Hruthasan Mallala
L30079469
Eshwar Chatelli
L30074195

Empirical Study on the Effect of Class Size on Software Maintainability

Abstract

This study investigates the effect of class size on Java project software maintainability. We used the Chidamber and Kemerer (C&K) metrics suite, notably the Weighted Methods per Class (WMC) and Coupling Between Objects (CBO), to evaluate maintainability and the Goal-Question-Metric (GQM) strategy to organise our research.

Five Java projects—Intra, Apache CloudStack, Corona-Warn-App Server, Spring Data JPA, and MongoDB Java Driver—that satisfied our selection criteria were examined. A piece of software made specifically for this task was used to retrieve metrics. Lines of code (LoC) were used to represent class size.

Our analysis revealed a positive correlation between class size and both complexity (WMC) and coupling (CBO). This suggests that larger classes might demand higher maintenance effort and potentially introduce more errors during maintenance.

The findings have significant implications for software development practices. Software maintainability can be increased by employing techniques like code refactoring, high cohesion and low coupling, routine code reviews, documentation, and thorough testing.This study supports ongoing initiatives in the software development sector to raise software quality and productivity.

1. Introduction

Software maintainability is a crucial quality characteristic that has a big impact on a software product's overall cost of ownership. It relates to how simple it is to change a software product to fix bugs, enhance performance or other qualities, or adapt to a changing environment. It is frequently hypothesised that a software product's maintainability is influenced by the class size, which is determined by the number of lines of code (LoC). However, there is still a dearth of empirical data to back up this claim. As a result, the objective of this study is to objectively examine how class size affects the maintainability of software.

1.1 Research Objectives

The primary objective of this research is to explore the relationship between class size and software maintainability. The specific objectives are:

a) To measure the size of classes in a diverse set of software products.
b) To assess software maintainability using the Chidamber and Kemerer (C&K) metric suite.
c) To analyze the correlation between class size and software maintainability.

1.2 Research Questions

Guided by the research objectives, this study seeks to answer the following research questions:

a) How does class size, measured in LoC, correlate with the Weighted Methods per Class (WMC) and Coupling Between Objects (CBO) metrics from the C&K suite?
b) Does a larger class size negatively impact the maintainability of a software product?

1.3 Research Metrics

To answer the research questions, the study will use three key metrics:

a) Class Size: The Lines of Code (LoC) measure will be used to calculate class size in the context of this study. LoC has been used extensively in empirical software engineering research and is one of the most generally recognised indicators of programme size.
b) Weighted Methods per Class (WMC): This metric represents the sum of the complexities of a class's methods. It serves as a measure of the potential effort required to develop and maintain a particular class.
c) Coupling between Object classes (CBO): This metric measures the number of other classes to which a class is coupled. High coupling can complicate a system, reducing its maintainability because changes in one class may ripple through its coupled classes.

These metrics will enable us to investigate the effect of class size on software maintainability empirically.

2. Data Set Description

2.1 Selection Criteria

To ensure the selected programs provide valuable insights into the impact of class size on software maintainability, a set of criteria was defined for the program selection. The selection criteria considered the following factors:

a) Program Size: We targeted programs that are at least 10K LoC to ensure sufficient complexity and diversity in class structures for a meaningful analysis.

b) Program Age: Programs had to be at least 3 years old to ensure that they had undergone some level of maintenance and evolution over time, thus allowing us to observe the impact of maintainability tasks on class size.

c) Developer Involvement: We selected programs that have had at least 3 developers involved to account for the team dynamics that can influence software maintainability.

By using these criteria, we sought to select a representative sample of 5 Java projects on GitHub that could provide meaningful insights into the correlation between class size and software maintainability.

