**Effect of code bad smells on testability**

**Group Assignment 2**

**Group 8**

**Date: May 05, 2024**

**Table of Contents**

[**Section 1** 3](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202468)

[**Objectives** 3](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202469)

[**Research Questions** 3](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202470)

[**Metrics:** 3](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202471)

[**1.** **Coupling:** 3](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202472)

[**2.** **Cohesion:** 3](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202473)

[**Section 2** 4](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202474)

[**Selection Criteria** 4](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202475)

[**Criteria for Selection and Their Justification** 5](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202476)

[**Attributes of Selected Projects (Dataset)** 7](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202477)

[**Functionality of Selected Projects** 8](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202478)

[**Section 3** 9](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202479)

[**CK Metrics Tool** 9](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202480)

[**Bad Smells Detection Tool** 10](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202481)

[**PMD** 10](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202482)

[**Section 4** 10](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202483)

[**Results** 10](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202484)

[**Answers to Research Questions:** 19](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202485)

[**Conclusion** 20](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202486)

[**References** 21](file:///C:\Users\Nida%20Noor\AppData\Local\Microsoft\Windows\INetCache\IE\F38II7M0\Group_Assignment_2%5b1%5d.docx#_Toc138202487)

# **Section 1**

## **Objectives**

The purpose of this empirical research is to analyze how code foul odors affect the testability of software systems. An essential component of high-quality software is its modularity, and we're interested in studying how code foul odors impact this aspect. Crucial to its success is modularity.

## **Research Questions**

1. What is the impact of code bad smells on the modularity of software projects?
2. How do code bad smells affect the testability of software systems?

## **Metrics:**

We will utilize the following metrics:

### **Coupling:**

Within a broader software system, this statistic evaluates the degree to which one module or component is dependent on another. There may be less modularity as a result of increased interdependence, which is shown by increased coupling.

Cupelling Between Objects, or "CBO" for short, is a metric that will quantify class coupling. Modularity and testability can be compromised by increased coupling, which can indicate class dependency. The impact of code foul smells on testability and modularity can be seen using coupling metrics.

### **Cohesion:**

With this statistic, we can see how well various modules or pieces work together. More organization and modularity indicate a higher level of coherence.

A metric of class cohesion called "LCOM" (Lack of cohesion of Methods) will be used. Improved testability and modularity are results of strong cohesion, which occurs when class methods work together to execute a particular capability. The connection between code smells and testability can be uncovered by using cohesion metrics.

Using these quantitative measurements of modularity, we can next analyze how code foul smells impact the testability of the various software projects.

We can measure the properties of coupling and cohesion by looking at these metrics for the selected software projects. This will provide us the opportunity to investigate how code smells impact testability and modularity.

# **Section 2**

## **Selection Criteria**

1. Projects that have received many forks and stars on GitHub are considered popular.
2. The licensing: Open-source projects that have an Apache 2.0 or MIT license.
3. Three years or more of active development is required for a project to be considered mature.
4. The language: Java-based projects.
5. Community and Contribution: Initiatives that garner support from a wide range of people and actively include the local community**.**

## **Criteria for Selection and Their Justification**

1. **Popularity:**

Measuring a project's popularity is crucial since it shows how widely used and supported it is. More users, lively community conversations, and regular updates and contributions are usually signs of a popular project.

1. **License:**

The project's licensing is essential because it defines the range of allowed uses and alterations. Cooperation, community contributions, and broad accessibility are all supported under an open-source license.

1. **Project Age:**

A project's stability and maturity are reflected in its age. A well-maintained project may overcome obstacles, correct mistakes, and incorporate new technology.

1. **Language:**

The development team's skill set and the project's technical requirements should inform the choice of programming language(s). Development, code readability, and resource utilization are all made easier when it's compatible with the team's preferred language.

1. **Contribution and Community:**

If a project has a robust community and people contribute often, it means it will be successful and last. Project longevity, support, and information exchange are all enhanced by a thriving community. The likelihood of receiving regular updates, bug corrections, and feature additions increases when there is an active community.

