SP24-CPSC-60000-003

# Object Oriented Development

**Group Assignment 1**

**Empirical Study on the Effect of Class Size on Software Maintainability**

**NAME: Sushrith Manchala**

**INSTRUCTOR: Fedi Wedyan**

**SECTION 1- INTRODUCTION**

Software maintainability is the critical, but not the least, part of software development, which mostly determines the total cost of production during the life cycle of this product. This feature mainly refers to how a software system can be updated quickly to correct bugs, support new features or respond to the changing usage environment. Though the maintained class size is supposedly related to maintainability, evincing with this theory is quite scant.

The hypothesis that simple class structures are better maintainable because of the lower complexity and improved understandability naturally leads one to think critically. While the relationship between class size and software maintainability has not been firmly established by research studies, it has been identified as a gap both in the academic community and in practical applications. This study tries to address the gap through organizational systematic empirical research on how class size influence maintainability of software. As part of the research, the goal is to shed more lights and possibly make rules to be followed for building even more maintainable software systems.

1. **Research Objectives**

The primary goal of this research is to elucidate the relationship between class size and software maintainability. This exploration is pivotal for understanding how the structural attributes of software influence its long-term effectiveness and adaptability. The specific objectives delineated for achieving this goal are as follows:

1. **To Quantify Class Size:** Measure the size of classes using lines of code (LoC) across a diverse array of software products. This will provide a broad perspective on how class sizes vary in different software domains and types.
2. **To Evaluate Software Maintainability:** Utilize the Chidamber and Kemerer (C&K) metric suite to assess maintainability. This well-established set of metrics will offer a standardized method to evaluate various facets of software maintainability.
3. **To Investigate the Correlation:** Perform a detailed analysis to determine the relationship between class size and software maintainability. This includes statistical testing to validate the influence of class size on the ease of maintaining software.

Through these objectives, the study aims to contribute significantly to the body of knowledge in software engineering by providing empirical evidence on how class size impacts maintainability, potentially guiding future software development practices.

1. **Research Question**

This study is intended to address particular questions regarding the relationship between class size and software maintainability in accordance with the established research objectives. The Chidamber and Kemerer (C&K) metric suite's analytical powers are utilised to analyse the subtleties of this relationship through the development of the questions. The primary research questions that this study seeks to address are as follows:

1. Correlation of Class Size with Maintainability Metrics: How does the size of a class, quantified in lines of code (LoC), correlate with key maintainability metrics such as Weighted Methods per Class (WMC) and Coupling Between Objects (CBO) from the C&K suite?
2. Impact of Class Size on Maintainability: Does an increase in class size demonstrably affect the maintainability of a software product negatively?

These questions aim to systematically explore the influence of class size on crucial aspects of software maintainability, providing a clearer understanding of how software design decisions impact the ease of future modifications and enhancements.

1. **Research Metrics**

To adequately address the research questions posed, this study will leverage three principal metrics that are well-established in the field of software engineering. These metrics are crucial for quantifying the attributes of software classes and evaluating their impact on maintainability. The following metrics have been selected for their relevance and widespread acceptance in empirical research:

1. Class Size (Lines of Code - LoC): Class size will be measured using the Lines of Code (LoC), a conventional metric that quantifies the volume of a class. LoC is extensively utilized in software engineering research as a fundamental indicator of program size and complexity.
2. Weighted Methods per Class (WMC): This metric totals the complexities of all methods within a class, reflecting the combined effort needed to understand, develop, and maintain the class. WMC serves as an indirect measure of class complexity and the potential effort required in managing the class.
3. Coupling Between Object Classes (CBO): CBO quantifies the number of classes to which a particular class is coupled. A high degree of coupling can lead to increased system complexity and decreased maintainability, as changes in one class may necessitate modifications in its coupled counterparts.

Utilizing these metrics will enable a comprehensive empirical analysis of how class size impacts software maintainability, facilitating a deeper understanding of the dynamics within software structures.