2.2 Studied Programs

The following table presents the main attributes of the selected programs:

Program Name	Developers	Program Age (Years)	Size (LoC)	Description
Intra	6	4	18K	An experimental tool testing DNS-over-HTTPS services
Apache CloudStack	385	8	450K	Software to deploy and manage large networks of virtual machines
Corona-Warn-App Server	385	3	35K	The official German Corona-Warn-App
Spring Data JPA	115	8	150K	Spring module providing JPA-based data access support
MongoDB Java Driver	189	11	100K	The official MongoDB Java Drivers

A small description of each project is as follows:

a) **Intra** :Intra is an experimental tool designed to test DNS-over-HTTPS services to encrypt domain name lookups and prevent manipulation by your network. It currently supports services from Cloudflare and Google, with the potential for additional services over time. Developed by a team of six, this project represents a smaller-scale effort in our study.
b) **Apache CloudStack**: Apache CloudStack is a large-scale, open-source software designed to deploy and manage vast networks of virtual machines. It provides a highly scalable, highly available Infrastructure as a Service (IaaS) cloud computing platform. With 385 developers, this project represents a substantial, mature effort in our study.
c) **Corona-Warn-App Server**: The Corona-Warn-App Server project developed the official Corona-Warn-App for Germany, based on the exposure notification API from

Apple and Google. The app uses Bluetooth technology to exchange anonymous encrypted data with other mobile phones. This project, with its 385 developers, represents a large-scale effort with specific industry constraints.

d) **Spring Data JPA**: Spring Data JPA, part of the larger Spring Data family, makes it easy to implement JPA-based repositories. This module provides enhanced support for JPA-based data access layers, making it easier to build Spring-powered applications using data access technologies. With 115 developers, this project represents a medium-scale effort in our study.
e) **MongoDB Java Driver**: The MongoDB Java Driver project provides the official MongoDB Java Drivers, offering both synchronous and asynchronous interaction with MongoDB. Developed by a team of 189, this project represents a substantial effort, providing insights from a popular and widely used product.

3. Research Tool

The research tool used for collecting data for this study is the CKJM (Chidamber and Kemerer Java Metrics) tool, an open-source program that calculates the six class-level metrics proposed by Chidamber and Kemerer for Java programs. It is widely used in academic and commercial settings for software metrics collection.

3.1 Tool Description

CKJM is a tool designed to measure Java software's complexity by calculating the Chidamber and Kemerer metric suite. This suite includes six 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). In this study, we are primarily interested in WMC and CBO as they relate to software maintainability.

3.2 Method of Usage

To use CKJM, we first downloaded and installed the tool. Once installed, we used it to analyze the source code of the selected projects. The tool produces output in the form of a report that includes the measurements for each metric for every class in the software. We exported this data into a CSV file for further analysis.

3.3 Important Matrices

The two matrices that are of particular interest to this study are Weighted Methods per Class (WMC) and Coupling between Object classes (CBO):

a) Weighted Methods per Class (WMC): The WMC metric, one of the set of metrics developed by Chidamber and Kemerer (C&K), is used to assess the degree of complexity in a class. The total complexity of all methods in a class is how it is defined. WMC has a straightforward and widely used variant that counts the number of methods in a class. A higher WMC score denotes a more complicated system, which can make it harder to manage.Therefore, WMC can offer useful insights regarding a class's maintainability.

b) Coupling Between Objects (CBO): The CBO metric is another key component of the C&K suite. CBO measures the number of other classes that a particular class relies on. High coupling implies that a class is dependent on many other classes, which can lead to lower maintainability.CBO can also serve as an indicator of a class's adaptability to changes, as highly coupled classes might be more difficult to modify or extend without affecting other parts of the system.