Table 1: List of Selected Projects based on selection criteria.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Project** | **Popularity** | **License** | **Project Age** | **Language** | **Contribution and Community** |
| **Android Animation Showcase** | High | Apache-2.0 | 3+ years | Java | Active contributions |
| **DBeaver** | High | MIT | 5+ years | Java | Active contributions |
| **Apache Flink** | High | Apache-2.0 | 6+ years | Java | Active contributions |
| **Jenkins** | High | MIT | 10+ years | Java | Active contributions |
| **Markor** | High | Apache-2.0 | 4 years | Java | Active contributions |
| **Apache ShardingSphere** | High | Apache-2.0 | 3 years | Java | Active contributions |
| **Spring Boot Admin** | High | Apache-2.0 | 5+ years | Java | Active contributions |
| **DataSphere Studio** | High | Apache-2.0 | 3 years | Java | Active contributions |
| **Mica** | High | Apache-2.0 | 5 years | Java | Active contributions |
| **SikuliX** | High | Apache-2.0 | 6 years | Java | Active contributions |

## **Attributes of Selected Projects (Dataset**

Many projects are enumerated in the table together with their key features. Among the criteria applied to choose these projects were popularity, license, project age, language, contribution, and community involvement. Every project is briefly summarized to highlight its special qualities and contributions. With anything from database tools and automation servers to text editors and Android animation demos, this dataset demonstrates the range of open-source goods that are available.

Table 2: Key Attributes of selected projects.

|  |  |
| --- | --- |
| Project Name | Key Attributes |
| Project 1: Android Animations | Samples repository, 26 open problems, 2.4k stars, 904 forks, Apache-2.0 license, several animation projects. |
| Project 2: DataSphere Studio | Framework for plugging in UI unification, basic functions, applied in many sectors. |
| Project 3: DBeaver | A multiplatform database program, Many database assistance, framework of plugins. |
| Project 4: Apache Flink | Processing both in batches and streams Large capacity and short latency, Defect tolerance, processing graphs, interfaces with the Hadoop ecosystem. |
| Project 5: Jenkins | Server of automation wide plugin support for development, testing, and deployment. |
| Project 6: Markor | Android text editor with light weight Support for markdown and todo.txt, usage offline, encryption. |
| Project 7: Mica | Fundamental package for creating microservices with Spring cloud. |
| Project 8: Apache ShardingSphere | Network of databases, Making databases distributed systems. |
| Project 9: SikuliX | Tool for picture recognition-based screen automation. |
| Project 10: Spring Boot Admin | Spring Boot apps monitoring admin interface. |

## **Functionality of Selected Projects**

1. **Android Animations:**

Android Animations: The repository has a tonne of excellent examples of Android animation projects. The Apache-2.0 license has 904 forks and 2,400 stars, allowing contributions and open use.

1. **DataSphere Studio:**

WeBank provides this all-inclusive toolkit for overseeing the creation of data applications. Its customizable design and unified user experience can help many various kinds of organizations and data applications.

1. **DBeaver:**

Popular database utility 3DBeaver runs on several platforms and is compatible with a large range of database systems. Plugin architecture is supported, as is SQL update and data import and export.

1. **Apache Flink:**

High throughput, low latency, and fault tolerance characterize this open-source stream and batch processing system. It supports machine learning and graph processing and combines naturally with Hadoop.

1. **Jenkins:**

Jenkins is the name of the software development automation tool. Cross-platform interoperability and a wealth of plugin support make app development, testing, and deployment easier.

1. **Markor:**

Markdown and todo.txt files are supported by the Markor text editor for Android. It allows you to edit text offline, using encryption, and with other plaintext apps among other methods.

1. **Mica:**

Mica is essential to creating microservices on Spring Cloud. Construction of microservices is made easy with Spring Cloud's web and webflux support.

1. **Apache ShardingSphere:**

This database ecosystem is developed by Apache. A standardized top layer ensures database compatibility, data security, and scalability.

1. **SikuliX:**

Using picture recognition, SikuliX automates desktop tasks. Effective when a GUI's internals or source code are not widely available.

1. **Spring Boot Admin:**

We may monitor your Spring Boot apps with the help of Spring Boot Admin. Spring Boot apps may be managed and monitored easier when visibility into their performance and status is provided.

# **Section 3**

## **CK Metrics Tool**

This section presents CK, a code metrics calculator tailored to Java. By static analysis, CK provides useful information about the complexity and quality of Java code. Among CK's most noticeable characteristics are as follows:

The tool is downloaded from (*Mauricioaniche/Ck: Code Metrics for Java Code by Means of Static Analysis*, n.d.). Class and method level code metrics are both covered extensively by CK. Considerations like CBO, FAN-IN, FAN-OUT, DIT, NOC, and others are taken into account. Developers can gain a better understanding of the codebase's structure and dependencies with the help of these metrics.