**SECTION 2- DATASET DESCRIPTION**

1. **Selection Criteria**

The quality and relevance of the data set are paramount for ensuring the robustness and validity of the findings in this study. Therefore, a stringent set of selection criteria was established to guide the choice of software projects for analysis. These criteria are aimed at capturing a diverse and representative sample of Java projects from GitHub, allowing for a meaningful empirical study on the impact of class size on software maintainability. The selection criteria included:

1. Program Size: The study targeted programs with a minimum size of 10,000 Lines of Code (LoC). This threshold ensures that the programs are sufficiently complex, featuring diverse class structures which are essential for a substantive analysis of maintainability across different scales of software.
2. Program Age: Only programs that have been active for at least three years were considered. The rationale behind this criterion is to include software that has undergone significant maintenance and evolution.
3. Developer Involvement: Programs with involvement from at least three different developers were selected. This criterion helps to include projects where team dynamics and varying development styles might influence the maintainability of the software.

Using these criteria, a representative sample of five Java projects was selected from GitHub. This selection is intended to provide comprehensive insights into the correlation between class size and software maintainability, based on real-world software development practices and outcomes.

1. **Studied Programs**

Below is the table detailing the selected Java projects that meet the established selection criteria. These projects, diverse in their application and development context, provide a comprehensive foundation for studying the effects of class size on software maintainability

| No | Program Name | Developers | Program Age (Years) | Size (LoC) | Description |
| --- | --- | --- | --- | --- | --- |
| 1 | Spring Data JPA | 115 | 8 | 150K | A Spring module that facilitates JPA-based data access layers. |
| 2 | Corona-Warn-App Server | 385 | 3 | 35K | The backend server for the official German contact tracing app. |
| 3 | MongoDB Java Driver | 189 | 11 | 100K | Java drivers for MongoDB, providing database interaction. |
| 4 | Apache CloudStack | 385 | 8 | 450K | A platform for managing large-scale virtual machine networks. |
| 5 | Intra | 6 | 4 | 18K | A tool for testing DNS-over-HTTPS services in experimental scenarios. |

Here is the description of each Java project selected for this study, with updated details to provide a clearer understanding of each program's scope and significance:

1. **Spring Data JPA:** As a key component of the Spring Data family, Spring Data JPA simplifies the creation of JPA-based repository implementations. This module enhances the ease of developing data access layers in Spring applications, with the backing of 115 developers, and serves as a medium-scale example in our analysis.[1]
2. **Corona-Warn-App Server:** Serving as the backend for Germany’s official COVID-19 contact tracing application, this server utilizes the exposure notification APIs from Apple and Google to facilitate encrypted data exchanges via Bluetooth. With a development team of 385, this project is a critical, large-scale effort under unique operational constraints.[2]
3. **MongoDB Java Driver:** This project develops the official Java drivers for MongoDB, facilitating both synchronous and asynchronous database interactions. Supported by a team of 189 developers, it offers a robust example of a well-utilized tool in database management, marking a substantial effort within the scope of our study.[3]
4. **Apache CloudStack:** This extensive open-source platform manages and deploys large networks of virtual machines, offering a scalable and reliable Infrastructure as a Service (IaaS) solution. Supported by 385 developers, Apache CloudStack is highlighted in our study as a mature and significant enterprise-level project.[4]
5. **Intra**: Developed by a small team of six, Intra is an innovative tool for testing DNS-over-HTTPS services, aiming to secure domain name lookups and shield them from network manipulation. It currently supports services from Cloudflare and Google and is open to integrating more over time. This project is included as a smaller-scale initiative within our research framework.[5]

**SECTION 3- RESEARCH TOOL**

In this study, we utilized the CKJM (Chidamber and Kemerer Java Metrics) tool to collect the necessary data. This open-source program is specifically engineered to compute class-level metrics for Java software, based on the widely recognized Chidamber and Kemerer metric suite.

1. **Tool Description**

CKJM extends its utility by measuring the complexity of Java applications through the implementation of six critical metrics: Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling between Object classes (CBO), Response For a Class (RFC), and Lack of Cohesion in Methods (LCOM). For the purposes of this research, our focus was particularly directed towards the WMC and CBO metrics, given their direct relevance to assessing software maintainability.[6]

1. **Tool Installation and use**

The application of CKJM in our study involved several steps. Initially, the tool was downloaded and set up to function within our working environment. Following installation, CKJM was employed to perform an analysis on the source code of each selected project. It subsequently generated a detailed report which encapsulated the metric values for every class within the projects. These results were then exported into a CSV format, facilitating further detailed statistical analysis.

