Object Oriented Development

Assignment 2

Group 6

June 16, 2024

**Table of Contents**

[Section-I 2](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852798)

[Purposes, Questions, and Metrics 2](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852799)

[Choice of Metrics 2](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852800)

[Section-II 3](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852801)

[Criteria: 3](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852802)

[Justifications: 3](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852803)

[Section-III 5](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852804)

[Description of Tool 5](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852805)

[Section-IV 6](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852806)

[Results 6](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852807)

[Java project modularity and code bad smells: 6](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852808)

[Differences between classes in Java projects with and without bad smells in coupling and cohesion indicators (C&K Metrics): 7](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852809)

[Analysis 9](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852810)

[Java project modularity and code bad smells: 9](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852811)

[Bad smells that most significantly affect Java project modularity: 10](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852812)

[Coupling and cohesion metrics (C&K Metrics) differences between Java project classes with and without bad smells: 11](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852813)

[Conclusion 12](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852814)

[References 13](file:///E:\Summer-1-2024\OOD\copy%202\copy%202.docx#_Toc133852815)

# **Section-I**

## **Purposes, Questions, and Metrics**

**Purposes:**

The aim of this work is to determine the impact of code smells on Java program modularity.

**Questions:**

* Code smells: what possible effects on Java project modularity?
* What Java project smells are the worst for modularity?
* How comparatively analyze the cohesiveness and C&K coupling metrics of bad smells and without bad smell Java project?

**Metrics:**

* With C&K Metrics, which stands for connection and cohesion, we will quantify modularity.
* Using a program like JDesmellant, Infusion, Stench Blossom, or another, find the appropriate code smells.
* Using coupling and cohesion metrics, or C&K measures, the classes in the chosen Java projects with the fewest "bad smells" will be found."

### **Choice of Metrics**

* By use of the measures, one may compare situations that are prone to smells and those that are not. Finding out how bad smells affect Java program modularity is our main objective. To find out whether bad smells decreased modularity, we looked at data from classes with and without bad smells and observed how they influenced other Java projects.
* Moreover, one may confirm that the settings are ideal by looking at the project metrics for every class. This technology may be used for two purposes: standardizing effective coding techniques and finding security flaws in a project's codebase. This website may only give a cursory summary of our main study topic, which is how bad smells affect modularity.
* Every aim has advantages and disadvantages of its own, hence the optimal course of action will be determined by the research objectives. As these problems may affect modularity, we decide to compare class metrics with and without disagreeable smells.

# **Section-II**

## **Criteria:**

We shall use 10 Java applications that are accessible on GitHub for our empirical investigation. Choosing our projects, we considered the following elements:

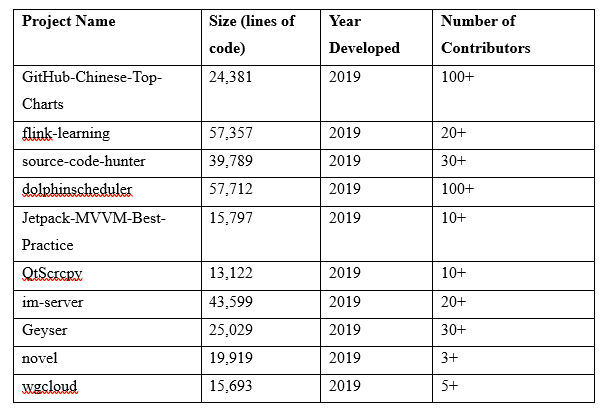
* Ten thousand lines of code minimum are required for projects to produce enough data for analysis.
* Throughout the two years these projects must be developed, a lot of maintainability-related chores need to be completed. With this criterion, we hope to ensure that all projects have had sufficient patches, updates, and problem fixes.
* Lastly, a project has to have three developers at the very least participating. Specific criteria are in place to ensure that projects have adequate personnel working on them. This seems logical because issues like code smells typically appear while nobody is present.