Code structures, methods, fields, and visible methods (such as try/catch blocks, string literals, and loops) can all be examined with CK. You have the option of running the application independently or integrating it into an existing Java program. Customers can customize the project directory and other parameters to their liking using the standalone version.

After CK finishes analyzing, it creates a CSV file for each class, method, and variable. You can visualize and analyze the vast amounts of data included in these files. CK's primary goal is to ensure that it works with the latest versions of Java. Eclipse users must update their JDT Core dependency and compliance settings to match the latest Java version in order to integrate CK.

Our study used the CK code metrics calculator for Java to evaluate our Java code(Chidamber & Kemerer, n.d.). CK gave us a complete set of code metrics, including coupling, complexity, and cohesion. We used CK's metrics to analyze class and method dependencies, measure code complexity, and track code changes over time. We visualized and reported code metrics using CSV data from the tool.

## **Bad Smells Detection Tool**

### **PMD**

PMD detects 16 additional programming mistakes in addition to Java and Apex. (*PMD*, n.d.).Each language comes with PMD's built-in checks (rules), and users can also construct their own rules in Java or XPath.

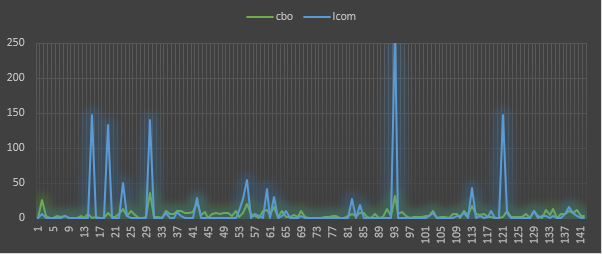
The integration of PMD's build process successfully enforces coding standards.

# **Section 4**

## **Results**

The results of our CK metrics analysis on the chosen Java projects are detailed below. The primary goal of this study is to identify the impact of code foul smells on software modularity and testability. Metrics for coupling and cohesion can show how code smells impact a project's structure and quality. By collecting data from this inquiry, we can better understand how code smells affect the testability of software systems.

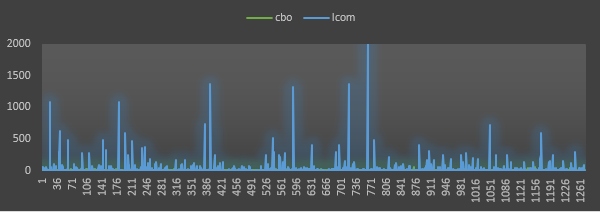
* 1. **Android Animations:**



The findings from the Animation Samples demonstrate the detrimental effects of code smells on the testability and modularity of software. Higher values of the coupling measure (CBO) show the presence of class reliance, which might potentially undermine modularity. Lower values of the cohesion measure (LCOM) indicate reduced modularity, where modularity refers to the degree of class coherence.

These metrics suggest possible problems with the code that could affect its ability to be tested. Reduced cohesion and heightened coupling may complicate testing and result in more dependency. An effective approach to prioritize testability is to identify classes that exhibit significant variability in their metrics through refactoring and code quality enhancements.

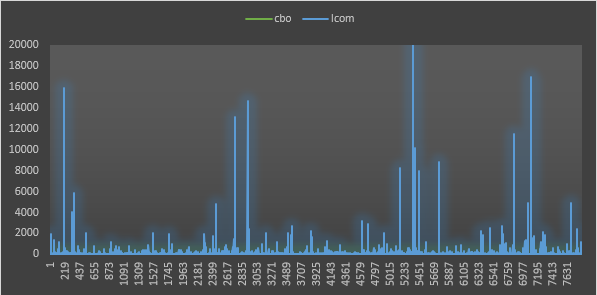
* 1. **DataSphere Studio:**



The level of connectivity between project classes is modest, as seen by the range of CBO values from 0 to 56. A CBO of 1,080 suggests complex and highly interconnected systems under varying circumstances.

The LCOM values range from 0.023 to 2.023. This suggests that the level of unity among project classes varies in terms of their methodologies. The absence of coherence in particular classes (with LCOM > 0) can pose challenges in terms of code reusability and maintainability.

