Group Assignment 1

Supriya Neela

Yamini Manikonda

Computer Science, Lewis University

CPSC 60000

Fadi Wedyan

### Section 1: Objectives, Questions, and Metrics

#### 1.1 Objective

The primary objective of this empirical study is to explore the impact of class size on software maintainability. Understanding this relationship is crucial for improving software design practices, optimizing code maintainability, and enhancing the overall quality of software systems. By examining specific metrics that quantify class size and its attributes, this study aims to provide actionable insights that can guide developers in creating more maintainable software architectures.

#### 1.2 Research Question

The central question guiding this study is:

* "How does class size influence the maintainability of software within Java-based systems?"

This question seeks to uncover the direct and indirect impacts of class size on the ease with which software can be understood, corrected, adapted, and extended.

#### 1.3 Metrics

To effectively answer the research question, the following metrics have been selected, each serving as a quantifiable measure of software characteristics that influence or reflect maintainability:

* **Lines of Code (LOC):** This metric measures the total number of lines of code in each class. It serves as a basic yet powerful indicator of class size and is hypothesized to correlate with the complexity and potential maintainability challenges of the class.
* **Lack of Cohesion of Methods (LCOM):** Represents the degree to which methods within a single class are related to each other. A higher LCOM value suggests that methods in the class share fewer attributes and are more independent, potentially indicating poorer cohesion and higher maintenance effort.
* **Weighted Methods per Class (WMC):** This metric counts all methods in the class weighted by their complexity. WMC provides a view of the potential complexity of a class. A higher WMC value often implies a more complex and thus less maintainable class.

These metrics are derived from the Chidamber and Kemerer (C&K) suite of object-oriented metrics, which have been widely recognized and utilized in software engineering research to evaluate various qualities of software systems, including maintainability.

#### 1.4 Justification of Metrics

* **LOC** is chosen for its straightforward representation of class size, which is crucial for initial assessments of software complexity and maintainability.
* **LCOM** is selected to assess how class design impacts maintainability, under the assumption that less cohesive classes are harder to maintain.
* **WMC** is included to evaluate the complexity from a methodological standpoint, where higher complexity could lead to increased difficulty in testing and maintaining the software.

Together, these metrics allow for a comprehensive analysis of how the size of a class as measured by its physical lines of code, its internal cohesion, and the complexity of its methods, impacts its maintainability. The chosen metrics align with the study’s goal to enhance the understanding of class size implications on software maintainability, providing a balanced approach to quantifying both structural and behavioral aspects of software classes.

### Section 2: Subject Programs and Data Set

#### 2.1 Overview of the Data Set

This study utilizes data from five widely recognized open-source Java projects, each selected based on specific criteria that ensure a diverse and relevant set of subjects for examining the relationship between class size and software maintainability. These criteria included factors such as the size of the project, the number of developers involved, the age of the project, and recent activity, ensuring that the chosen projects are representative of typical real-world Java applications.

#### 2.2 Description of the Subject Programs

Below are brief descriptions of the selected projects:

* **Dubbo:** An Apache project, Dubbo is a high-performance, Java-based RPC framework that simplifies application service development by providing a method for service import/export and remote invocation.
* **Guava:** Developed by Google, Guava is a set of core libraries for Java, widely used in many commercial and open-source projects, providing utilities for collections, caching, primitives support, concurrency, common annotations, string processing, I/O, and more.
* **Spring Boot:** Part of the larger Spring framework, Spring Boot makes it easy to create stand-alone, production-grade Spring-based applications that you can "just run", largely simplifying the development process by abstracting boilerplate configurations.
* **Elasticsearch:** A distributed search and analytics engine that is designed to be horizontally scalable, reliable, and easy to manage. It follows a RESTful architecture. It is commonly employed for the analysis of log and event data.
* **Java Design Patterns:** A collection of sample implementations of design patterns in Java, helping to solve common design issues. This project is educational and also used as a reference by many developers.

