Assignment 2

Akhila Parvathaneni, Bhanuprakash Banda, and Harika Mareedu

Lewis University

CPSC-60000; Object Oriented Development

Dr. Ziad Al-Sharif

06/16/2024

Table of Contents

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

Objective:

To investigate the impact of code smells on the modularity of software.

Questions:

- How does a bad code smell affect the coupling of classes in Java projects?
- How does a bad code smell affect the cohesion of classes in Java projects?
- What trends can be observed in modularity metrics for classes with bad smells compared to those without?

Metrics:

1. Lack of Cohesion in Methods (LCOM): LCOM quantifies the absence of coherence among the methods within a class. More LCOM values indicate inferior organization and potential difficulties in maintaining the code, as classes with more cohesion are typically more understandable and adaptable.
2. Coupling Between Objects (CBO): The CBO metric quantifies the degree of interdependence between a class and other classes within the system. More significant interdependencies are shown by higher CBO values. This can lead to more complexity and less maintainability since changes in one class may require changes in related classes.

Section 2: Description of Subject Programs (Dataset)

Criteria for Selection

- **Lines of Code (LOC):** Projects should have at least 10,000 lines of code.
- **Age:** Projects should be at least 3 years old.
- **Number of Developers:** Projects should have at least 3 developers.

Justification:

Lines of Code (LOC): Projects with at least 10,000 lines of code are likely to possess sufficient complexity and variety to exhibit various bad smells and modularity issues. Smaller projects may not provide enough data to enable a meaningful analysis of these factors. Therefore, selecting projects with a substantial amount of code ensures that the study can effectively identify and evaluate the presence and impact of bad smells.

Age: Projects that are at least 3 years old are more suitable for this study because they are likely to have gone through multiple phases of development and maintenance. These phases can introduce and evolve bad smells over time. Analyzing older projects allows for the examination of the long-term effects of bad smells on modularity, providing insights into how these issues develop and persist in mature codebases.

Number of Developers: Projects with at least 3 developers are more likely to involve collaborative development practices. Such collaboration often results in varied coding styles and a higher potential for the introduction of bad smells. This diversity in development practices creates a richer dataset for analysis, enabling a comprehensive study of the correlation between bad smells and modularity across different development environments.

Table:

Project Name	**LOC (Lines of Code)**	**Age (Years)**	**Number of Developers**	**Description**
Apache Commons Lang	81,000	20	124	Helper utilities for the java.lang API.
JUnit5	68,000	6	90	Testing framework for Java.
Spring Framework	2,000,000	17	1,270	Infrastructure support for Java applications.
Apache Hadoop	3,000,000	15	2,300	Framework for distributed processing of large data sets.
Java Design Patterns	28,000	8	268	Design patterns implemented in Java
Google Guava	200,000	13	275	Core libraries with new collection types and utilities.
Apache Tomcat	1,300,000	24	750	Java Servlet, JSP, EL, and WebSocket implementation.
Dubbo	39,000	10	393	Apache Dubbo is a high-performance, java based, open-source RPC framework.
RxJava	67,000	10	328	Library for composing asynchronous programs.

Apache Cassandra	1,400,000	14	1,100	Highly scalable, distributed NoSQL database.

Section 3: Tool Description

CK Metrics Tool:

The CK Metrics tool is a widely used static code analysis tool that calculates various metrics to assess the quality and maintainability of Java code. Developed by Mauricio Aniche, this tool specifically measures several key object-oriented metrics, including Coupling Between Objects (CBO) and Lack of Cohesion in Methods (LCOM), among others. These metrics provide insights into the modularity and complexity of the software. The CK Metrics tool was chosen for this study because it provides comprehensive and relevant metrics, such as CBO and LCOM, which are essential for analyzing the modularity of software. Its ease of use and robust reporting capabilities make it an ideal choice for this empirical study.

