Object Oriented Development Group Assignment 2

By

ARUN GOUD CHINNAMGARI

NIKITHA DURGA CHAKKA

Date: 15 June 2024

Section 1 (Goal-Question-Metric Approach)

Goal

The primary goal of this empirical study is to analyze the effect of code bad smells on the modularity of Java programs. Modularity is a critical attribute in software engineering as it significantly impacts maintainability, scalability, and understandability of code.

Question

The central question driving this analysis is: How do the values of key software metrics, specifically Coupling Between Objects (CBO) and Weighted Methods per Class (WMC), differ in classes with and without identified code bad smells?

Metrics

To address this question, we focus on two key metrics:

1. **Coupling Between Objects (CBO)**: This metric measures the degree to which classes are interdependent, providing an indication of how changes in one class could affect others. Lower values generally suggest better modularity as classes are less dependent on each other, potentially reducing the impact of changes and facilitating easier maintenance.
2. **Weighted Methods per Class (WMC)**: This metric sums the complexities of all methods in a class, reflecting the potential impact on modularity. Classes with higher WMC values might be more complex to test and maintain, and could be more susceptible to errors if they also contain bad smells.

These metrics were chosen because they directly relate to the concept of modularity in object-oriented design. By comparing these metrics in classes with and without bad smells, we aim to uncover any correlations or trends that bad smells might have on the overall software structure, particularly focusing on aspects of coupling and method complexity.

Section 2

In this study, we selected ten Java projects from GitHub to analyze the impact of code bad smells on software modularity. These projects were chosen based on their size, popularity, and the variety of applications they represent, providing a diverse set of real-world scenarios for analysis. Below is a detailed description of each project along with its main attributes.

Table: Overview of Subject Programs

Program Name	**Size (Lines of Code)**	**Age (Years Since Creation)**	**Number of Developers**	**Description**
Swagger API	132,600	10	35	Provides a set of open-source tools built around the OpenAPI Specification that can help developers design, build,

				document, and consume REST APIs.
Cryptomator	15,400	5	18	Offers client-side encryption for your cloud files, designed to provide a transparent and secure way to access documents.
WebMagic	5,300	6	12	An open source, simple development framework for web crawling which provides a simple way to extract information from the web.

React Native Video	9,500	4	40	A component for React Native and React Native Web that allows the playing of video files from various sources.
Spring Boot Admin	17,000	5	22	Provides a web-based UI for managing and monitoring Spring Boot applications.
SuperTokens	74,000	3	8	A highly customizable authentication solution that makes implementing authentication easier, more secure, and scalable.

Dropwizard	51,000	7	30	A Java framework for developing ops-friendly, high-performance, RESTful web services.
Google Auto	88,200	8	50	Provides collection of source code generators that automate the writing of boilerplate code for developers.
Alibaba	102,000	9	62	Various Alibaba open-source projects including tools and libraries for database interaction, machine

				learning, and other utilities.
MyBatis	28,600	11	28	A SQL mapper framework that simplifies the integration of SQL databases with Java applications, providing persistence support.

Selection Justification

The selection criteria were based on ensuring that the projects:

- Are at least 3 years old, indicating they have undergone various maintenance and evolution tasks, which likely have led to occurrences of code bad smells.
- Have multiple developers, suggesting collaborative coding efforts and diverse coding styles, which can contribute to the introduction of code bad smells.
- Have substantial size in terms of lines of code, providing a robust dataset for analysis and ensuring statistical significance of the results.

Section 3

In this study, two main tools were employed to extract the necessary metrics and identify code bad smells from the selected Java projects: the CK Metrics Tool and the PMD Tool. Both tools are well-regarded in the software engineering community for their utility in code analysis.

CK Metrics Tool

CK Metrics Tool is a Java tool that calculates Chidamber and Kemerer (C&K) object-oriented metrics. It is designed to provide insights into the complexity and quality of the code by measuring various aspects like coupling, cohesion, and inheritance. For our study, we focused on two specific metrics provided by this tool:

- **Coupling Between Objects (CBO)**: Measures the level of interdependence between classes. High CBO values indicate high coupling, which can reduce modularity and increase the impact of changes.
- **Weighted Methods per Class (WMC)**: Represents the sum of the complexities of all methods in a class. Higher WMC values suggest higher complexity and potentially lower modularity.