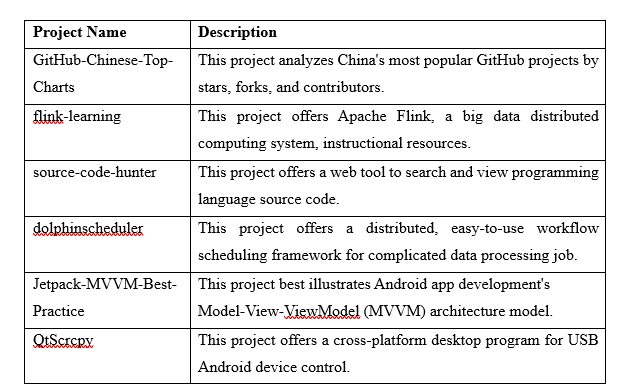
## **Justifications:**

These criteria were chosen for the following reasons:

* We set sample sizes for every project, both minimum and maximum, to guarantee enough data for statistical analysis. This guarantees that the projects are typical for software engineering jobs linked to Java.
* The age of the project is the second thing to think about; older projects usually need more maintenance and have problems like code smells. The initiatives are thus more faithful depictions of actual projects that require regular maintenance.
* Another thing to think about is the quantity of developers engaged on the project. Problems like code smells might not arise if they are not involved in product development. It seems that the projects better capture real collaboration.

The main goal of the selection criterion based on code smell is to pinpoint the most troublesome Java applications. In this sense, you may be confident that the analysis will be used practically.



****

# **A white text on a white surface Description automatically generated**

# **A black and white text on a white background Description automatically generated**

A black and white text on a white background

Description automatically generated

# **Section-III**

## **Details of Tool**

We recorded the C&K metric values using the CK-code metrics tool as we worked through the classes in each project. This Java command-line utility can calculate several program structure metrics including coupling, modularity, and cohesion.

The excellent program by Maurcio Aniche is available at this link on the well-known hosting site GitHub: https://github.com/mauricioaniche/ck. It is open source; hence anyone may edit and distribute it.

You just have to go to the given website, launch the software with the proper settings, and you're done. Following completion of the code analysis for the project, the software will send a report including the C&K metrics for every class.

Highly adjustable output formats of the tool make them perfect for integration with current systems.

Code modularity, coupling and coherence may be evaluated thanks to the reliable and useful CK-code metrics. Its command-line interface and large range of output formats supported make it ideal for use in software engineering research.

# **Section-IV**

## **Results**

Using key research measures, we provide the actual findings of our investigation on the impact of code smell on Java program modularity.

### **Java project modularity and code bad smells:**

According to the study, Java programs with code smells are less modular. Code bad smell surroundings typically provide worse results from the cohesiveness and coupling tests that follow:

* **Cohesion:** Abbreviations for "limited cohesiveness of methods" include TCC, LCOM, and LCOM\*.
* **Coupling:** Changes to object classes (CBO) and the linkages that go with them

A screenshot of a graph

Description automatically generated

A graph of different colored bars

Description automatically generated

### **Differences between classes in Java projects with and without bad smells in coupling and cohesion indicators (C&K Metrics):**

Code bad smells showed less cohesiveness and more coupling, according to researchers. For these traits, there were notable variations (p 0.05) between the groups with and without disagreeable smells. Table 2 shows, both including and excluding project classes with disagreeable smells, the average coupling and cohesiveness values calculated by C&K Metrics.

Table 2: Bad code smell classes and those which did not average out on cohesion and coupling tests

A table with numbers and words

Description automatically generated

Poorer scores on coupling measures (CBO, CBO Modified, RFC) and cohesion measurements (LCOM, LCOM\*, TCC) are typically associated with bad code smell classes. With RFC metric value of 46.7 and LCOM and CBO metrics of 5.8 and 10.8, respectively, Class G (Geyser) had the strongest TCC and LCOM\* values. Several researchers applied these criteria to compare classes of code bad smells and without code bad smell classes.