JDeodorant:

JDeodorant is an Eclipse plug-in designed to detect code bad smells in Java applications. Developed by Theodoros Tsantalis, it identifies common bad smells such as God Class, Long Method, and Feature Envy, and suggests possible refactoring solutions. JDeodorant's detection algorithms are based on well-established research in the field of software engineering. JDeodorant was selected for this study due to its effectiveness in identifying a wide range of code bad smells, which are critical for our analysis. Its integration with Eclipse and user-friendly interface make it accessible for both novice and experienced developers, ensuring that bad smells are detected accurately and efficiently.

Section 4: Results

Project	Avg CBO	Avg LCOM	With Smell	Without Smell
apache-cassandra	6.630418	56.188151	6364	4202
apache-hadoop	6.836977	54.394456	8611	6577
apache-tomcat	4.485937	151.157508	2424	2198
Apahce-commons-lang	3.120930	273.753488	518	557
dubbo	5.177738	41.552289	2557	1899
google-guava	3.872595	47.358163	2488	3956
java-design-patterns	3.374439	1.904709	651	1133
Junit-5	4.365096	37.137295	1399	1529
rxjava	2.524062	47.416603	2485	9380
spring-framework	5.249460	30.516154	8664	8453

Table 1: Average Coupling and Cohesion Metrics for Projects with and without Code Bad Smells

