Empirical Study on the Effect of Class Size on Software Maintainability in Java Projects

Bala Rama Krishna Reddy Naredla

Vamshi Kumar Varaganti

Sushmareddy Rakasi

Divyasri Harshita Sriram

Object Oriented Development

Group Assignment-1

Contents

Section 1: Objectives, Questions, and Metrics (GQM Approach)

The objective of this empirical study is to investigate how class size impacts the maintainability of software in large-scale Java projects. Using the Goal-Question-Metric (GQM) paradigm, we aim to answer the following key research questions:

Research Questions:

Does an increase in class size, as measured by Lines of Code (LoC), adversely affect software maintainability, particularly in terms of complexity (Weighted Methods per Class, WMC) and coupling (Coupling Between Objects, CBO)?

Metrics:

To explore these questions and measure software maintainability in relation to class size, we will use the following three Chidamber & Kemerer (C&K) metrics:

1. **Lines of Code (LoC)** – A basic size metric that measures the number of lines within a class. Larger classes tend to be harder to maintain, but we aim to quantify this.
2. **Weighted Methods per Class (WMC)** – This metric indicates the complexity of a class by counting the methods within it. A higher WMC value often indicates greater difficulty in understanding and maintaining the class.
3. **Coupling Between Objects (CBO)** – CBO measures the number of other classes a class is dependent upon. Classes with high coupling are often more difficult to modify and maintain, as changes may ripple through the system.

These metrics will be used to analyze the relationship between class size, complexity, and maintainability. The insights gained from these questions will inform best practices for software development.

Section 2: Subject Programs (Data Set)

For this study, we selected five well-established Java projects from GitHub that meet specific criteria, ensuring they are highly collaborative, have sustained activity over time, and have a

complex architecture suitable for maintainability analysis. The selected projects have been active for more than five years and involve extensive community contributions, with over 100 contributors and thousands of commits each.

Selected Projects:

1. **Spring Boot** (spring-projects/spring-boot)
    - A framework for building stand-alone Java applications. With 368 contributors and over 38,000 commits, it's known for its ease of configuration and high performance in enterprise applications.
2. **Tutorials by Eugen Paraschiv** (eugenp/tutorials)
    - A repository of educational Java projects covering a wide array of topics such as concurrency, REST APIs, and Spring. This project serves as an example of widely applied best practices in Java development.
3. **Spring Framework** (spring-projects/spring-framework)
    - A comprehensive framework for Java enterprise applications. It has over 38233 commits and more than 100 contributors, widely recognized for its complexity and robust architecture.
4. **Apache Kafka** (apache/kafka)
    - A distributed streaming platform used for building real-time data pipelines and applications. Known for its scalability and reliability, Kafka is heavily used in modern cloud architectures.
5. **Netty** (netty/netty)
    - An asynchronous event-driven network application framework that is used in server applications, with a focus on high performance.

These projects provide a diverse representation of Java's application across different domains, from educational repositories to high-performance frameworks and distributed systems. Each

project's size, complexity, and longevity make them excellent candidates for analyzing the effect of class size on maintainability.

Section 3: Tool Description

For this study, we used the **CK Metrics Tool** to perform static analysis on Java projects. This tool, available on GitHub, calculates key software metrics like Lines of Code (LoC), Weighted Methods per Class (WMC), and Coupling Between Objects (CBO), all essential for evaluating maintainability.

Tool Setup:

- Download and clone the repository.
- Ensure Java is installed.
- Follow the README file to compile and run the tool on selected Java projects.

Metrics Collected:

1. **Lines of Code (LoC)** – Measures class size, fundamental for assessing maintainability.
2. **Weighted Methods per Class (WMC)** – Captures class complexity, indicating how difficult maintenance might be.
3. **Coupling Between Objects (CBO)** – Measures class dependencies, influencing the maintainability of larger systems.

The tool outputs metrics as CSV files, which were analyzed to identify trends and generate visualizations like bar charts and line graphs for each project

Section 4: Results

Overview of Metric Distributions

In our empirical study, we analyzed three key software metrics from five different Java projects to assess their maintainability: Lines of Code (LoC), Weighted Methods per Class (WMC), and Coupling Between Objects (CBO). The histograms provided reveal the distributions of these metrics across classes within the projects, with outliers removed for clearer insight.