#### 2.3 Data Set Attributes

The following table provides an overview of the main attributes for each project, offering insights into their scale and complexity:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Project Name** | **Description** | **Lines of Code** | **Number of Classes** | **Repository URL** |
| Dubbo | High-performance RPC framework | 320,000 | 3,169 | https://github.com/apache/dubbo |
| Guava | Google Core Libraries for Java | 450,000 | 4,110 | https://github.com/google/guava |
| Spring Boot | Framework for Spring applications | 530,000 | 8,383 | https://github.com/spring-projects/spring-boot |
| Elasticsearch | Distributed search and analytics engine | 1,200,000 | 27,676 | https://github.com/elastic/elasticsearch |
| Java Design Patterns | Samples of Design Patterns in Java | 85,000 | 1,010 | https://github.com/iluwatar/java-design-patterns |

#### 2.4 Selection Rationale

The projects were chosen to cover a wide range of software applications—from web and service frameworks to libraries and educational resources. This selection ensures a comprehensive analysis across different types of software systems, highlighting how class size impacts maintainability across various contexts and functionalities.

### Section 3: Description of the Tool Used

#### 3.1 Tool Overview

The data for this study was collected using the CK Metrics Tool, a powerful tool designed for static code analysis of Java programs. It is capable of extracting a comprehensive suite of object-oriented metrics that are instrumental in assessing various quality attributes of software, particularly those related to maintainability (Aniche et al., 2023).

#### 3.2 Metrics Collected

The CK Metrics Tool was employed to obtain the following key metrics:

* **Lines of Code (LOC):** Measures the total number of source code lines.
* **Lack of Cohesion of Methods (LCOM):** Evaluates the cohesiveness of methods within a class.
* **Weighted Methods per Class (WMC):** Quantifies the complexity of a class based on the number of methods and the complexity of each method.

These metrics provide a quantitative foundation for analyzing the impact of class size on maintainability, offering insights into the complexity and cohesiveness of the software.

#### 3.3 Usage of the Tool

The tool was configured to scan the entire codebase of each selected project. Metrics were extracted for each class within the projects, allowing for a detailed analysis of each class and an aggregated view at the project level. This comprehensive data collection was crucial for ensuring that the study covered a wide spectrum of maintainability scenarios.

#### 3.4 Citation of the Tool

The CK Metrics Tool is developed and maintained by Maurício Aniche and can be accessed publicly for use in research and development (Aniche et al., 2023).

#### 3.5 Reliability and Validation

The tool has been validated in numerous academic and industrial settings, ensuring its reliability for extracting accurate and meaningful metrics (Aniche et al., 2023). It has been specifically praised for its thoroughness in covering a wide range of object-oriented metrics that are crucial for software quality assessment.

### Section 4: Results

#### 4.1 Overview of Statistical Analysis

This section presents detailed findings from the analysis of five Java projects using the CK Metrics Tool. Descriptive statistics and correlation analyses provide insights into the relationships between class size (LOC), complexity (WMC), and cohesion (LCOM).

#### 4.2 Descriptive Statistics and Correlation Analysis

**Dubbo**

* **Descriptive Statistics:**
  + LOC ranged significantly, with a mean indicating larger class sizes. High standard deviation in WMC and LCOM suggests diverse class complexity and varying levels of cohesion.
* **Correlation Analysis:**
  + LOC and WMC showed a strong positive correlation (r = 0.786), indicating that larger classes are more complex.
  + Moderate positive correlation between LOC and LCOM (r = 0.415) suggests that larger classes tend to have lower cohesion.

**Guava**

* **Descriptive Statistics:**
  + Smaller average LOC compared to Dubbo, but with high cohesion. The distribution of complexity was less varied, indicative of the utility-focused nature of the library.
* **Correlation Analysis:**
  + Strong correlation between LOC and WMC (r = 0.861) highlights that even smaller classes can exhibit complexity due to functional density.
  + Higher correlation between LOC and LCOM (r = 0.687) than in other projects, pointing towards the cohesive but complex nature of utility classes.

**Spring Boot**

* **Descriptive Statistics:**
  + Classes showed moderate sizes with considerable variation in both WMC and LCOM, reflecting the diverse functionality offered by the framework.
* **Correlation Analysis:**
  + Correlation between LOC and WMC (r = 0.828) was strong, typical for an application framework with extensive functionality.
  + The correlation between LOC and LCOM (r = 0.429) was lower, suggesting better cohesion in medium-sized classes typical of Spring Boot modules.