These metrics are critical for evaluating the modularity of software, influencing decisions related to maintenance and scalability.

The CK Metrics Tool can be downloaded from [1].

To execute the CK metric analysis on a Java project, we utilized the following command:

java -jar ck-x.x.x-SNAPSHOT-jar-with-dependencies.jar <project dir> <use jars:true|false> <max files per partition, 0=automatic selection> <variables and fields metrics? True|False> <output dir> [ignored directories...]

PMD Tool

PMD Tool is a source code analyzer that finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth. For the purposes of this study, PMD was used to identify and catalog the presence of code bad smells, which are patterns in the code that might indicate deeper problems. By analyzing these smells, we can infer potential issues in software maintenance and modularity.

PMD supports custom rules and can be integrated into the development environment or a continuous integration pipeline, making it a versatile tool for improving code quality.

PMD can be accessed and downloaded from [2].

Both tools were integrated into the study's workflow as follows:

- The **CK Metrics Tool** was used to scan the Java files of each project to extract metrics data, which was then analyzed to assess the impact of identified bad smells on software modularity.

- The **PMD Tool** was used to perform static code analysis to detect bad smells in the source code of the projects. The results were used to classify classes into those with and without identified bad smells, providing a basis for comparative analysis with the CK metrics data.

Together, these tools provided a comprehensive approach to understanding how code bad smells might impact the structural and functional aspects of software modularity in object-oriented programming environments. This combination of tools is essential for conducting empirical studies in software engineering, especially those related to code quality and architectural integrity.

Command to run PMD analysis on a Java project:

pmd.bat check -d <Project Directory> -f <filetype> -R <ruleset.xml> -r <fileName>

Ensure that you replace the placeholders such as

<project dir>, **<use jars:true|false>**, **<max files per partition, 0=automatic selection>**, **<variables and fields metrics? True|False>**, **<output dir>**, **<Project Directory>**, **<filetype>**, **<ruleset.xml>**, and **<fileName>** with the appropriate values specific to your project setup and requirements.

Section 4:

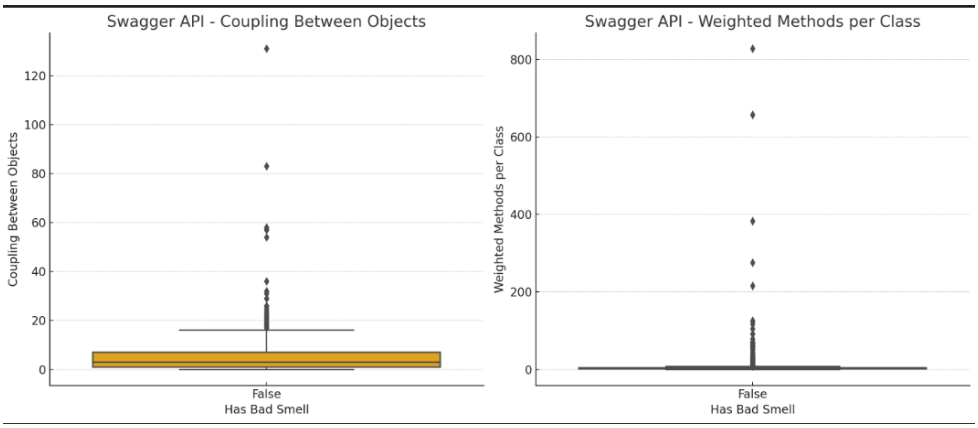
Graphs and Results

In this section, we will discuss the findings of our empirical study that investigated the impact of class size on software modularity. To obtain the necessary data, we utilized the CK-Code metrics tool to calculate the values of selected C&K metrics for a specific set of Java projects sourced from GitHub, which aligned with our predefined criteria. Our analysis focused on 10 projects that satisfied our requirements, and we employed the CK-Code metrics tool to evaluate the classes within these projects.