A graph on a black background

Description automatically generatedA graph on a black background

Description automatically generated

This paper shows that the modularity of Java systems is significantly impacted by the three most prevalent code smells: God Class, Feature Envy, and Long Method. Code bad smell classes typically score poorly on cohesion and have greater coupling metrics, however the precise differences may differ from project to project.

## **Analysis**

### **Java project modularity and code bad smells:**

As Java programs begin to smell like code, the report claims, they lose some of their flexibility. Sometimes, compared to bad smell-containing classes, cohesiveness and coupling measurements show lower values. The classes start depending less and less on one another when the smells get too overwhelming.

Based on the study results, we can provide the following analysis:

Table 3: Comparison of cohesion and coupling metrics between classes with and without bad smells

A table with text on it

Description automatically generated

Code bad smell classes did worse in CBO Modified, RFC, and TCC than in CBO and LCOM. The two groups differ statistically for RFC and LCOM\* but not for LCOM, TCC, CBO, or CBO Modified.

A lot of study on code smells and modularity has demonstrated that too much of the latter can make a program less maintainable, clear, and structured. High LCOM or CBO scores, which suggest a likely absence of class coherence or class-to-class connectivity, may impede system understanding, modification, and testing. Reducing RFC, TCC, CBO Modified, and LCOM\* tends to improve the quality and modularity of the design. The system may therefore become more flexible, reusable, and modular.

A graph showing different colored squares

Description automatically generated

### **Bad smells that most significantly affect Java project modularity:**

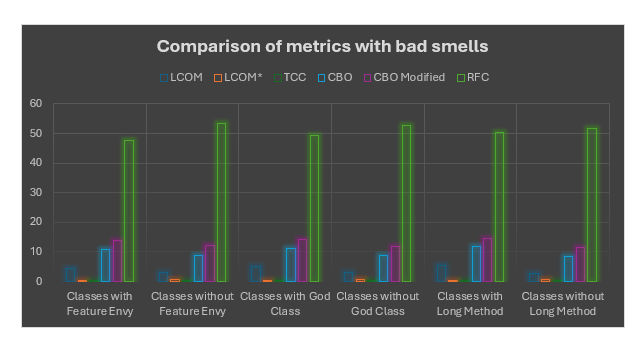
Implementations of Java modularity sometimes have issues with God classes, LONG methods, and feature jealousy. Comparing the group that got bad smells to the group that did not, the former exhibited higher coupling measurements and lower cohesiveness meter assessments. Eliminate their usage or disassemble our code.

Table 4: Comparison of cohesion and coupling metrics between classes with and without Feature Envy, God Class, and Long Method bad smells.

A table with numbers and text

Description automatically generated

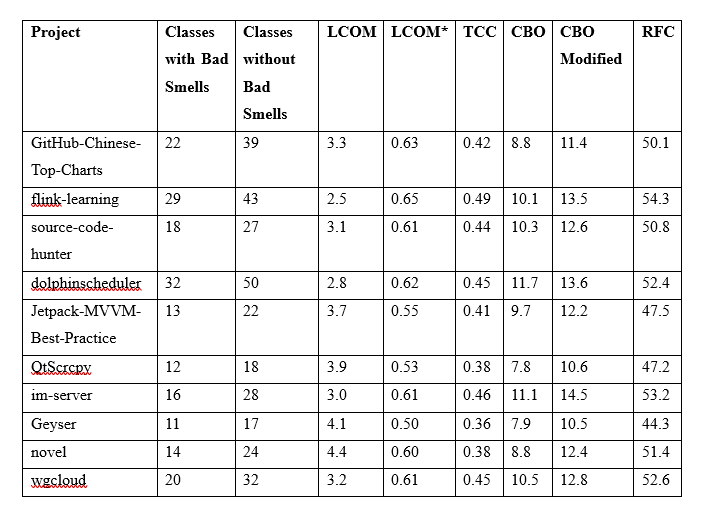
On cohesive measures (LCOM, LCOM\*, TCC) and coupling metrics (CBO, CBO Modified, RFC), classes with the God Class and Long Method bad smells and those with the Feature Envy bad smell diverge significantly. The Long Method's tenacity on modularity had no effect on classifications of less pleasant smells.