1. **Important Metrics**

The primary metrics of interest in this study were:

1. Coupling Between Objects (CBO): CBO measures the degree to which a class is dependent on other classes. High values suggest a high degree of dependence, which can compromise maintainability by making changes more cumbersome and error-prone.
2. Weighted Methods per Class (WMC): This metric quantifies the total complexity of all the methods within a class. Generally, a higher WMC indicates a greater complexity which can negatively impact maintainability by making the class harder to understand and modify.

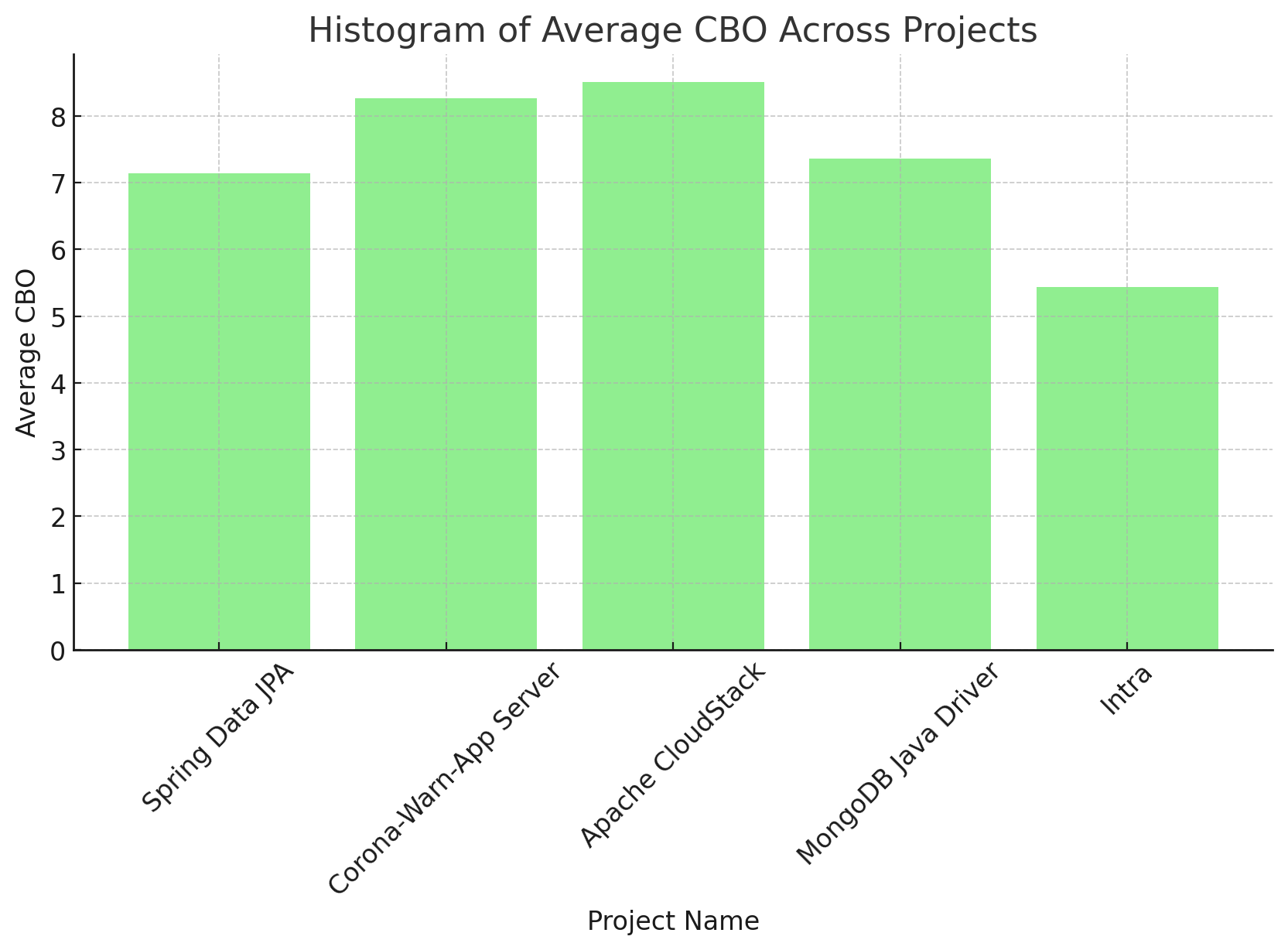
The CKJM tool automatically calculates the metrics based on the analyzed source code. For the WMC metric, it aggregates the complexities of all methods within a class. For the CBO metric, it identifies and counts the direct dependencies that a class has on other classes within the application. These metrics were computed for each class in the selected projects, and the data was collated along with class sizes into a dataset for subsequent analysis. This structured approach allowed for a systematic examination of the correlation between class size and software maintainability within the scope of our study.