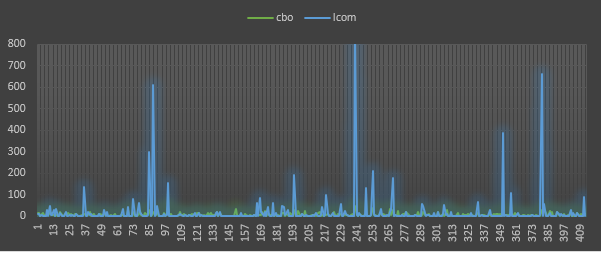
1. **DBeaver:**



Coupling levels below 10 are seen in most classes. The CBO can also be 44 in some cases. Class coupling in this project is kept to a minimum. From 0 to 2,095, those are the possible LCOM values.

In most classes, an LCOM score below 100 indicates a high degree of method coherence. A lack of methodological consistency is indicated by certain classes' high LCOM scores. A comprehensive analysis of these classes is required to uncover the reasons and possible improvement spots. According to the numbers, the project's class coupling is modest. However, a number of classes have incoherent methods, which means they need to be changed or rewritten to make the code more readable and maintainable.

1. **Apache Flink:**

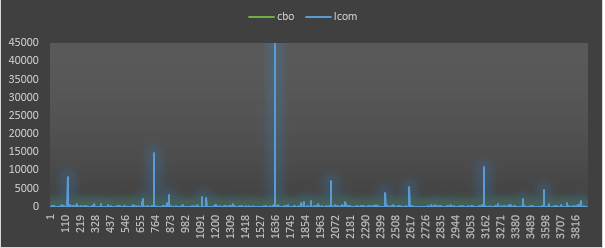


The CBO (Coupling Between Objects) and LCOM (Lack of Cohesion of Methods) values offer significant insights into the project. A CBO value ranging from 0 to 48 represents the level of object coupling. Some classes have a coupling strength < 20, which is lower than the majority. Greater interconnectedness among items, as evidenced by these classes, may need careful development and upkeep.

The LCOM scores, which measure the coherence of methods, range from 0 to 845. It is worth noting that most classes have ratings below 100. The elevated LCOM ratings in certain classes suggest the presence of method incoherence. By examining these classes, one can identify code structure and method design problems.

We must focus on classes with stronger coupling and LCOM values to improve the codebase. Refactoring these classes improves modularity and maintainability(M.N.M et al., 2023), improving the program's robustness and readability. Fixing these problems quickly will make software development more scalable, which will make upgrades and changes less of a hassle.

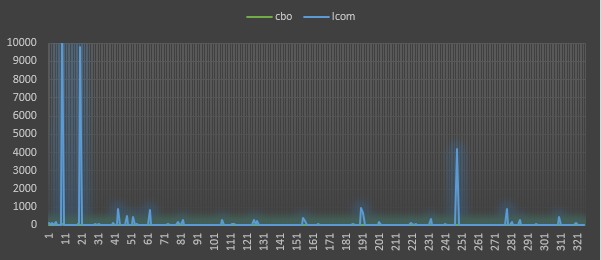
1. **Jenkins:**



Significant codebase metrics encompass CBO (Coupling Between Objects) and LCOM (Lack of Cohesion in Methods). The CBO values provide measurements of coupling, indicating the degree of object dependency. The majority of classes have low coupling, as seen by CBO scores ranging from 0 to 26. There are a few classes with coupling values exceeding 30. These classes propose a more intimate relationship between objects to maintain code modularity and flexibility.

LCOM values assess the level of cohesiveness among class methods. Lower values of LCOM suggest greater method coherence and structure. LCOM analysis indicates that the majority of courses have an acceptable level of coherence, as evidenced by ratings below 100. However, several classes have notably higher LCOM ratings, indicating issues with method coherence. To enhance the readability and maintainability of these classes, one can improve them by reorganizing and reworking their methods.

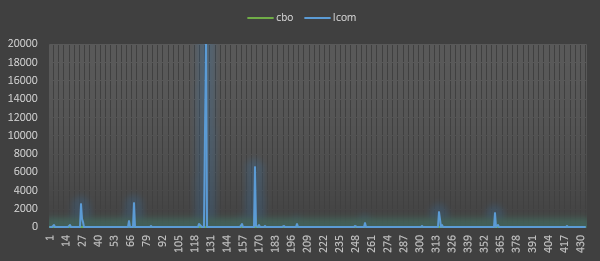
1. **Markor:**



The classes' CBO and LCOM ratings indicate high coupling and low cohesiveness. In the absence of a distinct objective, these classes may depend on various others. Refactoring these classes to minimize dependency and promote coherence enhances both code readability and maintainability.