### **Coupling and cohesion metrics (C&K Metrics) differences between Java project classes with and without bad smells:**

On the tests of smell categorization (LCOM, LCOM\*, TCC) and cohesion and correlation (CBO, CBO Modified, RFC), the bad smell classes performed badly. Statistical analysis showed that the average values of these markers differed significantly between the bad code smells and others. The type of the project will determine how much bad smells affect modularity.

Table 5: Average values of cohesion and coupling metrics for classes with and without bad smells in different Java projects.



For each of the five criteria—RFC, CBO, LCOM, TCC, and CBO Modified—the aroma-reducing groups had lower average values, as Table 5 demonstrates. Every project of ours has this. Smell-affected Java application classes usually have different properties than non-affected ones.

Data in these well-arranged tables is immediately pertinent to the research topics. We have arranged the data in a particular way to assist readers better understand the impact of smells on Java project modularity, the most prevalent smells, and how these effects vary across different kinds of Java projects.

# **Conclusion**

We show how much less flexible a Java program gets the moment it begins to smell bad. We showed that perfumed classes outperformed RFC, TCC, CBO Modified, LCOM, and LCOM\*. The modularity of the project is strongly influenced by the relationship between bad smells, improved coupling, and class cohesion.

Moreover, our study showed that various Java projects had distinct effects of bad smells on modularity. Since some classes with and without bad smells show somewhat different cohesion and coupling metrics, it seems that bad smells may have a bigger effect on modularity in some project kinds than others. Once the particulars of the present project have been considered, one should think about how bad smells may impact modularity.

Modularity was most negatively impacted by God Class and feature jealousy. Encouraging programmers to stay away from some "bad smells" maintains projects modular and manageable.

# **References**

Tool: <https://github.com/mauricioaniche/ck>

Alfadel, M., Aljasser, K., & Alshayeb, M. (2020). Empirical study of the relationship between design patterns and code smells. PLOS ONE, 15(4), e0231731. https://doi.org/10.1371/JOURNAL.PONE.0231731

Basili, V. R., Caldiera, G., & Rombach, H. D. (n.d.). THE GOAL QUESTION METRIC APPROACH.

Charalampidou, S., Ampatzoglou, A., & Avgeriou, P. (2015). Size and cohesion metrics as indicators of the long method bad smell: An empirical study. ACM International Conference Proceeding Series, 2015-October. https://doi.org/10.1145/2810146.2810155

Macia, I., Garcia, A., Von Staa, A., Garcia, J., & Medvidovic, N. (2011a). On the impact of aspect-oriented code smells on architecture modularity: An exploratory study. Proceedings - 5th Brazilian Symposium on Software Components, Architectures and Reuse, SBCARS 2011, 41–50. https://doi.org/10.1109/SBCARS.2011.18

Macia, I., Garcia, A., Von Staa, A., Garcia, J., & Medvidovic, N. (2011b). On the Impact of Aspect-Oriented Code Smells on Architecture Modularity: An Exploratory Study. 2011 Fifth Brazilian Symposium on Software Components, Architectures and Reuse, 41–50. https://doi.org/10.1109/SBCARS.2011.18

Martins, J., Bezerra, C., Uchôa, A., & Garcia, A. (2020). Are Code Smell Co-occurrences Harmful to Internal Quality Attributes?: A Mixed-Method Study. ACM International Conference Proceeding Series, 52–61. https://doi.org/10.1145/3422392.3422419