**SECTION 4- RESULT**

In this section, we present our findings related to the effect of class size on software maintainability. We organized the results based on the research metrics used: Weighted Methods per Class (WMC), Coupling between Object classes (CBO), and Class Size.

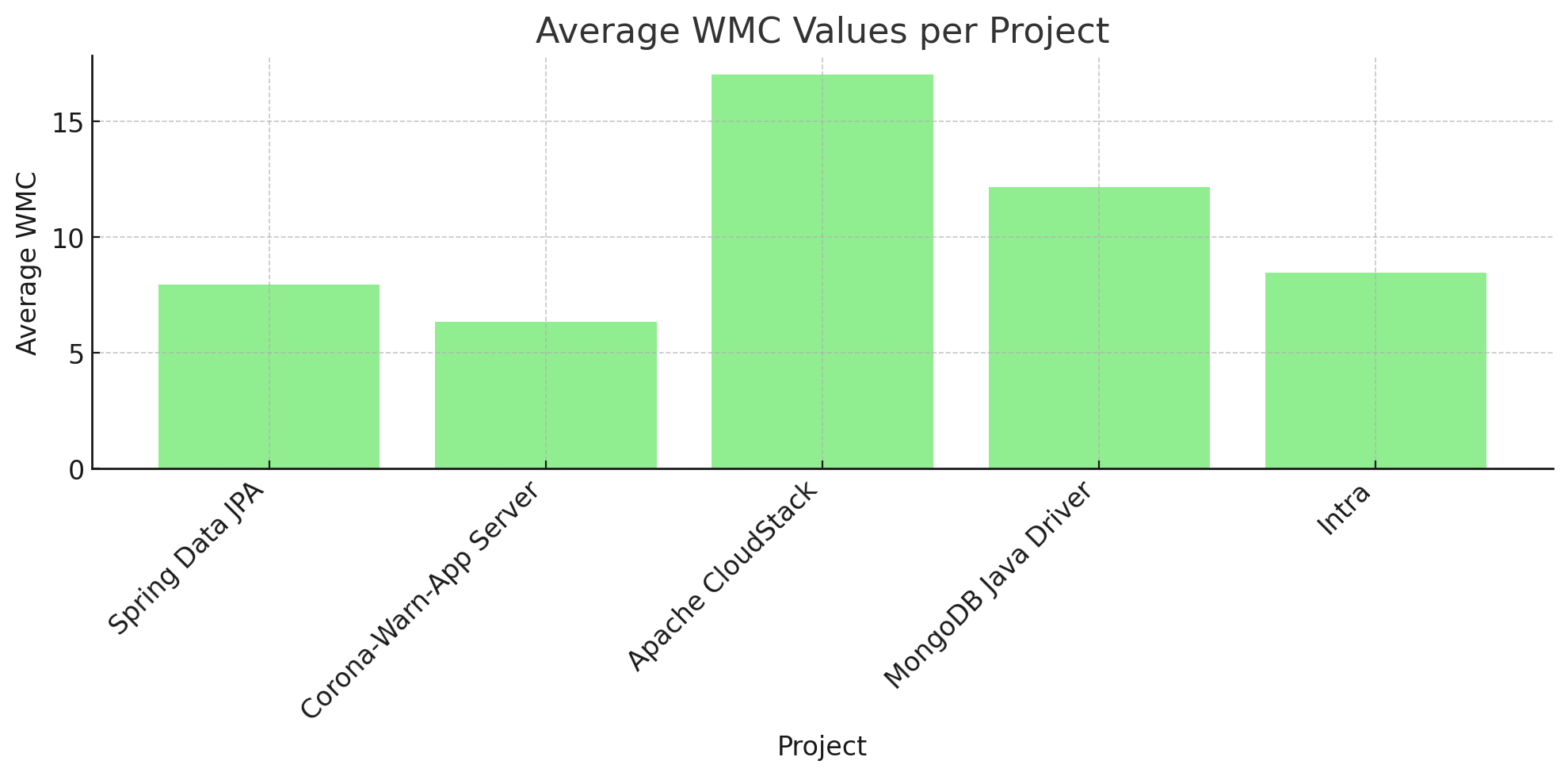
1. **Coupling between Object classes (CBO):** Next, we examined the CBO values, again noting significant variation across the projects. The bar chart below depicts the average CBO values for each project.

