | 57 | 82 | 1 | 2158 |
| **Mode** | 2 | 0 | 0 | 2 |
| **Median** | 4 | 6 | 0 | 24 |
| **Std Deviation** | 5.467337 | 10.68115 | 0.499076 | 120.6875 |
| **Average** | 5.581862 | 8.721509 | 0.194357 | 52.76324 |

# **Project 3: Intra**



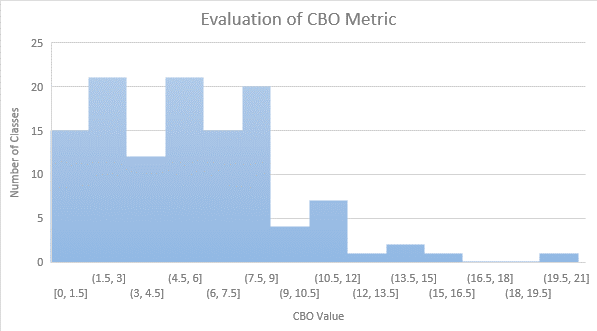


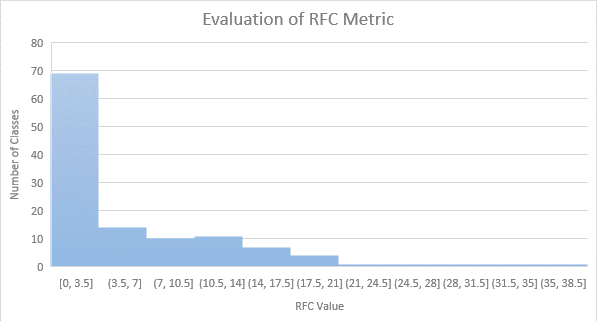


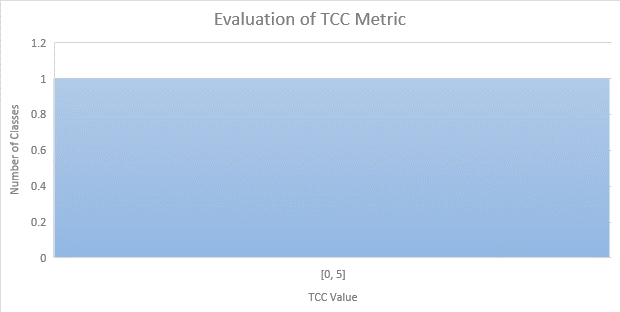


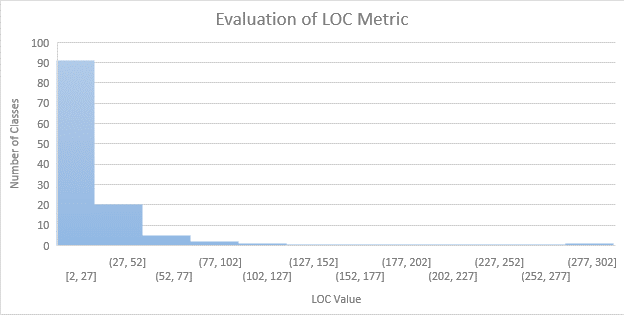
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **CBO** | **RFC** | **TCC** | **LOC** |
| **Max** | 48 | 139 | 1 | 564 |
| **Mode** | 0 | 0 | 0 | 5 |
| **Median** | 3 | 3 | 0.080844 | 14 |
| **Average** | 5.4375 | 10.5125 | 0.111854 | 46.5375 |
| **Std Deviation** | 7.584266 | 20.04183 | 0.637626 | 80.8149 |