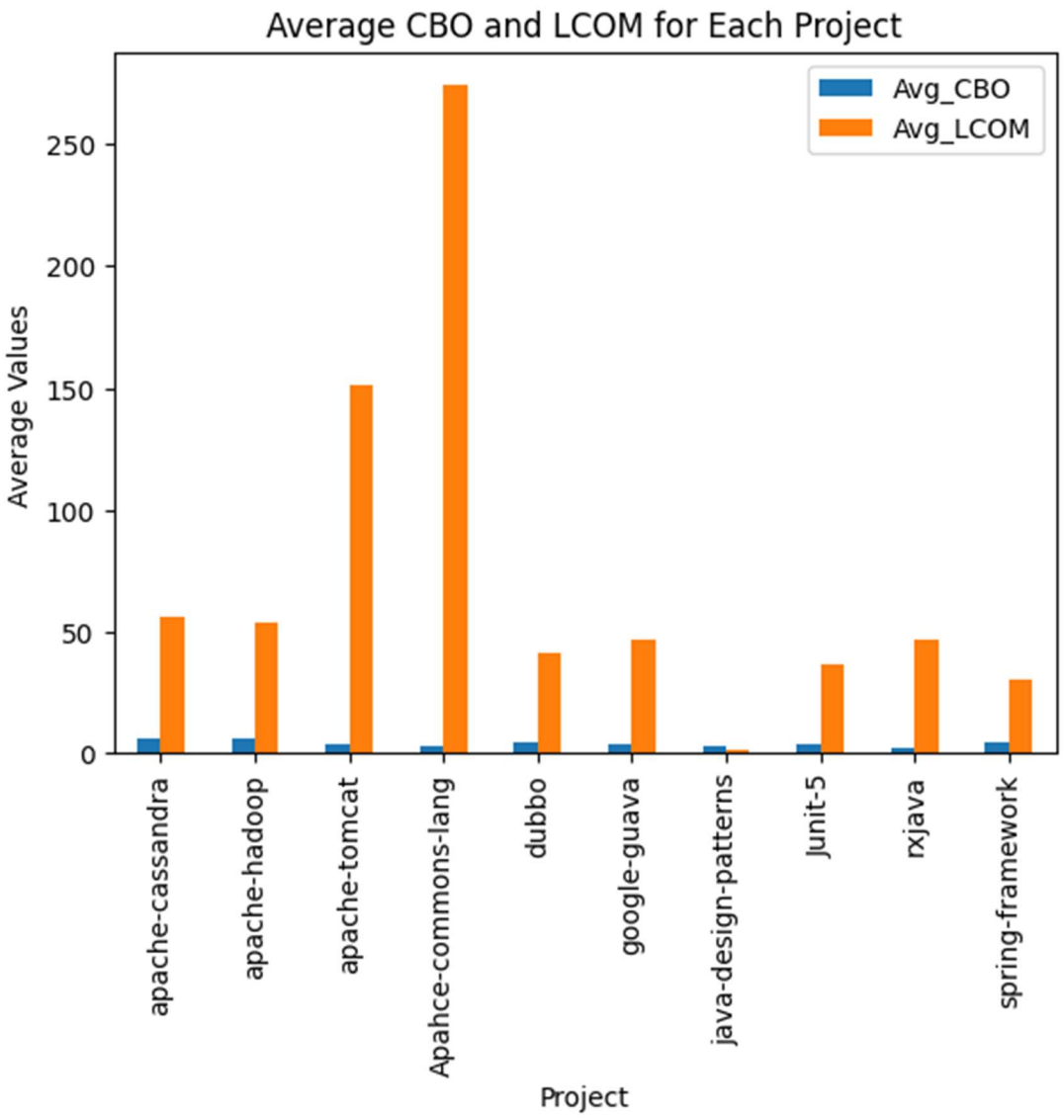

Fig 1: Average Coupling (CBO) and Cohesion (LCOM) for Each Project

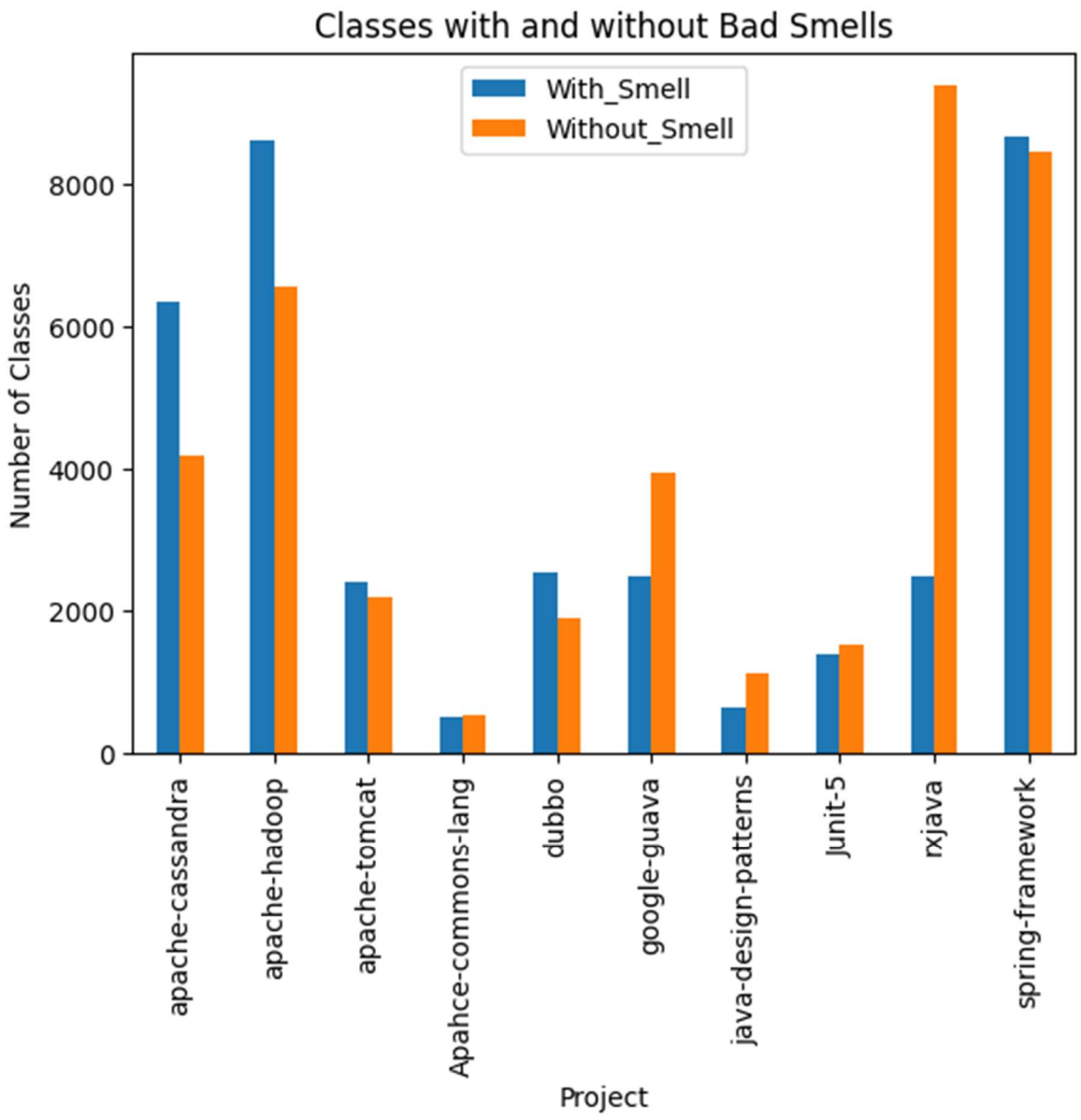

Fig 2: Number of Classes with and without Code Bad Smells in Each Project

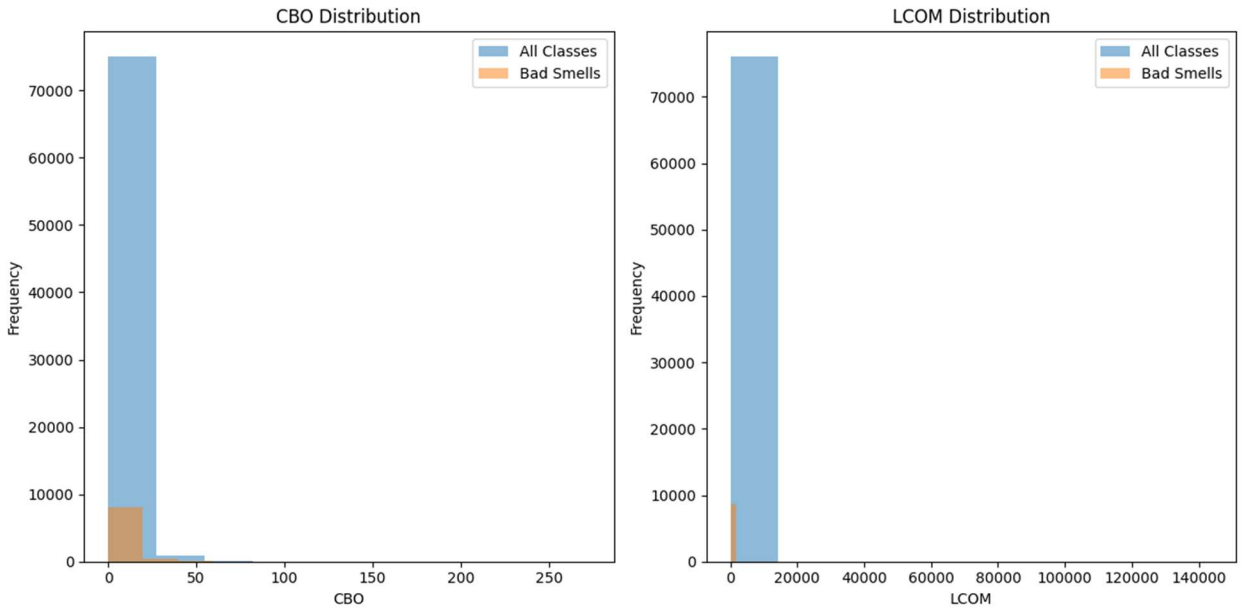

Fig 3: Distribution of CBO and LCOM Metrics for Classes with and without Code Bad Smells

Key Observations:

CBO (Coupling Between Objects):

- Projects like **apache-hadoop** and **apache-cassandra** have higher average CBO values, indicating a higher level of coupling between classes when code smells are present.
- For some projects like **google-guava**, the average CBO is lower with code smells, which might indicate specific architectural or coding practices that mitigate the impact of smells on coupling.

LCOM (Lack of Cohesion in Methods):

- Projects such as **apache-tomcat** and **apache-commons-lang** show extremely high LCOM values with code smells, suggesting that methods in classes are not well-related, thereby reducing modularity.

- Interestingly, **rxjava** has a significantly lower LCOM without smells, indicating better method cohesion in the absence of code smells.