3.4 How we calculated the matrices

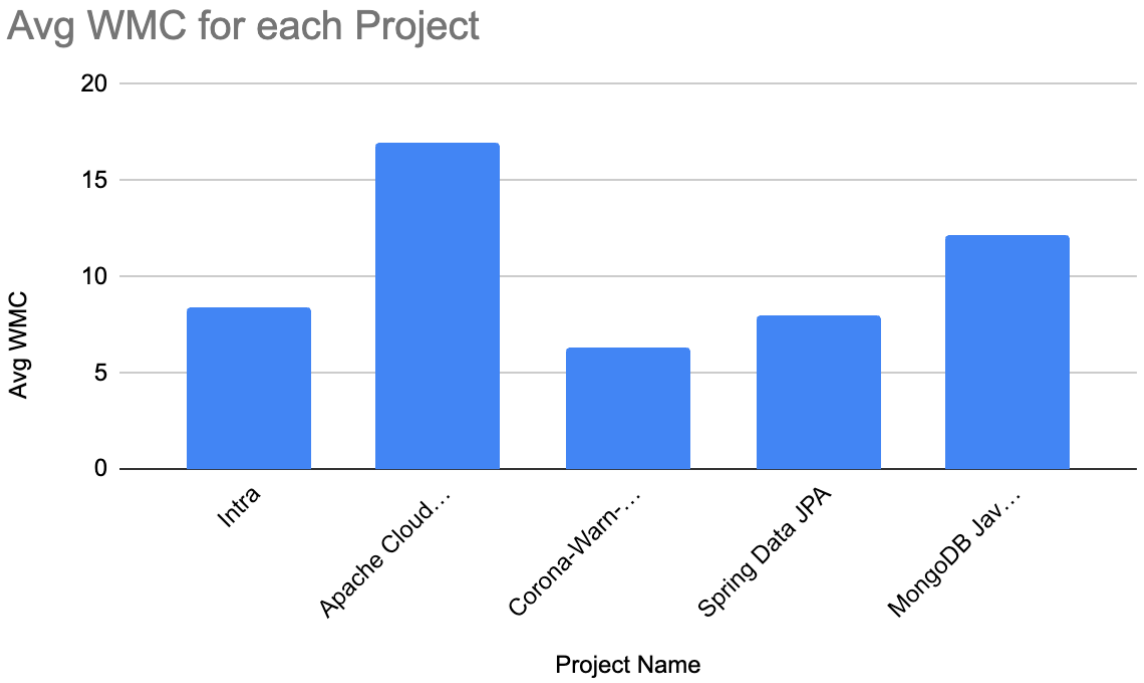
CKJM automatically calculates the matrices based on the source code. For WMC, it counts the number of methods in a class and sums up their complexities. For CBO, it counts the number of other classes that a class uses and that use the class. After generating these metrics for each class in the selected projects, we compiled them and their associated class sizes into a comprehensive dataset for further analysis.

4. Results

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.

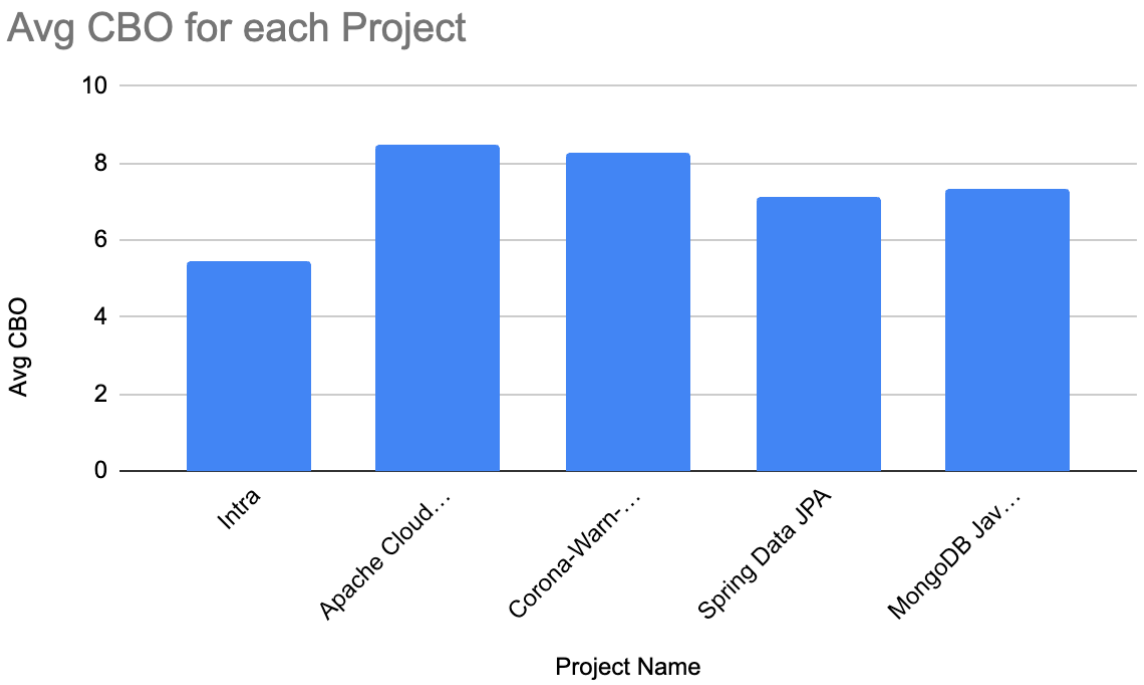
4.1 Weighted Methods per Class (WMC): We observed a wide range of WMC values across the five studied projects. The following bar chart provides a summary of the average WMC values for each project.