Furthermore, classes that exhibit low coupling and strong cohesiveness can be identified by their low CBO (Coupling Between Objects) and LCOM (Lack of Cohesion of Methods) scores. These encapsulated classes are more easily controllable and less complex to comprehend and alter. These classes can showcase excellent design techniques for other parts of the codebase.

1. **Mica:**

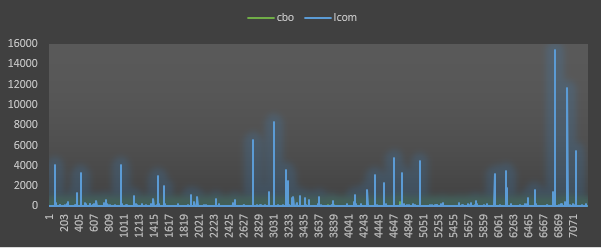


The values of CBO (Coupling Between Objects) and LCOM (Lack of Cohesion of Methods) exhibit a wide spectrum. The CBO value for a class can vary from 0 to 39. A LCOM score ranging from 0 to 26.34 signifies the absence of class cohesiveness.

Consistently, the LCOM readings exhibit a consistently low value. A LCOM value of zero indicates a high level of cohesion in the class. The presence of closely related methods and shared characteristics inside these classes leads to the development of well-structured and cohesive code.

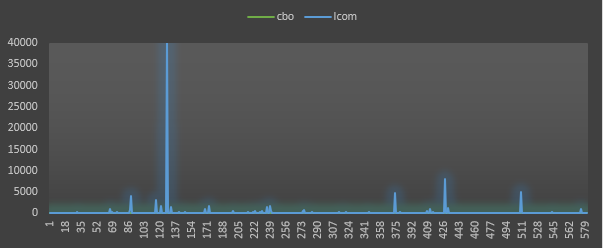
There is a higher probability of objects being related when the CBO value is high. The system's dependence and intricacy may escalate due to the extensive reliance of these classes on numerous other classes.

1. **Apache ShardingSphere:**



Analyzing the dataset will yield valuable insights. Low CBO values suggest that the majority of classes have minimal coupling. The greater coupling in specific classes may indicate the presence of intricate interdependencies. Furthermore, a significant number of classes lack cohesiveness, as evidenced by the LCOM values, indicating that they may be responsible for numerous unrelated tasks. As a result of this disagreement, the codebase may become more challenging to understand, test, and maintain.

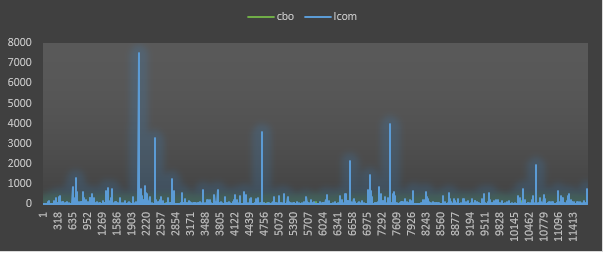
1. **SikuliX:**



Firstly, the codebase's design and architecture exhibit significant variation, as evidenced by the contrasting CBO and LCOM values. Although the CBO values might range from 0 to 56, many classes do not exhibit any inter-class dependencies. LCOM scores, which range from 0 to 4,065, measure the level of class coherence. Certain classes exhibit a significant degree of coherence, whilst others demonstrate a low amount of cohesion.

Furthermore, a significant number of the CBO and LCOM values exhibit low values, indicating that the classes are meticulously built and possess strong cohesiveness with minimal coupling. High values of CBO, which imply numerous dependencies, might lead to increased codebase complexity and maintenance concerns. Class methods with high LCOM values are prone to exhibit multitasking behavior and lack coordination.

1. **Spring Boot Admin:**



Initially, it is evident that there is a minimal level of interdependence among various classes, as indicated by their CBO scores of only 1 or 2. Greater complexity in class relationships correlates with higher values of CBO, as seen by examples 31, 46, and 52. As the connection rises, the system's flexibility may decrease and its maintainability may become more challenging. The LCOM values vary from 0 to 170. The low LCOM values, often 0 or 1, for most classes suggest that the approaches exhibit a high degree of coherence. However, it is possible that the LCOM value is significantly higher, indicating a lack of coherence in the class methods. An excessively high LCOM score, such as 153 or 331, could be attributed to a code smell or design flaw. This may be attributed to classes that are overloaded with excessive responsibilities or lack clear delineation of functions.