Section 5: Conclusion

The analysis of the effect of code smells on modularity in various software projects reveals significant insights into the impact of poor coding practices on system design and maintenance. The study examined two key metrics: Coupling Between Objects (CBO) and Lack of Cohesion in Methods (LCOM), which serve as indicators of modularity and the presence of code smells in the projects. The findings suggest that code smells have a detrimental effect on modularity, influencing both the coupling and cohesion of the codebase.

Firstly, the CBO values highlight the extent of coupling between different classes within the projects. High CBO values are indicative of tightly coupled systems where classes are highly dependent on one another. Projects such as apache-hadoop and apache-cassandra exhibit high average CBO values, especially when code smells are present. This suggests that code smells exacerbate the coupling between classes, making the code more interconnected and harder to maintain. On the contrary, projects like google-guava show lower CBO values even with the presence of code smells, which could be attributed to specific architectural or coding practices that help in mitigating the impact of smells on coupling.

Secondly, the LCOM values shed light on the cohesion within the classes. High LCOM values imply that the methods within a class are not well-related, reducing the overall modularity of the system. The analysis reveals that projects such as apache-tomcat and apache-commons-lang have extremely high LCOM values when code smells are present, indicating a significant reduction in cohesion and, consequently, modularity. In contrast, rxjava shows a markedly lower LCOM in the absence of code smells, pointing to better method cohesion and improved modularity when code smells are minimized.

The presence of code smells in a codebase tends to correlate with higher coupling and lower cohesion, leading to reduced modularity. This reduction in modularity can have several adverse effects on software projects. High coupling makes the system more brittle and less adaptable to changes, as modifications in one class may require changes in several other dependent classes. This interdependency complicates maintenance and increases the risk of introducing bugs. Furthermore, low cohesion within classes means that the functionality of the class is not well-defined, making it harder to understand, test, and reuse.

Moreover, the varied impact of code smells across different projects underscores the importance of context and the specific coding practices employed in each project. Some projects may adopt architectural patterns or coding standards that mitigate the adverse effects of code smells, leading to better modularity despite their presence. For instance, the lower CBO values in google-guava suggest that certain practices can help maintain modularity even when code smells exist.

In conclusion, the effect of code smells on modularity is evident across the studied projects. Higher coupling and lower cohesion are significant indicators of the negative impact of code smells, leading to less modular and more difficult-to-maintain codebases. Addressing code smells through refactoring and adhering to good coding practices is essential for improving modularity and ensuring the long-term maintainability and flexibility of software systems. The study highlights the need for continuous monitoring and management of code quality to mitigate the detrimental effects of code smells on modularity.

References

1. Aniche, M. (n.d.). CK: Chidamber and Kemerer Java Metrics. GitHub repository. Retrieved from https://github.com/mauricioaniche/ck
2. Tsantalis, T. (n.d.). JDeodorant: Identification and Refactoring of Code Smells. GitHub repository. Retrieved from https://github.com/tsantalis/JDeodorant
3. Iluwatar, E. (n.d.). GitHub repository. Retrieved from https://github.com/iluwatar/java-design-patterns
4. Apache. (n.d.). GitHub repository. Retrieved from https://github.com/apache/dubbo
5. Apache Commons Lang. (n.d.). GitHub repository. Retrieved from https://github.com/apache/commons-lang
6. JUnit5. (n.d.). GitHub repository. Retrieved from https://github.com/junit-team/junit5
7. Spring Framework. (n.d.). GitHub repository. Retrieved from https://github.com/spring-projects/spring-framework
8. Apache Hadoop. (n.d.). GitHub repository. Retrieved from https://github.com/apache/hadoop
9. Google Guava. (n.d.). GitHub repository. Retrieved from https://github.com/google/guava
10. Apache Tomcat. (n.d.). GitHub repository. Retrieved from https://github.com/apache/tomcat
11. RxJava. (n.d.). GitHub repository. Retrieved from https://github.com/ReactiveX/RxJava
12. Apache Cassandra. (n.d.). GitHub repository. Retrieved from https://github.com/apache/cassandra