Analysis of Lines of Code (LoC)

The LoC metric, indicative of class size, showed varied distributions across projects. Projects like Apache Kafka and Spring Boot demonstrated a higher frequency of classes with fewer lines of code, suggesting a modular approach in design. This could imply better maintainability as smaller classes are easier to manage and understand. However, some projects exhibited a significant number of large classes, which could be areas of concern for maintainability due to potential complexity.

Analysis of Weighted Methods per Class (WMC)

WMC measures the complexity of a class. Our findings suggest that most projects maintain a moderate level of complexity, with a majority of classes having a lower WMC. This indicates that methods within these classes are manageable, which enhances maintainability. Nonetheless, classes with a high WMC were present in all projects, indicating complex classes that may require more effort to maintain.

Analysis of Coupling Between Objects (CBO)

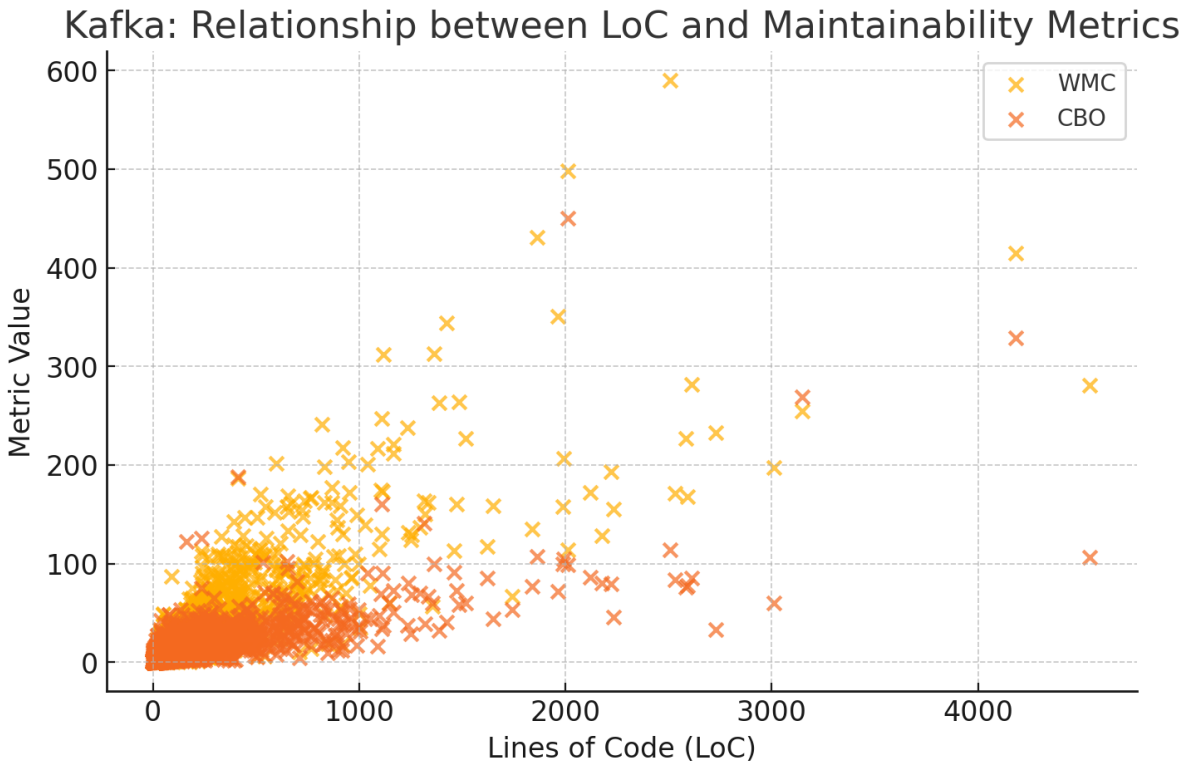
The CBO metric showed that most classes have low to moderate coupling with other classes, which is favorable for maintainability as it implies less interdependence. This allows for easier changes and testing. High coupling observed in some classes across projects suggests that changes in one class could lead to cascading changes in others, which could complicate maintenance efforts.