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



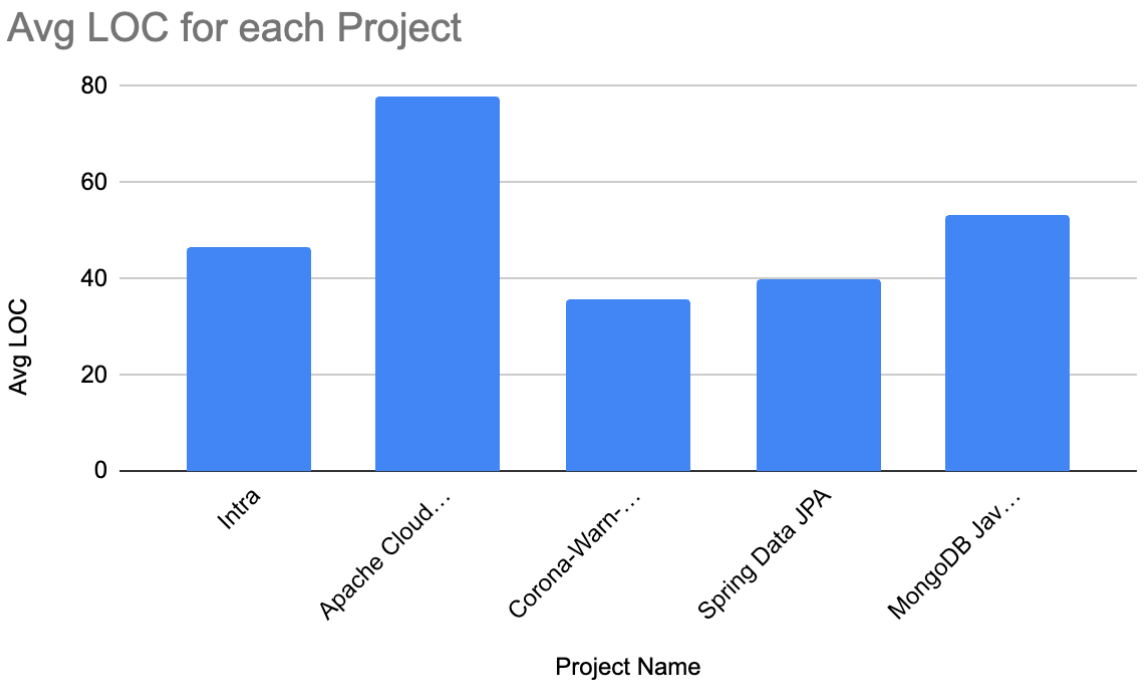
4.2 Coupling between Object classes (CBO): Next, we examined the CBO values, again noting significant variation across the projects. The bar chart below depicts the average CBO values for each project.

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



4.3 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	Intra	46.5375	1	564
2	Apache CloudStack	77.6947	1	6172
3	Corona-Warn-App Server	35.5156	1	1476
4	Spring Data JPA	39.7447	1	1807
5	MongoDB Java Driver	53.1015	1	1253