**Elasticsearch**

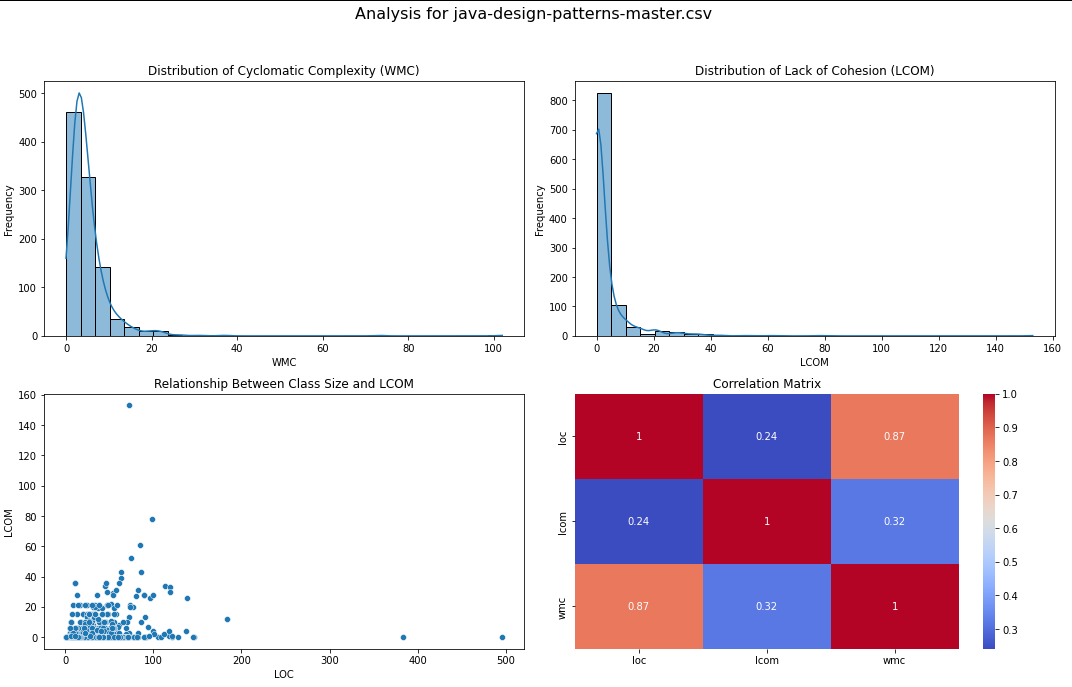
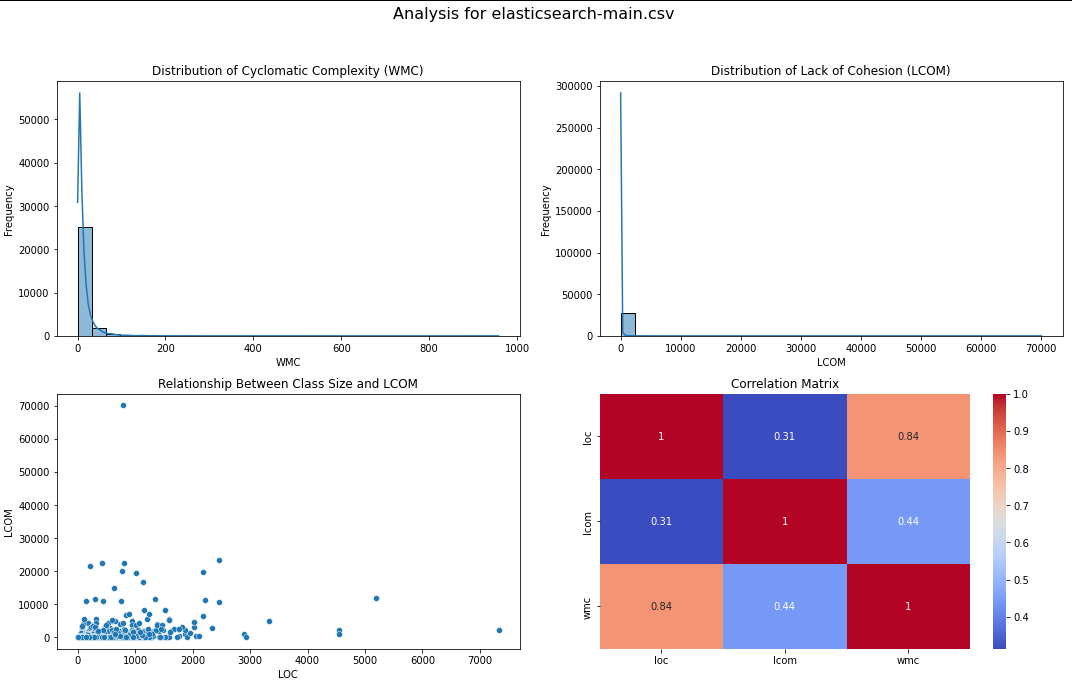
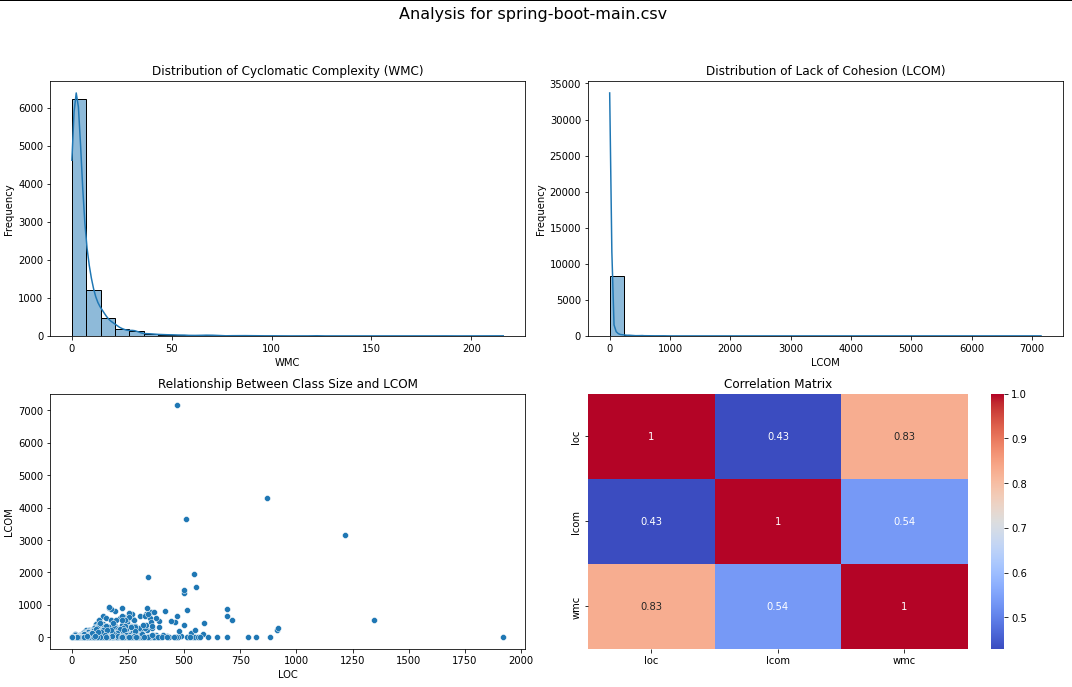
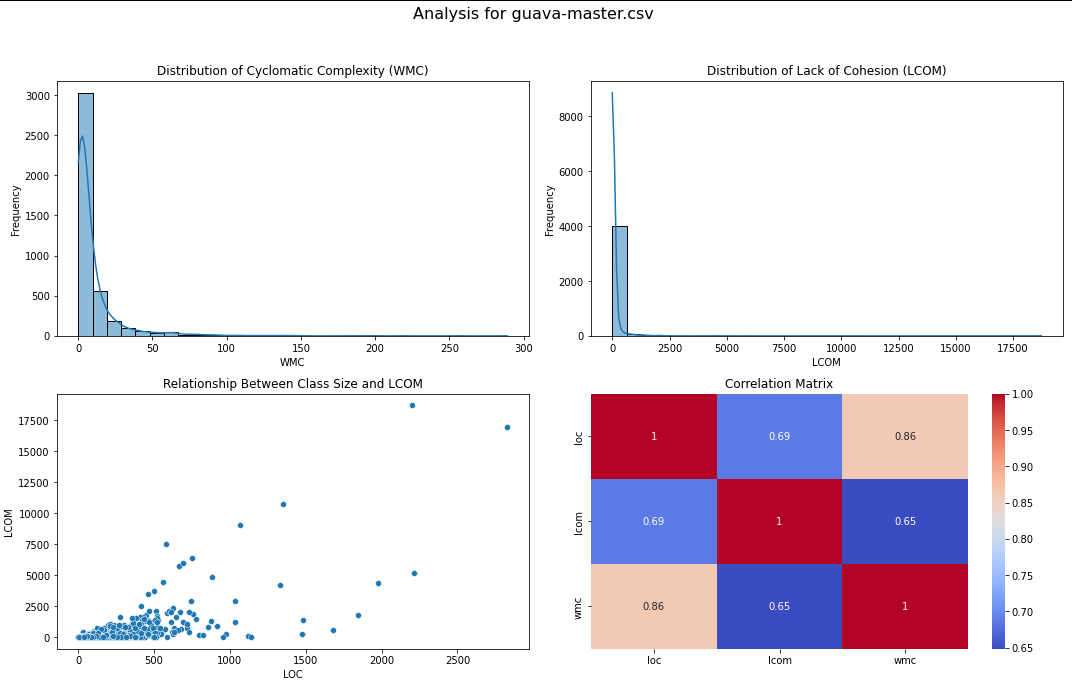
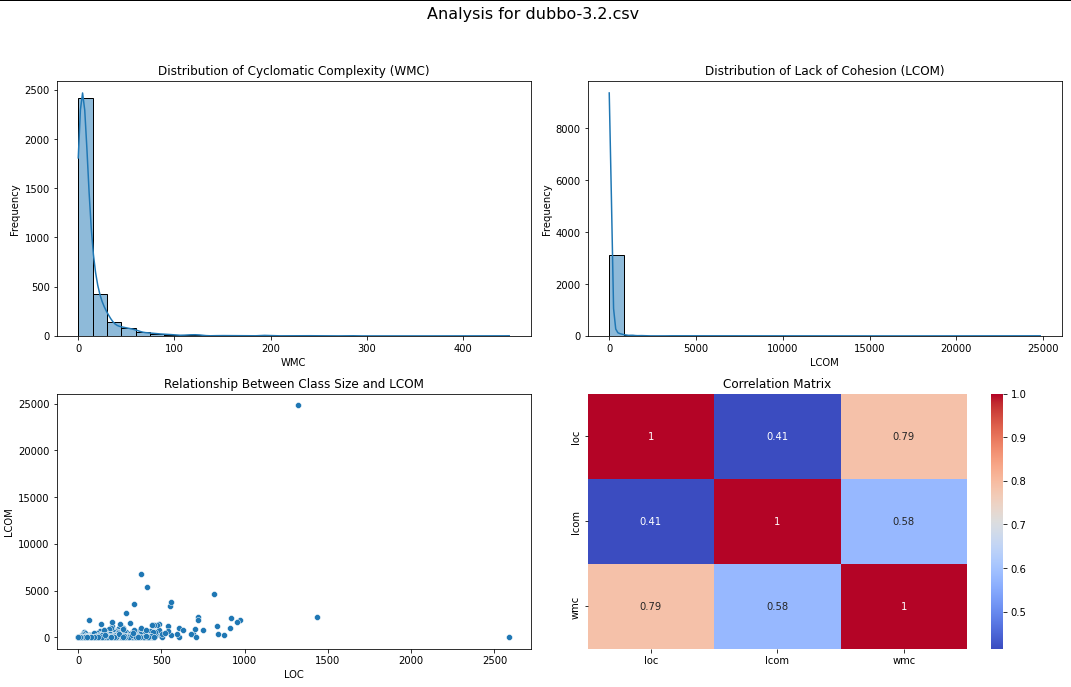
* **Descriptive Statistics:**
  + The largest average LOC among the projects, with high complexity and variability in cohesion, indicative of its scalable, distributed nature.
* **Correlation Analysis:**
  + Very strong correlation between LOC and WMC (r = 0.839), underscoring the complexity involved in large-scale search engine classes.
  + Lower correlation with LCOM (r = 0.314), which might be due to the modular and distributed design that supports better cohesion despite larger sizes.