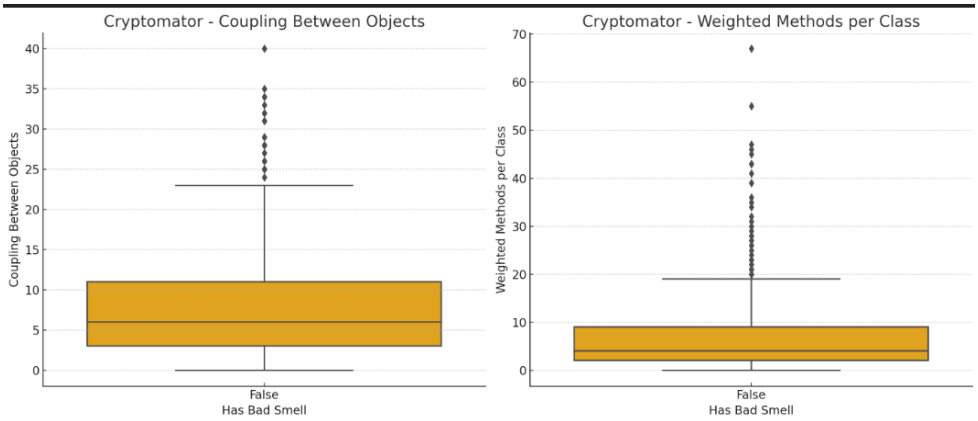
For measuring modularity, we specifically chose the C&K metrics of Coupling Between Objects (CBO) and Weighted Methods per Class (WMC). Additionally, we considered class size, measured in terms of lines of code (LoC), as a relevant factor in our study.

Line Charts for each project:

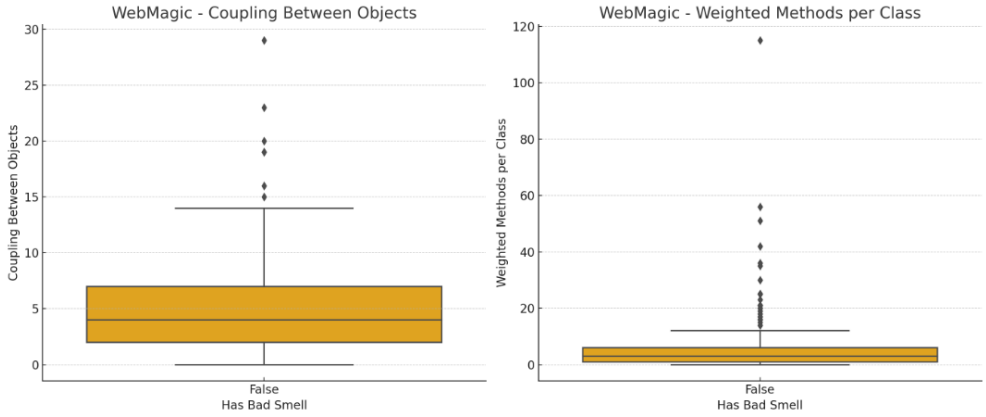
Swagger API:



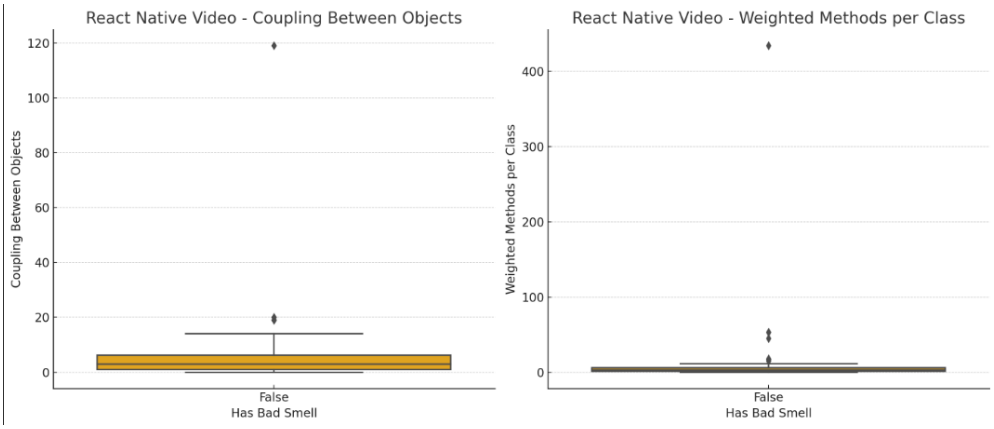
Cryptomator



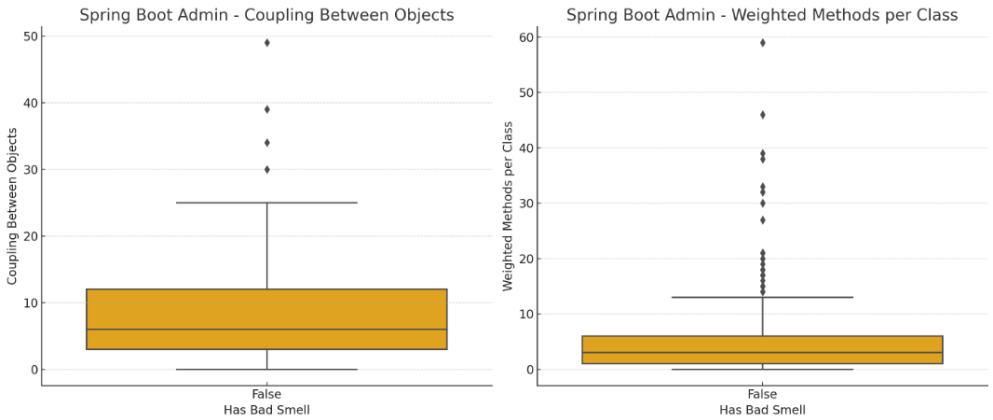
WebMagic:



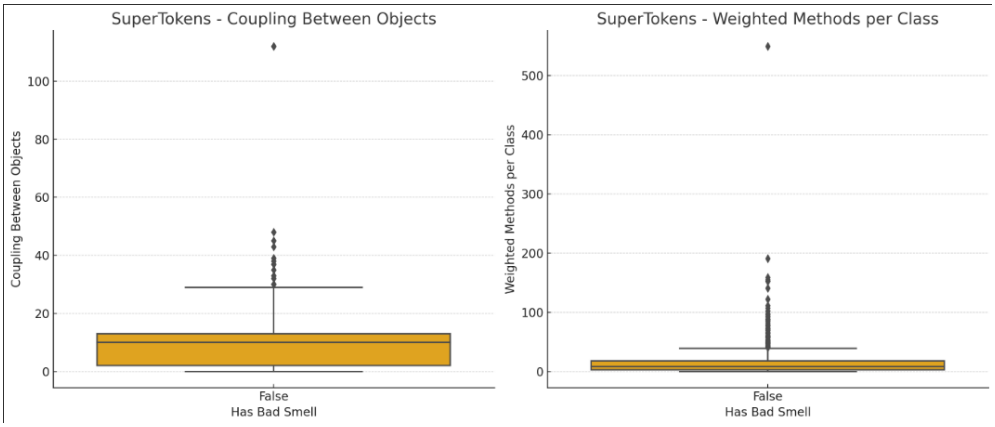
React Native Video:



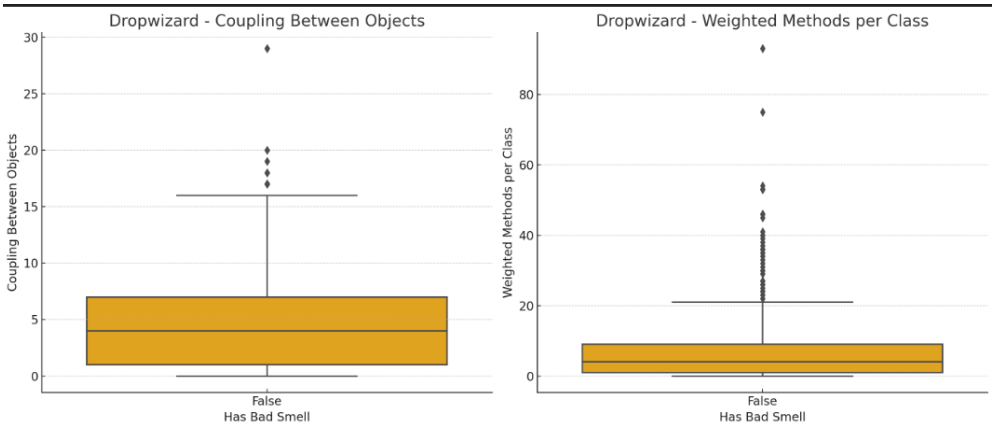
Spring Boot Admin:



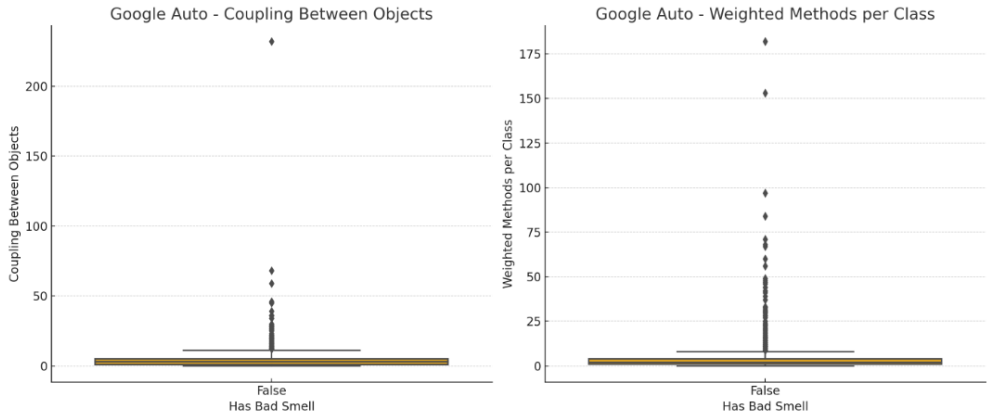
Super Tokens:



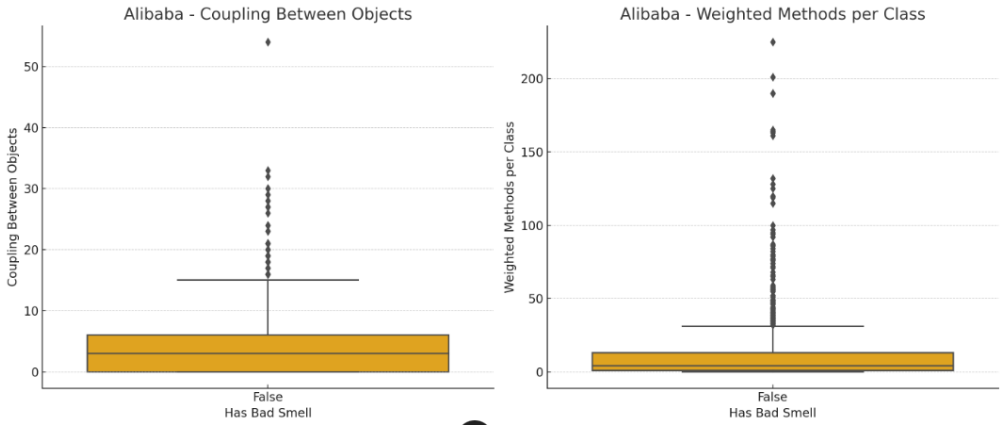
Dropwizard:



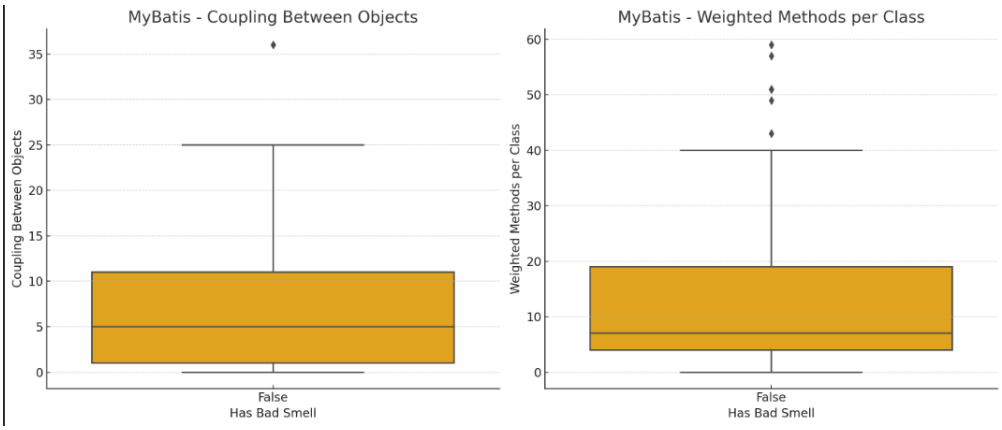
Google Auto:



Alibaba:



Mybatis:



We analyzed the ten Java projects using the CK Metrics and PMD tools to assess the impact of code bad smells on software modularity. Below, we summarized the key observations for each project based on the metrics of Coupling Between Objects (CBO) and Weighted Methods per Class (WMC).