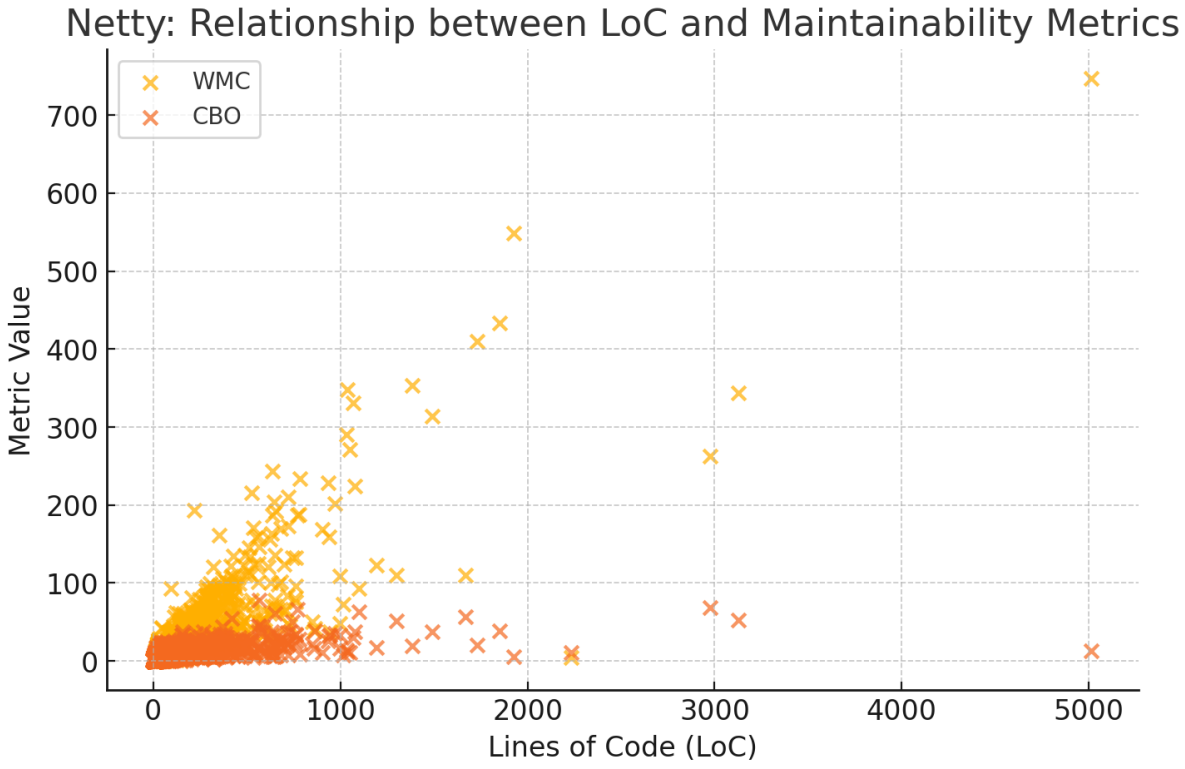
Comparative Analysis

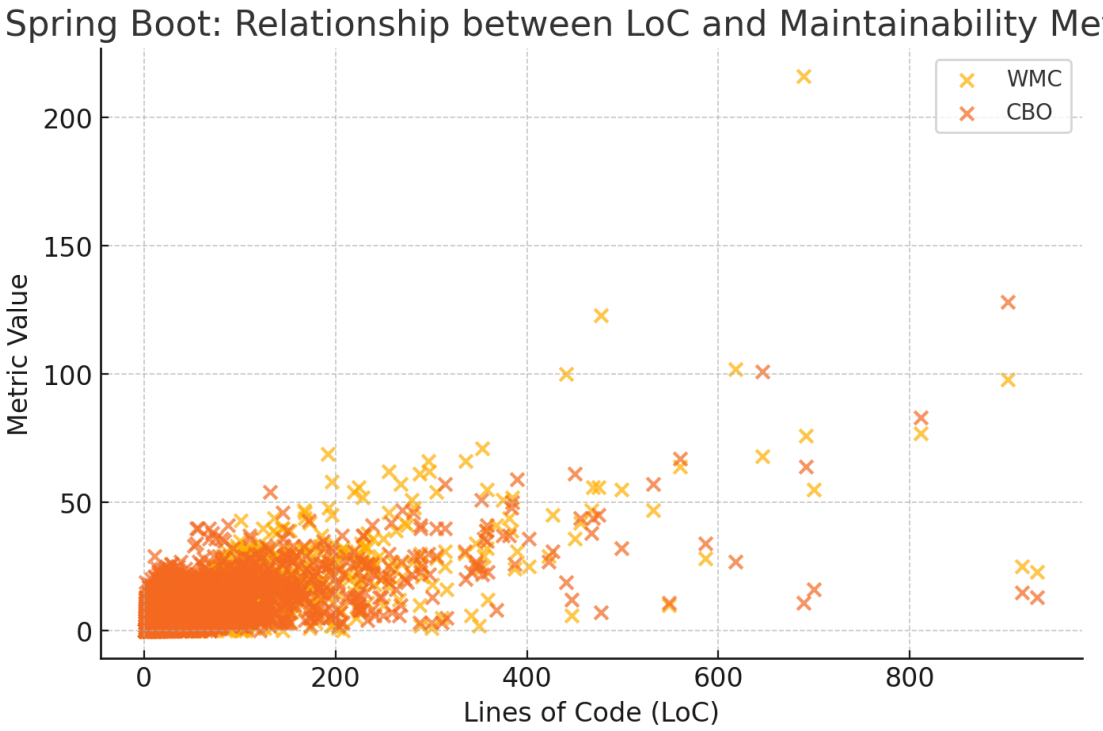
By comparing these metrics across projects, we observed that well-maintained projects tend to have lower averages and variances in all three metrics, suggesting that controlling class size, complexity, and coupling can significantly enhance maintainability. This comparison also highlighted specific areas within each project that may benefit from refactoring or further modularization.