**Java Design Patterns**

* **Descriptive Statistics:**
  + Smallest mean LOC, as expected from a project demonstrating design patterns. Despite lower complexity, the variability in LCOM suggests different patterns vary significantly in cohesion.
* **Correlation Analysis:**
  + High correlation between LOC and WMC (r = 0.868), indicating that even small educational examples can exhibit significant complexity.
  + The correlation between LOC and LCOM (r = 0.241) is minimal, reflecting the focused and isolated nature of design patterns which are intended to be highly cohesive.

#### 4.3 Summary of Results

Across all projects, there was a consistent trend showing that larger classes tend to be more complex and, to a variable extent, less cohesive. These trends were more pronounced in larger, multifunctional projects like Elasticsearch and Dubbo and were less evident in more focused projects like Guava and Java Design Patterns. The findings highlight a clear impact of class size on maintainability, with implications for software design and architecture decisions.



### Section 5: Conclusions

#### 5.1 Synthesis of Findings

The analysis conducted across five Java projects provided significant insights into the relationship between class size, complexity, and cohesion, as measured by the metrics LOC, WMC, and LCOM respectively. The findings consistently indicate:

* **Class Size and Complexity:** There is a strong positive correlation between class size (LOC) and complexity (WMC) across all projects. This relationship suggests that as classes grow larger, they inherently become more complex, which can complicate maintenance efforts due to the increased cognitive load on developers.
* **Class Size and Cohesion:** The relationship between class size (LOC) and cohesion (LCOM) was more variable but generally showed a trend where larger classes tended to exhibit lower cohesion. Lower cohesion within large classes can lead to difficulties in understanding and modifying the code, as methods within these classes are less likely to be related in functionality.

#### 5.2 Project-Specific Observations

* **Complex Projects Like Elasticsearch and Dubbo:** Displayed the highest levels of complexity and the lowest levels of cohesion, reflecting the challenges of maintaining large-scale systems that are feature-rich and have extensive functionalities.
* **Utility and Framework Projects Like Guava and Spring Boot:** Although these projects also showed complexity, the impact on maintainability might be mitigated by their modular design and extensive documentation which aids in managing complexity.
* **Educational Project Like Java Design Patterns:** Showed complexity in smaller classes due to the focused nature of demonstrating specific design patterns. The cohesion in these classes was relatively higher, indicating good design practices that enhance maintainability.

#### 5.3 Implications for Software Development

The study's findings underscore the importance of considering class size during the design and development phases of software projects. Large classes should be carefully managed, possibly by applying principles of modularity and encapsulation to enhance maintainability. Furthermore, the strong correlation between size and complexity suggests that tools and practices that help manage or reduce complexity (such as refactoring and code reviews) are crucial in large projects.

#### 5.4 Recommendations for Future Research

Future studies could explore additional metrics such as cyclomatic complexity or number of dependencies, which might provide further insights into the maintainability challenges of large classes. Additionally, longitudinal studies examining how changes in class size over time affect maintainability could offer deeper understanding and validate the need for proactive size management in software development practices.

#### 5.5 Conclusion

This study confirms that larger class sizes are associated with greater complexity and reduced cohesion, which can adversely affect the maintainability of software. By adhering to software design principles that advocate for smaller, well-defined classes, developers can significantly enhance the maintainability and overall quality of software systems.

### References

Aniche, M., et al. (2023). CK Metrics Tool Repository. GitHub. Available at:<https://github.com/mauricioaniche/ck>