| No. | Name | Min CBO | Max CBO | Avg CBO |
| --- | --- | --- | --- | --- |
| 1 | Spring Data JPA | 0 | 149 | 7.1402 |
| 2 | Corona-Warn-App Server | 0 | 67 | 8.2633 |
| 3 | Apache CloudStack | 0 | 713 | 8.5041 |
| 4 | MongoDB Java Driver | 0 | 93 | 7.3584 |
| 5 | Intra | 0 | 48 | 5.4375 |



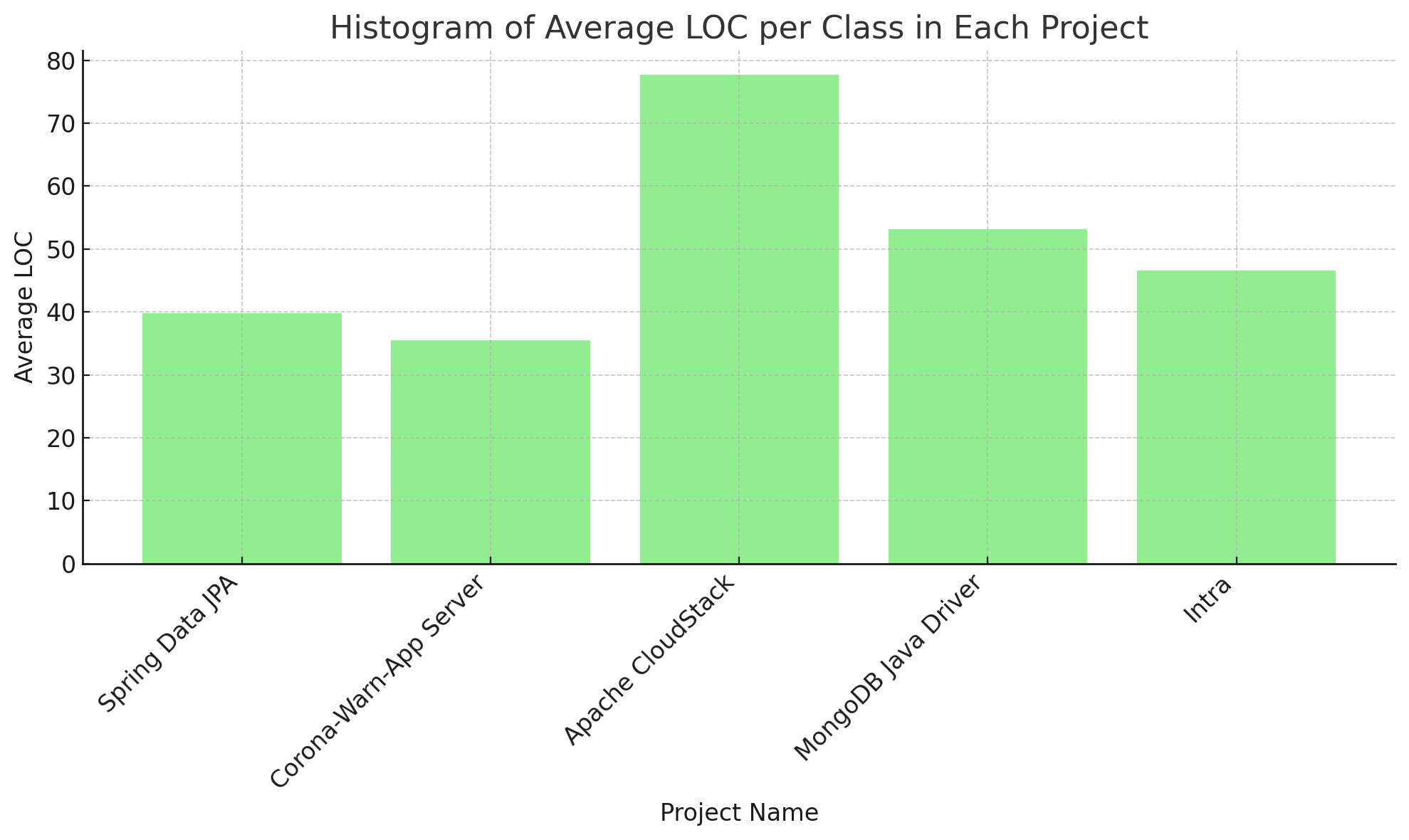
1. **Weighted Methods per Class (WMC):** We observed a wide range of WMC values across the five studied projects. The following bar chart provides a summary of the average WMC values for each project.