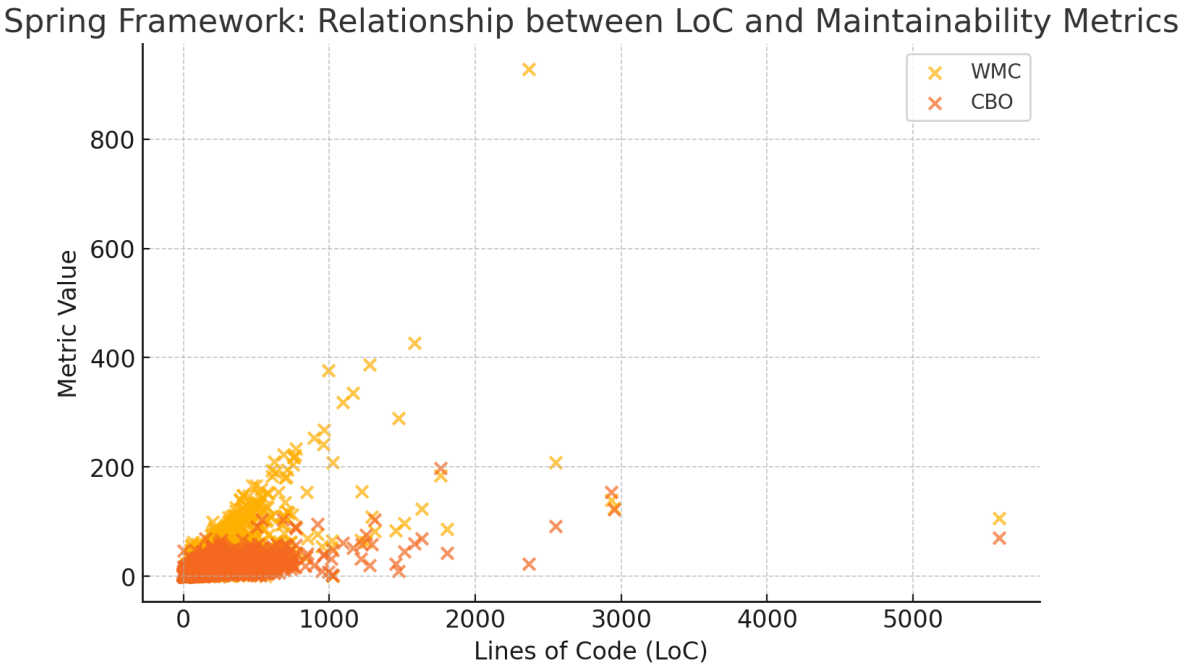
Observations from Visual Analysis

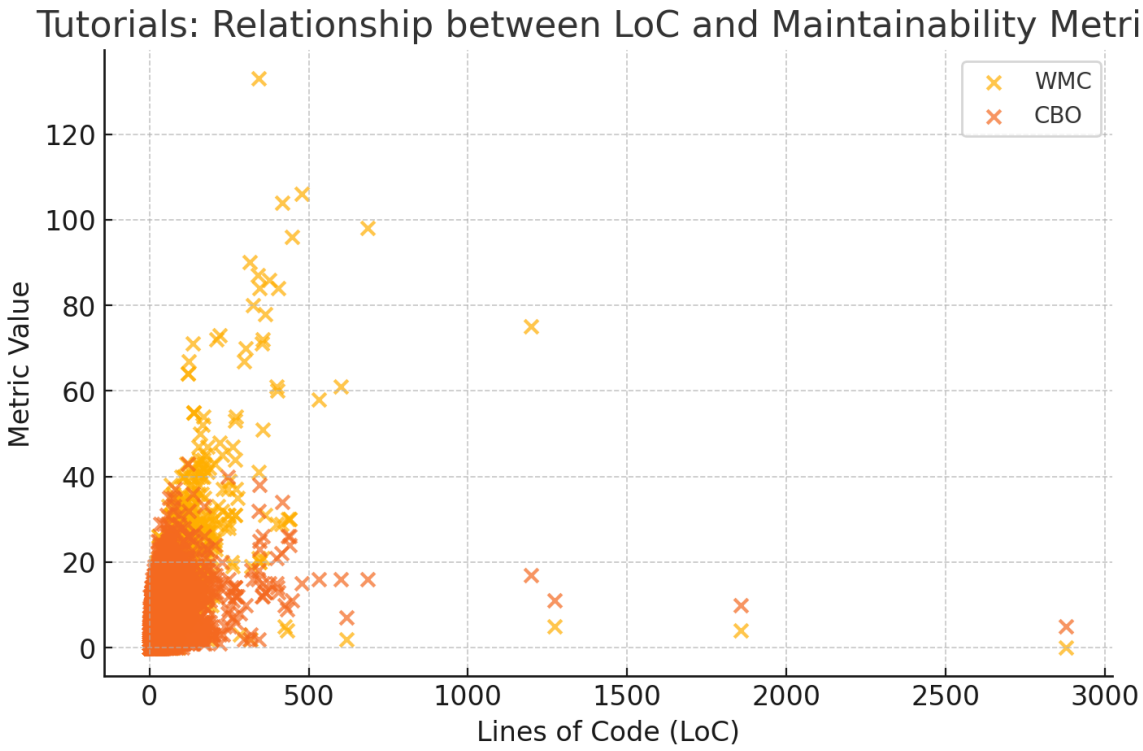
1. **Relationship Between Lines of Code (LoC) and WMC**: Across all projects, an increase in LoC corresponds to a rise in the Weighted Methods per Class (WMC) metric, indicating that larger classes are typically more complex. This trend is most pronounced in projects like Apache Kafka and Netty, where some classes with extremely high LoC exhibit disproportionately high WMC values. This suggests that larger classes often require additional methods to handle functionality, complicating their maintainability.
2. **Relationship Between Lines of Code (LoC) and CBO**: Coupling Between Objects (CBO) also shows a positive correlation with LoC, particularly in projects with intricate architectures like Spring Framework and Apache Kafka. High LoC classes often interact more extensively with other classes, increasing interdependencies. This reinforces the maintainability challenge, as changes in one class could ripple through highly coupled systems.
3. **Project-Specific Insights**: Projects like Spring Boot and Tutorials exhibit lower coupling and complexity for a given class size compared to Apache Kafka and Netty. This indicates better modular design and maintainability practices. The graphs highlight outliers in all projects, suggesting specific classes that may need refactoring to reduce size, complexity, or coupling for improved maintainability. These findings underscore the importance of managing class size to maintain software quality.