# **Project 4: intellij-sdk-docs**



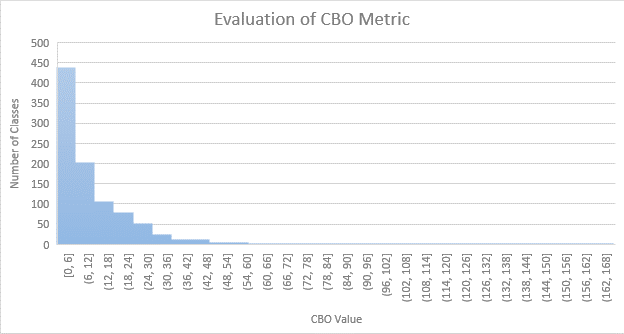


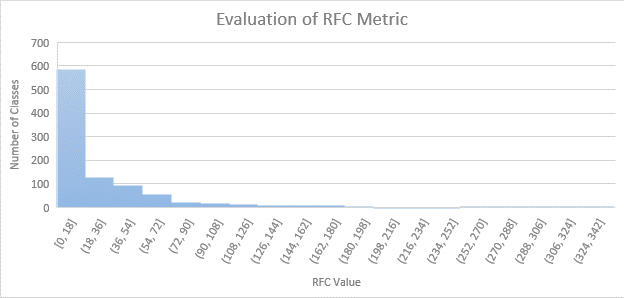


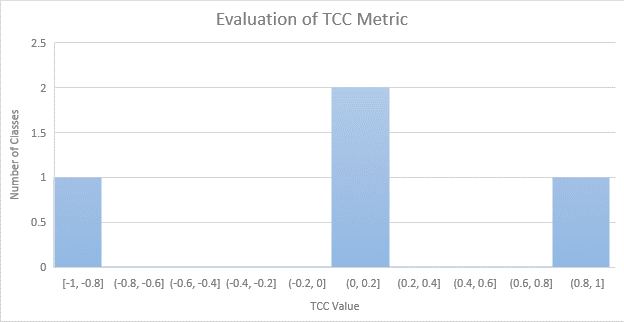


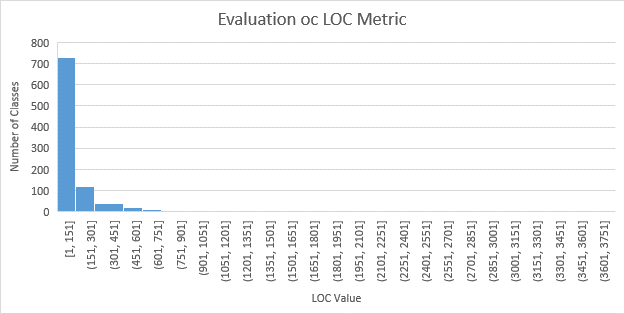
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **CBO** | **RFC** | **TCC** | **LOC** |
| **Max** | 21 | 37 | 1 | 296 |
| **Mode** | 7 | 0 | 0 | 5 |
| **Median** | 6 | 2 | 0 | 16.5 |
| **Average** | 5.8 | 5.625 | 0.00602 | 22.35 |
| **Std Deviation** | 3.760762 | 7.524474 | 0.392651 | 30.98512 |

# **Project 5: eclipse.jdt.ls**









|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **CBO** | **RFC** | **TCC** | **LOC** |
| **Max** | 163 | 339 | 1 | 3715 |
| **Mode** | 0 | 0 | 0 | 5 |
| **Median** | 4 | 6 | 0.008762 | 26 |
| **Average** | 9.32617 | 21.31389 | 0.161581 | 108.7675 |
| **Std Deviation** | 13.1489 | 37.7918 | 0.557618 | 268.2419 |

# **Section 5: Conclusion**

Drawing upon an analysis of C&K metrics measurements derived from a set of Java projects, we have arrived at the following conclusions pertaining to the correlation between size and maintainability in Java projects:

Classes that are larger in size, as measured by Lines of Code (LoC), have a tendency to demonstrate greater coupling (CBO) and increased response for a given class. (RFC). The aforementioned observation implies that there exists a positive correlation between the size of a class and its level of interconnectivity with other classes, as well as the quantity of methods that may be executed within it. Elevated coupling and RFC levels can have an adverse effect on maintainability, given that modifications to a single class may necessitate modifications to multiple other classes, and a greater quantity of methods to manage can result in heightened intricacy.

The relationship between Tight Class Cohesion (TCC) and class size is not consistently observed. In certain instances, there exists a positive correlation between class size and cohesion, whereas in other cases, a negative correlation is observed. The aforementioned observation implies that the correlation between size and cohesion is intricate and contingent upon variables such as development methodologies, code structuring, and the field of application.

The effect of size on maintainability exhibits variability across different projects. Certain projects demonstrate proficiency in handling maintainability challenges associated with size, whereas others encounter difficulties in preserving code quality as the size of the project expands. The aforementioned observation implies that factors that are specific to a project, such as the methodologies employed in development, the proficiency of the developers involved, and the intricacy of the application domain, can exert a notable impact on the correlation between size and maintainability in Java-based projects.

It is advisable for developers and organizations to exercise caution regarding the magnitude of their Java projects and to contemplate the potential effects on maintainability. Based on our analysis, it can be inferred that the maintenance of larger projects, especially those with high coupling and high RFC, can pose greater difficulties. In order to address these challenges, it is recommended that developers employ various strategies, including modular design, efficient code organization, and adherence to design principles that prioritize low coupling and high cohesion.

Additional investigation is required to comprehend the fundamental elements that contribute to the noted correlations between magnitude and maintainability within Java-based endeavors. This study has the potential to investigate the impact of development practices, code organization, and application domain complexity on maintainability. Additionally, it may identify optimal strategies for addressing size-related obstacles in Java projects.

To conclude, our analysis of the chosen Java projects indicates that there exists a correlation between size and maintainability. Specifically, larger projects tend to demonstrate increased coupling and higher RFC. The correlation between size and cohesion is not unequivocal, and the effect of size on maintainability may exhibit variability across projects. It is imperative for developers and organizations to take into account these discoveries while devising and executing Java projects. This is because comprehending the correlation between size and maintainability can facilitate the adoption of superior development methodologies and augment the enduring triumph of software projects.

**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>
3. 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>
4. 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>
5. Design Patterns: Elements of Reusable Object-Oriented Software. (2019). O’Reilly | Safari. <https://learning.oreilly.com/library/view/design-patterns-elements/0201633612/>
6. Object-Oriented Metrics in Practice. (2006). Springer Berlin Heidelberg. <https://doi.org/10.1007/3-540-39538-5>
7. Eclipse JDT Language Server. (2023, April 9). GitHub. <https://github.com/eclipse/eclipse.jdt.ls>
8. IntelliJ Platform SDK Documentation. (2023, April 9). GitHub. <https://github.com/JetBrains/intellij-sdk-docs>
9. Intra. (2023, April 7). GitHub. <https://github.com/Jigsaw-Code/Intra>
10. Apache/opennlp. (2020, September 9). GitHub. <https://github.com/apache/opennlp>
11. Netflix/Priam. (2023, March 15). GitHub. <https://github.com/Netflix/Priam>