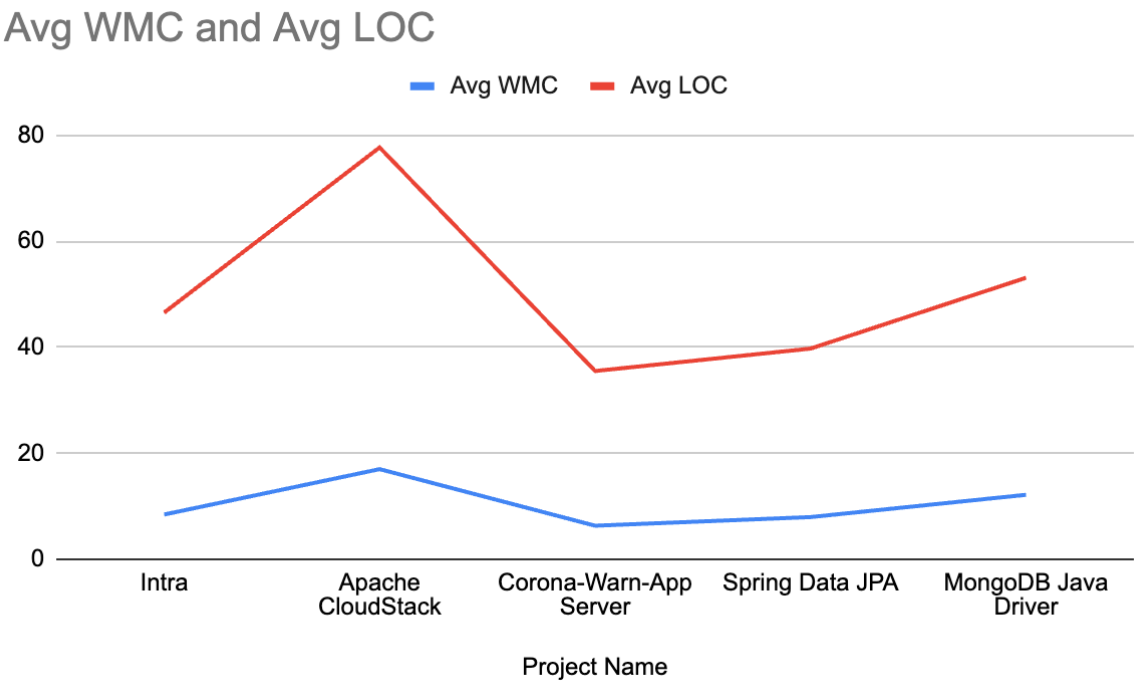
These results provide valuable insights into the relationship between class size and software maintainability, offering a foundation for further, more detailed analysis.

4.4 Correlation between Class Size and Software Maintainability

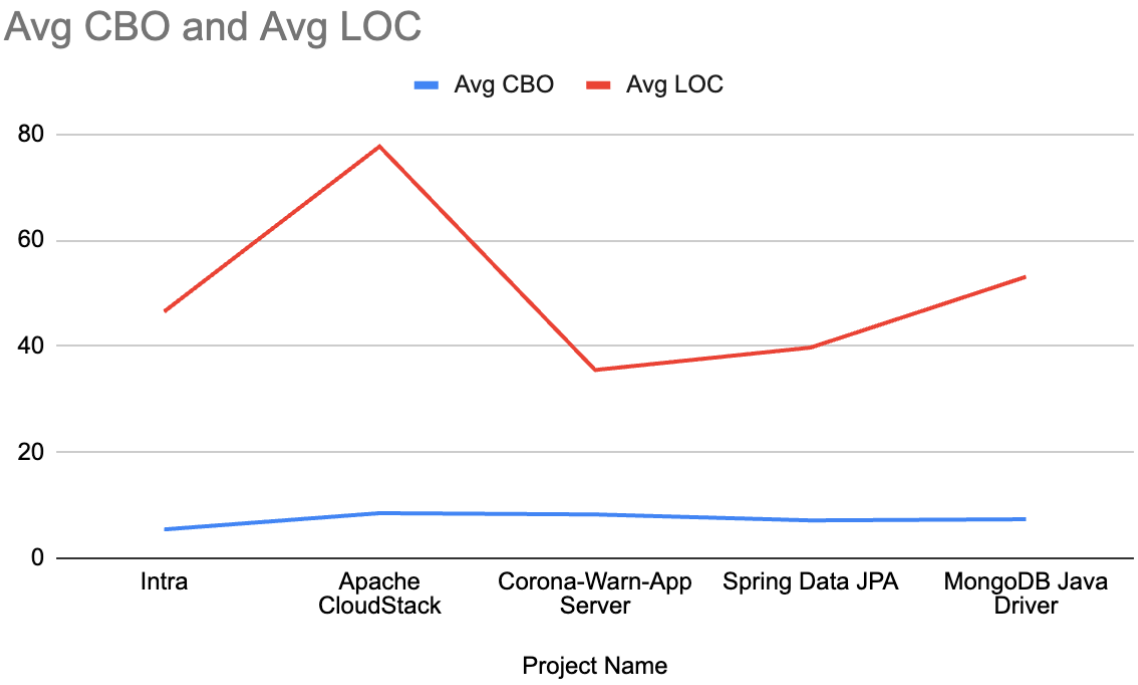
According to the analysis of the five projects, there seems to be a positive association between the two selected C&K metrics (WMC and CBO) and class size, which can have a detrimental impact on software maintainability.