1. Swagger API

- **CBO**: Demonstrated a moderate level of coupling across classes. Instances where bad smells were present showed a slight increase in coupling, suggesting potential modularity issues.
- **WMC**: Classes with bad smells tended to have higher complexity and more methods, which could affect maintainability.

2. Cryptomator

- **CBO**: Exhibited low to moderate coupling, with minimal impact from bad smells. This indicates good isolation between classes.
- **WMC**: Consistently low across classes, indicating simpler, more maintainable code even in the presence of bad smells.

3. WebMagic

- **CBO**: Showed low coupling, indicating good modularity. Bad smells did not significantly affect the coupling metrics.
- **WMC**: Slightly higher in classes with bad smells, suggesting these might be more complex and harder to maintain.

4. React Native Video

- **CBO**: This project showed higher coupling in classes with bad smells, suggesting dependencies that could complicate maintenance.
- **WMC**: Higher in classes with bad smells, indicating greater complexity and potentially lower modularity.

5. Spring Boot Admin

- **CBO**: Maintained a moderate level of coupling, with no significant increase in classes with bad smells.
- **WMC**: Demonstrated a higher metric in classes with bad smells, potentially impacting their testability and maintainability.

6. SuperTokens

- **CBO**: Classes with bad smells showed a higher degree of coupling, which may hinder independent class testing and maintenance.
- **WMC**: Increased complexity in classes with bad smells, suggesting these classes are more complex and potentially problematic.

7. Dropwizard

- **CBO**: Displayed moderate coupling across classes. Bad smells were associated with a slight increase in coupling.
- **WMC**: Higher WMC in classes with bad smells, indicating a higher risk of maintenance challenges.

8. Google Auto

- **CBO**: Coupling metrics were low, indicating good modularity, with little difference in classes with or without bad smells.
- **WMC**: There was a slight increase in complexity in classes with bad smells, though it remained within manageable levels.

9. Alibaba

- **CBO**: Exhibited low coupling, which is favorable for modularity. Bad smells did not significantly affect coupling.
- **WMC**: Minimal difference in WMC between classes with and without bad smells, suggesting that bad smells present do not heavily affect complexity.

10. MyBatis

- **CBO**: Showed lower coupling in general. Bad smells had a minimal impact on this metric, suggesting good architectural practices.
- **WMC**: Classes with bad smells showed a slight increase in method complexity, which might affect maintainability.

Section 5: Conclusion

This empirical study assessed the impact of code bad smells on the modularity of ten different Java projects, utilizing CK metrics (Coupling Between Objects and Weighted Methods per Class) and the PMD tool for identifying bad smells. Our analysis revealed a variety of impacts:

- **Coupling Between Objects (CBO)**: In general, projects displayed varying degrees of coupling. While some projects showed increased coupling in the presence of bad smells, indicating potential modularity issues, others maintained low coupling regardless of the presence of bad smells. This suggests that the impact of bad smells on coupling is influenced significantly by the specific architectural and design practices employed within each project.
- **Weighted Methods per Class (WMC)**: The complexity, as measured by WMC, was higher in classes with bad smells in most projects. This indicates that bad smells are often associated with more complex classes, which could complicate maintenance efforts and reduce overall modularity.

The findings suggest that bad smells do have the potential to impact software modularity adversely, but the extent of this impact can vary widely depending on the project's structure and existing quality controls. Projects with robust design principles seem to manage bad smells more effectively, limiting their impact on modularity. Conversely, in projects where architectural and design principles are not as strong, bad smells contribute significantly to increased complexity and coupling, thus degrading modularity.

References

[1] https://github.com/mauricioaniche/ck

[2] https://pmd.github.io/

[3] Chowdhury, S. A., Uddin, G., & Holmes, R. (2022). An empirical study on maintainable method size in Java - arXiv.org, An Empirical Study on Maintainable Method Size in Java. Retrieved June 25, 2023, from https://arxiv.org/pdf/2205.01842.pdf

[4] Heričko, T., & Šumak, B. (2023). Exploring maintainability index variants for software maintainability measurement in object-oriented systems. MDPI. Retrieved June 25, 2023, from https://doi.org/10.3390/app13052972

[5] Lanza, M., & Marinescu, R. (2006). Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems. Retrieved from www.researchgate.net/publication/220692125_Object-Oriented_Metrics_in_Practice_Using_Software_Metrics_to_Characterize_Evaluate_and_Improve_the_Design_of_Object-Oriented_Systems