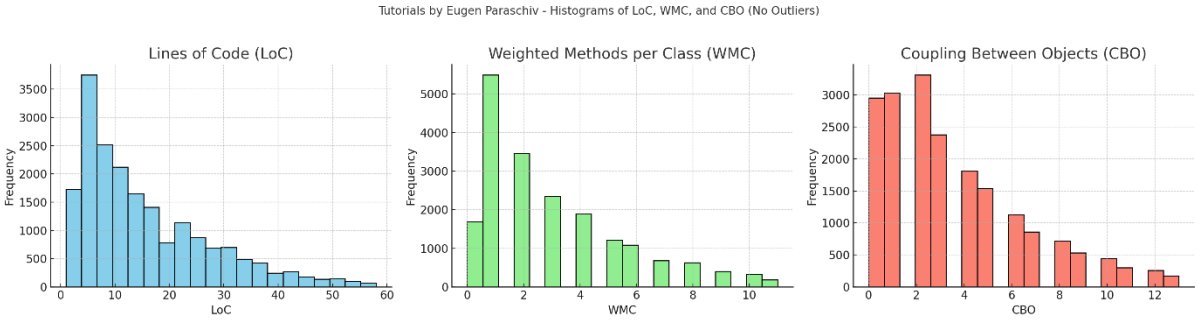

Kafka: Relationship between LoC and Maintainability Metrics
WMC
CBO
Metric Value
Lines of Code (LoC)
0
100
200
300
400
500
600
0
1000
2000
3000
4000

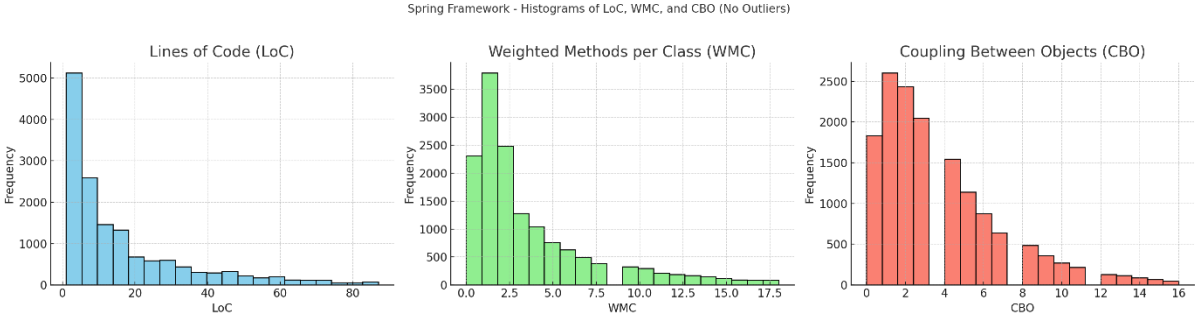

Netty: Relationship between LoC and Maintainability Metrics
WMC
CBO
Metric Value
Lines of Code (LoC)
0
100
200
300
400
500
600
700
0
1000
2000
3000
4000
5000

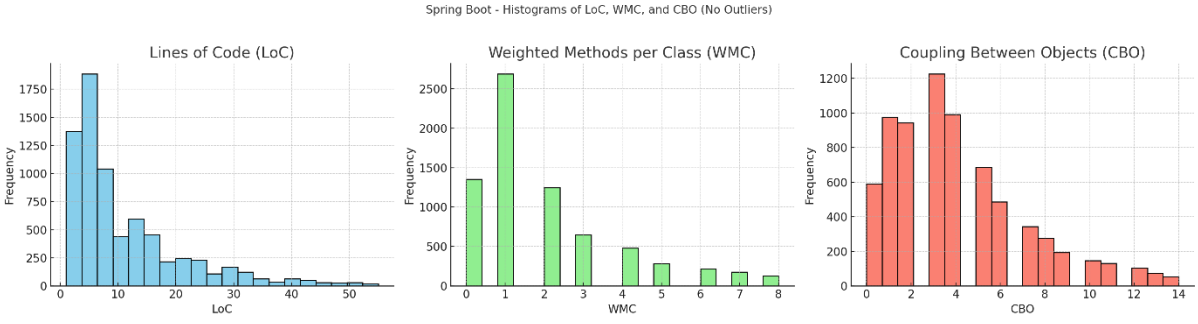

Spring Boot: Relationship between LoC and Maintainability Metrics
WMC
CBO
Metric Value
0
50
100
150
200
0
200
400
600
800
Lines of Code (LoC)