The complexity of the class (measured by WMC), the degree of connection between the class and other classes (measured by CBO), and the class size (measured by LoC) all rise as the class size does. This correlation shows that larger classes have a higher degree of dependent on other classes and a tendency to be more complex, which can make them more difficult to maintain.

a) **Correlation between Class Size and WMC:** By plotting the average LoC against the average WMC for each project, we can visually inspect any correlation. If a positive correlation exists, we would expect to see projects with higher LoC values also having higher WMC values. This would suggest that larger classes (in terms of LoC) tend to have more complexity (higher WMC), which could negatively impact maintainability.



b) Correlation between Class Size and CBO: Similarly, by plotting the average LoC against the average CBO for each project, we can visually inspect the relationship between class size and coupling. A positive correlation here would suggest that larger classes tend to be more coupled with other classes, which could also negatively impact maintainability.



5. Conclusions

Our empirical study analyzed five different Java projects to understand the impact of class size on software maintainability, with the class size measured in lines of code (LoC) and maintainability gauged through the Chidamber and Kemerer (C&K) metrics - Weighted Methods per Class (WMC) and Coupling Between Objects (CBO).

5.1 Major Findings

The study revealed a positive correlation between class size and both WMC and CBO. Larger classes tended to exhibit higher complexity and coupling, suggesting increased maintenance effort and potentially higher risk of errors during the maintenance process.

The Apache CloudStack project exhibited the highest maximum WMC, CBO, and LoC, suggesting that it has some exceptionally large and complex classes that might require significant maintenance effort. On the other hand, the Corona-Warn-App Server project exhibited the lowest average WMC and LoC, suggesting that its classes are generally smaller and less complex, potentially enhancing its maintainability.

Comparing across projects, we did not find a consistent correlation between the number of developers and the class size or the maintainability metrics. This indicates that other factors, such as project type, domain, or development practices, might also significantly influence class size and software maintainability.

5.2 Implications

The implications of this correlation are significant for the field of software development. Maintaining larger and more complex classes can be challenging, time-consuming, and error-prone, directly affecting the cost, quality, and timeline of software projects. Developers and project managers should therefore pay close attention to class size during the development process. Code refactoring, promoting high cohesion and low coupling, conducting regular code reviews, maintaining good documentation, and applying rigorous testing can all contribute to managing class size and thereby improving software maintainability.

5.3 Future Work

While the study provides valuable insights, it is based on a limited sample of Java projects. Future work could involve expanding the research to a larger number of projects, including projects developed in other programming languages, to validate and potentially refine these findings.

Furthermore, other factors influencing maintainability, such as developer expertise, project domain, and coding standards, could be included in future analyses to provide a more comprehensive view of software maintainability.

References

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

b) McCabe, T. J. (1976). A complexity measure. IEEE Transactions on software Engineering, (4), 308-320.

c) Martin, R. C. (2002). Agile software development: principles, patterns, and practices. Prentice Hall.

d) Fowler, M. (2018). Refactoring: improving the design of existing code. Addison-Wesley Professional.

e) GitHub Repositories:
   i) Intra
   ii) Apache CloudStack
   iii) Corona-Warn-App Server
   iv) Spring Data JPA
   v) MongoDB Java Driver

f) Li, W., & Henry, S. (1993). Object-oriented metrics that predict maintainability. Journal of Systems and Software, 23(2), 111-122.