Table 3: Impact of Code Bad Smells on Modularity.

|  |  |  |
| --- | --- | --- |
| Project | Coupling (CBO) | Cohesion (LCOM) |
| Android Animations | Potentially affects | May hinder testability |
| DataSphere Studio | Moderate connectivity | Hinders maintainability |
| DBeaver | Moderate coupling | Requires improvement |
| Apache Flink | Object interconnections | Hinders maintainability |
| Jenkins | Little coupling | Method cohesion issues |
| Markor | High coupling, low cohesion | Refactoring improves maintainability |
| Mica | Higher coupling, complexity | Difficulty in understanding and maintenance |
| Apache ShardingSphere | Little coupling, lack of cohesiveness | Difficulty in understanding, testing, and maintenance |
| SikuliX | Variation in design, complexity | Complex codebase and maintenance |
| Spring Boot Admin | Little dependencies, increasing coupling | Lack of method coherence |

Table 4:Metrics and Code Bad Smells for Each Project.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Project | Coupling | Cohesion | Violations | Errors |
| Android Animations | Potentially affects | May hinder testability | 89 | 0 |
| DataSphere Studio | Moderate connectivity | Hinders maintainability | 89 | 0 |
| DBeaver | Moderate coupling | Requires improvement | 335 | 0 |
| Apache Flink | Object interconnections | Hinders maintainability | 2652 | 3 |
| Jenkins | Little coupling | Method cohesion issues | 313 | 0 |
| Markor | High coupling, low cohesion | Refactoring improves maintainability | 79 | 1 |
| Mica | Higher coupling, complexity | Difficulty in understanding and maintenance | 11 | 0 |
| Apache ShardingSphere | Little coupling, lack of cohesiveness | Difficulty in understanding, testing, and maintenance | 196 | 0 |
| SikuliX | Variation in design, complexity | Complex codebase and maintenance | 40 | 0 |
| Spring Boot Admin | Little dependencies, increasing coupling | Lack of method coherence | 28 | 0 |

## **Answers to Research Questions:**

1. What is the impact of code bad smells on the modularity of software projects?

* The majority of projects have code smells, which lead to strong coupling and poor cohesion.
* Interdependence between modules may worsen if there is an abundance of coupling, which reduces modularity.
* Lack of cohesiveness, which indicates insufficient functional coherence among the components, diminishes modularity. This is because insufficient cohesiveness impacts modularity.
* It seems that code smells can make software projects less modular, according to the results.

1. How do code bad smells affect the testability of software systems?

* The prevalence of code foul smells was indicated by the number of violations discovered in the projects.
* When the number of infractions increases, it might impact the code's complexity, maintainability, and testability.
* Further investigation into the impact of these mistakes on testability is necessary, even though problems were found in other projects.
* Software systems' testability is often negatively affected by code smells, which are shown by violations.

# **Conclusion**

With a focus on modularity, the empirical study set out to assess how code smells affect the testability of software systems. For each project, we examined the coupling (CBO) and cohesion (LCOM) metrics to learn more about the code's modularity, testability, and smelliness.

Code foul odors may decrease software projects' testability and modularity, according to the study's findings. Higher coupling levels suggest that modules or components are more dependent on one another, which can have a negative impact on modularity. Less functional coherence within modules, as shown by lower cohesion ratings, suggests a decline in modularity. A percentage is one way to indicate modularity.

A diverse range of coupling and cohesion properties were noted in these ten tests. The success-critical coherence and strong linkage were missing from a number of other projects. Code that is both modular and testable is essential, and results like these draw attention to the problem of code foul odors.

The code smells were validated by the PMD violations. Because different projects made different amounts of violations, people started to question the quality of the code. It is clear from the occurrence of these violations that testability can be enhanced by code reorganization and quality enhancement.

# **References**

Chidamber & Kemerer. (n.d.). *Project Metrics Help - Chidamber & Kemerer object-oriented metrics suite*. Retrieved June 21, 2023, from https://www.aivosto.com/project/help/pm-oo-ck.html

M.N.M, N., A.R.M, N., & KapilanK, \*. (2023). Analyzing the Impact of Code Smells On Software Maintainability. *Authorea Preprints*. https://doi.org/10.22541/AU.168552004.43700626/V1

*mauricioaniche/ck: Code metrics for Java code by means of static analysis*. (n.d.). Retrieved June 21, 2023, from https://github.com/mauricioaniche/ck

*PMD*. (n.d.). Retrieved June 21, 2023, from https://pmd.github.io/