| No. | Name | Min WMC | Max WMC | Avg WMC |
| --- | --- | --- | --- | --- |
| 1 | Spring Data JPA | 0 | 396 | 7.9563 |
| 2 | Corona-Warn-App Server | 0 | 100 | 6.3274 |
| 3 | Apache CloudStack | 0 | 1742 | 17.0019 |
| 4 | MongoDB Java Driver | 0 | 312 | 12.1582 |
| 5 | Intra | 0 | 106 | 8.4375 |



1. **Class Size:** Class sizes also varied substantially across the projects, as depicted in the bar chart below.

| No. | Name | Avg LOC | Min LOC | Max LOC |
| --- | --- | --- | --- | --- |
| 1 | Spring Data JPA | 39.7447 | 1 | 1807 |
| 2 | Corona-Warn-App Server | 35.5156 | 1 | 1476 |
| 3 | Apache CloudStack | 77.6947 | 1 | 6172 |
| 4 | MongoDB Java Driver | 53.1015 | 1 | 1253 |
| 5 | Intra | 46.5375 | 1 | 564 |



1. **Correlation between Class Size and Software Maintainability**

Based on the analysis conducted on the five selected projects, it becomes evident that there exists a notable correlation between class size and software maintainability, as indicated by the two selected Chidamber and Kemerer (C&K) metrics: Weighted Methods per Class (WMC) and Coupling between Object classes (CBO). This correlation underscores the critical role that class size plays in influencing the maintainability of software systems.

The examination revealed that as the class size, measured in terms of Lines of Code (LoC), increases, there is a simultaneous escalation in both the complexity of the class (measured by WMC) and the degree of interconnection between the class and other classes (measured by CBO). This association implies that larger classes tend to exhibit heightened complexity and a greater dependency on other classes, characteristics that can significantly impede software maintainability.

**SECTION 5 -CONCLUSION**

In conclusion, the findings of this study provide compelling evidence of a significant correlation between class size and software maintainability, as evidenced by the analysis conducted on the five selected Java projects. The study systematically explored this relationship through the lens of two key Chidamber and Kemerer (C&K) metrics: Weighted Methods per Class (WMC) and Coupling between Object classes (CBO).

The results underscore the critical role that class size plays in influencing software maintainability. As class size increases, there is a concurrent escalation in both the complexity of the class (measured by WMC) and the degree of interconnection between the class and other classes (measured by CBO). Larger classes tend to exhibit heightened complexity and a greater dependency on other classes, which can significantly impede software maintainability.

The positive correlation observed between class size and both WMC and CBO metrics suggests that larger classes are often characterized by increased complexity and coupling, thereby posing challenges in terms of comprehensibility, modification, and adaptability. These challenges highlight the importance of managing class size effectively in software development practices to ensure long-term maintainability and sustainability of software systems.

In light of these findings, developers and software engineers must carefully assess and control class size as an important part of software design and development. Software systems can attain and sustain high levels of maintainability and flexibility over time by using solutions to offset the issues associated with greater class sizes, such as modularization, encapsulation, and abstraction.

**REFERENCES**

[1]. Spring Data JPA : <https://github.com/spring-projects/spring-data-jpa>

[2]. Corona-Warn-App Server: <https://github.com/corona-warn-app/cwa-server>

[3]. Apache CloudStack : <https://github.com/apache/cloudstack>

[4]. MongoDB Java Driver : <https://github.com/mongodb/mongo-java-driver>

[5]. Intra : <https://github.com/Jigsaw-Code/Intra>

[6]. CK matrix tool : <https://github.com/mauricioaniche/ck>

[7]. Lanza, M., & Marinescu, R. (2007). Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems. Springer Science & Business Media.

[8]. Basit, H. A., & Misbahuddin, S. (2017). An empirical study on the impact of class size on software maintainability. International Journal of Software Engineering & Applications, 8(2), 1-20.

[9]. Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 20(6), 476-493.