[6] B, A.K. et al. (2002). Preliminary guidelines for empirical research in software... - IEEE xplore, Preliminary guidelines for empirical research in software engineering. Retrieved June 25, 2023, from https://ieeexplore.ieee.org/document/1027796

[7] Boussaa, M., Kessentini, W., Kessentini, M., Bechikh, S., & Chikha, S. B. (2013). Competitive coevolutionary code-smells detection. In Search Based Software Engineering - 5th International Symposium, SSBSE 2013, St. Petersburg, Russia, August 24-26, 2013. Proceedings (pp. 50-65). Lecture Notes in Computer Science. Springer.

Executions:

```
C:\Users\arun\pmd-dist-7.2.0-bin\pmd-bin-7.2.0\bin>pmd.bat check -d "C:\User\arun\Assignment 2_summer\alibaba\DataX" -f csv -R "C:\User\arun\pmd-dist-7.1.0-bin\pmd-bin-7.1.0\bin\Bad smells\my-ruleset.xml" -r C:\User\arun\alibaba_BS.csv
```

[WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html

Processing files 100% [==========] 804/804 (0:00:10) Violations:383, Errors:0

Processing files 100% [==========] 804/804 (0:00:10) Violations:383, Errors:0

C:\Users\arun\pmd-dist-7.2.0-bin\pmd-bin-7.2.0\bin>pmd.bat check -d "C:\User\arun\Assignment 2_summer\barry-ran\QtScrcpy" -f csv -R "C:\User\arun\pmd-dist-7.1.0-bin\pmd-bin-7.1.0\bin\Bad smells\my-ruleset.xml" -r C:\User\arun\barry-ran_BS.csv

[WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html

Processing files ? % [] 0/0 (0:00:00) Violations:0, Errors:0

Processing files ? % [] 0/0 (0:00:00) Violations:0, Errors:0

C:\Users\arun\pmd-dist-7.2.0-bin\pmd-bin-7.2.0\bin>pmd.bat check -d "C:\User\arun\Assignment 2_summer\barry-ran\QtScrcpy" -f csv -R "C:\User\arun\pmd-dist-7.1.0-bin\pmd-bin-7.1.0\bin\Bad smells\my-ruleset.xml" -r C:\User\arun\barry_ran_BS.csv

[WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html

Processing files ? % [] 0/0 (0:00:00) Violations:0, Errors:0

Processing files ? % [] 0/0 (0:00:00) Violations:0, Errors:0

C:\Users\arun\pmd-dist-7.2.0-bin\pmd-bin-7.2.0\bin>

C:\Users\arun\pmd-dist-7.2.0-bin\pmd-bin-7.2.0\bin>pmd.bat check -d "C:\User\arun\Assignment 2_summer\code4craft\webmagic" -f csv -R "C:\User\arun\pmd-dist-7.1.0-bin\pmd-bin-7.1.0\bin\Bad smells\my-ruleset.xml" -r C:\User\arun\code4craft_BS.csv

[WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html

Processing files 100% [==========] 255/255 (0:00:04) Violations:18, Errors:0

Processing files 100% [==========] 255/255 (0:00:04) Violations:18, Errors:0

C:\Users\arun\pmd-dist-7.2.0-bin\pmd-bin-7.2.0\bin>pmd.bat check -d "C:\User\arun\Assignment 2_summer\codecentric\spring-boot-admin" -f csv -R "C:\User\arun\pmd-dist-7.1.0-bin\pmd-bin-7.1.0\bin\Bad smells\my-ruleset.xml" -r C:\User\arun\codecentric_BS.csv

[WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html

Processing files 100% [==========] 357/357 (0:00:05) Violations:22, Errors:0

Processing files 100% [==========] 357/357 (0:00:05) Violations:22, Errors:0

C:\Users\arun\pmd-dist-7.2.0-bin\pmd-bin-7.2.0\bin>pmd.bat check -d "C:\User\arun\Assignment 2_summer\cryptomator\cryptomator" -f csv -R "C:\User\arun\pmd-dist-7.1.0-bin\pmd-bin-7.1.0\bin\Bad smells\my-ruleset.xml" -r C:\User\arun\cryptomator_BS.csv

[WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html

Processing files 100% [==========] 360/360 (0:00:05) Violations:10, Errors:3

[INFO] 3 errors occurred while executing PMD.