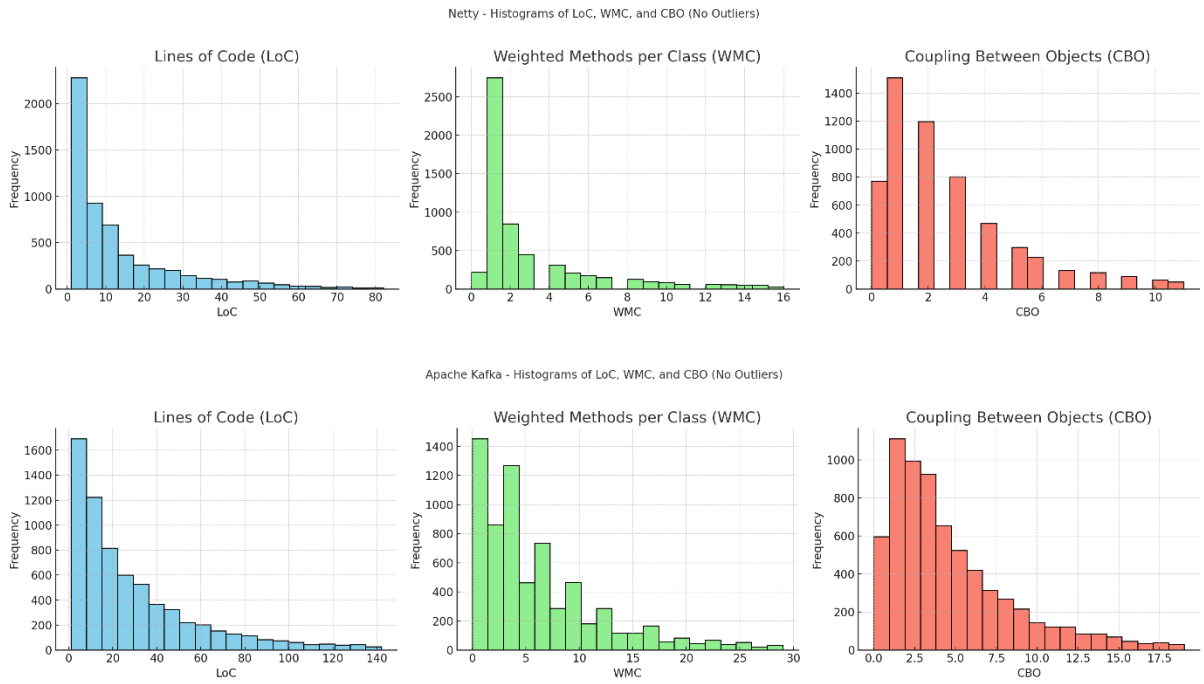

Spring Framework: Relationship between LoC and Maintainability Metrics
WMC
CBO
Metric Value
0
200
400
600
800
0
1000
2000
3000
4000
5000
Lines of Code (LoC)


Tutorials: Relationship between LoC and Maintainability Metrics
WMC
CBO
Metric Value
Lines of Code (LoC)


Tutorials by Eugen Paraschiv - Histograms of LoC, WMC, and CBO (No Outliers)
Lines of Code (LoC)
Weighted Methods per Class (WMC)
Coupling Between Objects (CBO)
Frequency
LoC
WMC
CBO


Spring Framework - Histograms of LoC, WMC, and CBO (No Outliers)
Lines of Code (LoC)
Weighted Methods per Class (WMC)
Coupling Between Objects (CBO)
Frequency
LoC
WMC
CBO


Spring Boot - Histograms of LoC, WMC, and CBO (No Outliers)
Lines of Code (LoC)
Weighted Methods per Class (WMC)
Coupling Between Objects (CBO)
Frequency
LoC
WMC
CBO



Section-5: Conclusion

Our analysis utilizing the Chidamber & Kemerer metrics on five Java projects has substantiated that maintainability is positively influenced by smaller class sizes, lower complexity, and minimal coupling. These findings highlight the critical nature of designing software with manageable, modular components to ease maintenance and enhance adaptability. While our study provides valuable insights into Java-based environments, further research could expand these observations to other programming contexts and examine additional factors such as testing protocols and documentation quality to offer a more comprehensive understanding of software maintainability.

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] Aniche, M. (2021). CK Metrics Tool. GitHub repository. https://github.com/mauricioaniche/ck