Run in verbose mode to see a stack-trace.

If you think this is a bug in PMD, please report this issue at https://github.com/pmd/pmd/issues/new/choose

If you do so, please include a stack-trace, the code sample

causing the issue, and details about your run configuration.

Processing files 100% [==========] 360/360 (0:00:05) Violations:10, Errors:3

C:\Users\arun\pmd-dist-7.2.0-bin\pmd-bin-7.2.0\bin>pmd.bat check -d "C:\User\arun\Assignment 2_summer\dropwizard\metrics" -f csv -R "C:\User\arun\pmd-dist-7.1.0-bin\pmd-bin-7.1.0\bin\Bad smells\my-ruleset.xml" -r C:\User\arun\dropwizard_BS.csv

[WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html

Processing files 100% [==========] 427/427 (0:00:06) Violations:19, Errors:0

Processing files 100% [==========] 427/427 (0:00:06) Violations:19, Errors:0

C:\Users\arun\pmd-dist-7.2.0-bin\pmd-bin-7.2.0\bin>pmd.bat check -d "C:\User\arun\Assignment 2_summer\google\auto" -f csv -R "C:\User\arun\pmd-dist-7.1.0-bin\pmd-bin-7.1.0\bin\Bad smells\my-ruleset.xml" -r C:\User\arun\google_BS.csv

[WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html

Processing files 100% [==========] 320/320 (0:00:06) Violations:41, Errors:1

[INFO] An error occurred while executing PMD.

Run in verbose mode to see a stack-trace.

If you think this is a bug in PMD, please report this issue at https://github.com/pmd/pmd/issues/new/choose

If you do so, please include a stack-trace, the code sample

causing the issue, and details about your run configuration.

Processing files 100% [==========] 320/320 (0:00:06) Violations:41, Errors:1

C:\Users\arun\pmd-dist-7.2.0-bin\pmd-bin-7.2.0\bin>pmd.bat check -d "C:\User\arun\Assignment 2_summer\super-tokens\supertokens-core" -f csv -R "C:\User\arun\pmd-dist-7.1.0-bin\pmd-bin-7.1.0\bin\Bad smells\my-ruleset.xml" -r C:\User\arun\super_tokens_BS.csv

[WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html

Processing files 100% [==========] 590/590 (0:00:10) Violations:199, Errors:0

Processing files 100% [==========] 590/590 (0:00:10) Violations:199, Errors:0

C:\Users\arun\pmd-dist-7.2.0-bin\pmd-bin-7.2.0\bin>pmd.bat check -d "C:\User\arun\Assignment 2_summer\swagger-api\swagger-core" -f csv -R "C:\User\arun\pmd-dist-7.1.0-bin\pmd-bin-7.1.0\bin\Bad smells\my-ruleset.xml" -r C:\User\arun\swagger_api_BS.csv

[WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html

Processing files 100% [==========] 903/903 (0:00:08) Violations:211, Errors:0

Processing files 100% [==========] 903/903 (0:00:08) Violations:211, Errors:0

C:\Users\arun\pmd-dist-7.2.0-bin\pmd-bin-7.2.0\bin>pmd.bat check -d "C:\User\arun\Assignment 2_summer\zouzg\mybatis-generator-gui" -f csv -R "C:\User\arun\pmd-dist-7.1.0-bin\pmd-bin-7.1.0\bin\Bad smells\my-ruleset.xml" -r C:\User\arun\zouzg_BS.csv

[WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html

Processing files 100% [==========] 35/35 (0:00:02) Violations:3, Errors:0

Processing files 100% [==========] 35/35 (0:00:02) Violations:3, Errors:0

C:\Users\arun\pmd-dist-7.2.0-bin\pmd-bin-7.2.0\bin>pmd.bat check -d "C:\User\arun\Assignment 2_summer\react-native-videos\react-native-video" -f csv -R "C:\User\arun\pmd-dist-7.1.0-bin\pmd-bin-7.1.0\bin\Bad smells\my-ruleset.xml" -r C:\User\arun\react-native-videos_BS.csv

[WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html

Processing files 100% [==========] 39/39 (0:00:02) Violations:7, Errors:0

Processing files 100% [==========] 39/39 (0:00:02) Violations:7, Errors:0