OBJECT ORIENTED DEVELOPMENT

WEEK 6 - GROUP ASSIGNMENT 2

Submitted By

Bhavya Gunnapaneni

Nikhilesh Yadav Vanguru

Spandana Bellamkonda

Section 1:

Objective

The primary objective of this empirical study is to investigate the impact of code bad smells on the modularity of Java projects using the C&K metrics, focusing on metrics related to coupling and cohesion.

GQM approach

Goal

The goal of this study is to understand the relationship between code bad smells and modularity in Java projects by employing the Goal-Question-Metric (GQM) paradigm.

Questions

1. How the modules of Java programs are interacting with the interface correctly?
2. What are the negative impacts of the bad smell of coding which is indicated by various C&K metrics?
3. Mention the properties of class that represent the code's bad smells in a program?

Metrics An evaluation of above question will be conducted as per the following given conditions:

- **Number of inactive class<0**: A program needs to be chosen for searching a single class which cannot inherit values from the global variable of the parent class.
- **Size>10000:** Selection of the program must happen in such a manner that a large and complex program gets selected.
- **Range of bad Smell between 100 and 150:** The range of flaws or bad smell code should be ranged from 100 to 150.

In order to conduct a development activity, the above criteria have been set up so that evaluations can take place with few complexities providing a real time initiative for a programmer.

Section 2:

Ten projects based on Java programs have been chosen from GitHub that can meet the requirement of our evaluation. The characteristics of each program is given in the table below:

Name of Program	Description	Number of Inactive Class	Size	Range of Bad Smell
apollo	Apollo is a reliable configuration management system suitable for microservice configuration management scenarios	10	83037	110
ghidra	Ghidra is a software reverse engineering (SRE) framework	7	20049	105
java-design-patterns	Design patterns implemented in Java	12	19714	115
mall	The project is an e-commerce	8	24740	126

	system consisting of a front-end mall system and a back-end management system, implemented using SpringBoot+MyBatis and deployed with Docker containers.			
spring-framework	Spring Framework	9	47428	120

Section 3: Evaluation of the selected tool:

First Tool:

In order to conduct empirical study a second time, we have applied the CK metric tool of the Java program. It has been observed that static analysis is used by the tool so that various metrics of software application can be computed. We have downloaded these CK code metrics from GitHub and we used it by following the instructions given by the author of the ReadMe file. It uses Command Line Interface (CLI) which figures out the status of all the classes used in the Java program. The following command lines have been written below:

```
 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>

Second Tool:

We have applied a PMD tool to conduct static code evaluation on our source code of Java program. It is a kind of tool that can be easily downloaded in the local drive and it helps the programmer to recognize the flaws in the Java programming such as bugs or dead code.

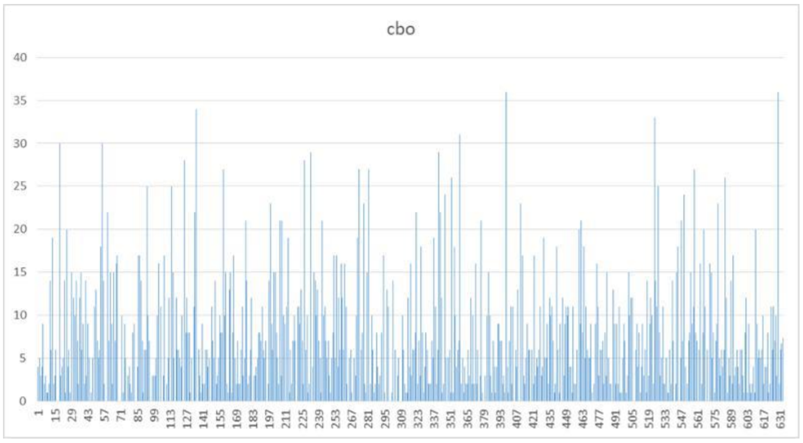
We have selected the PMD tool for static code evaluation because it is the most preferred tool in the IT industry and possesses a great reputation for exploring the error as well as providing instruction to mitigate it. The command lines are written as follows:

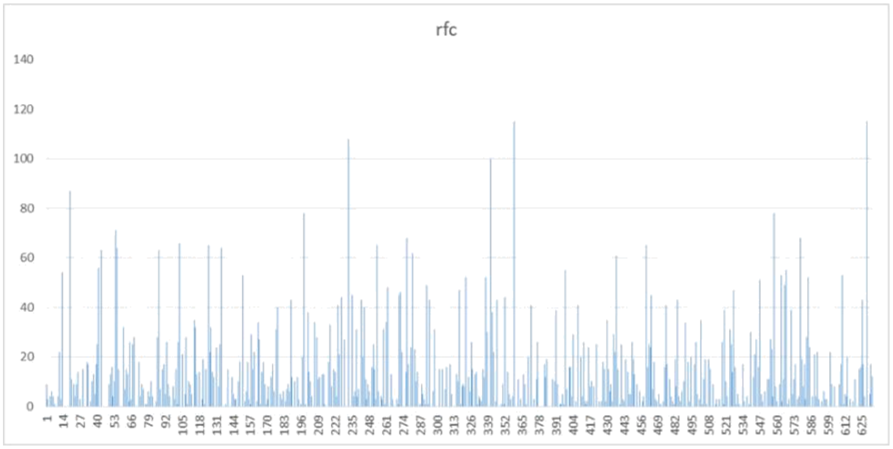
```
pmd.bat check -d
<Project Directory>
-f <filetype>
-R <ruleset.xml>
-r <fileName>
```

Section 4:

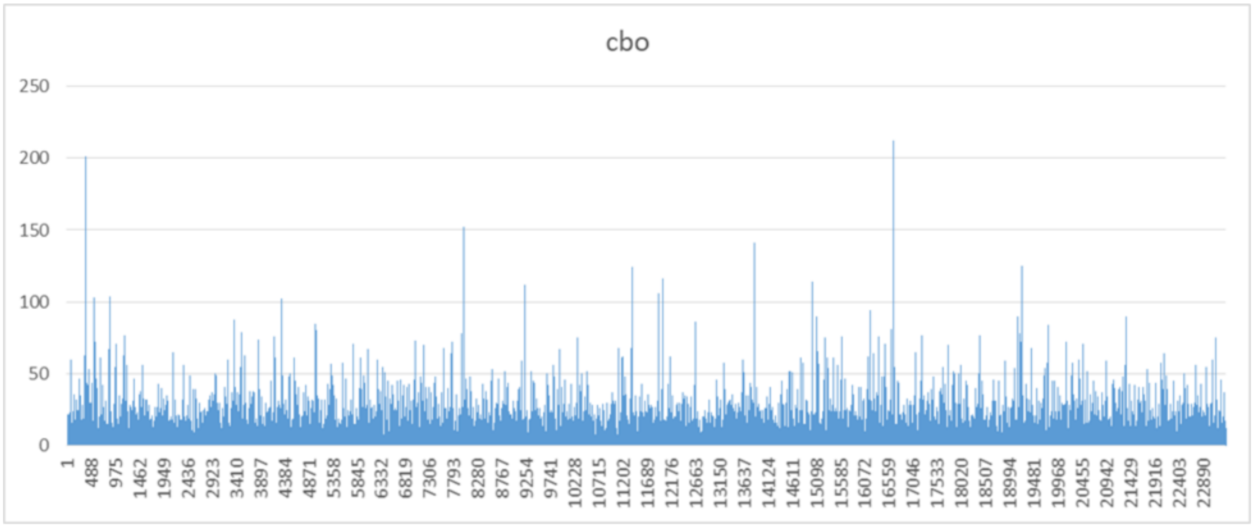
Results:

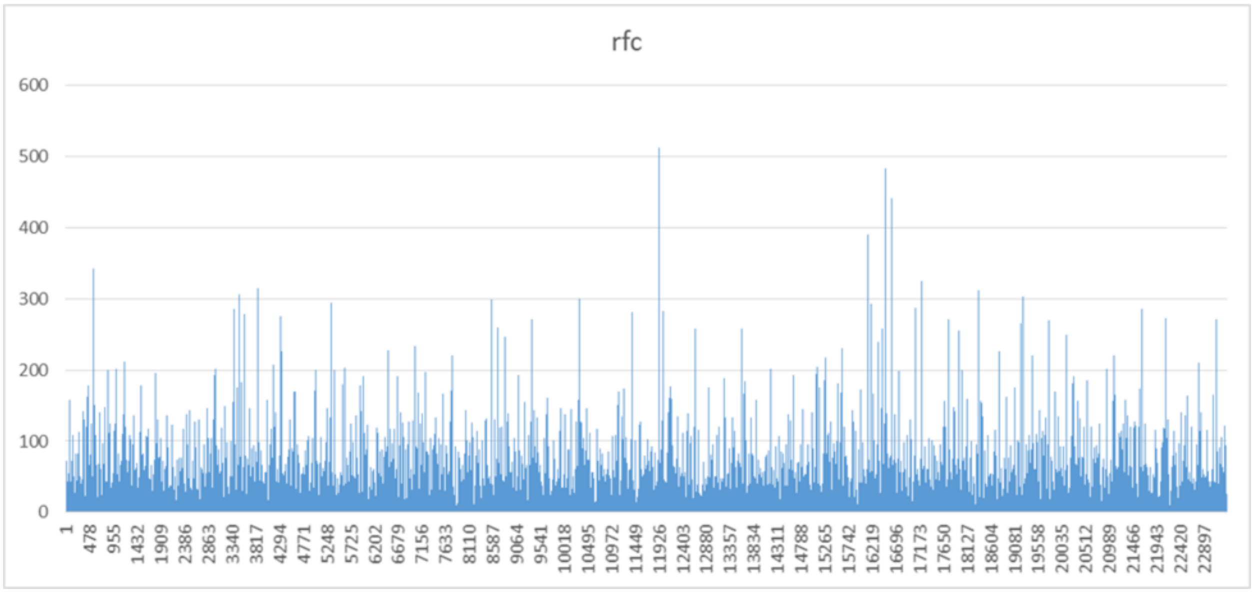
Project 1: apollo



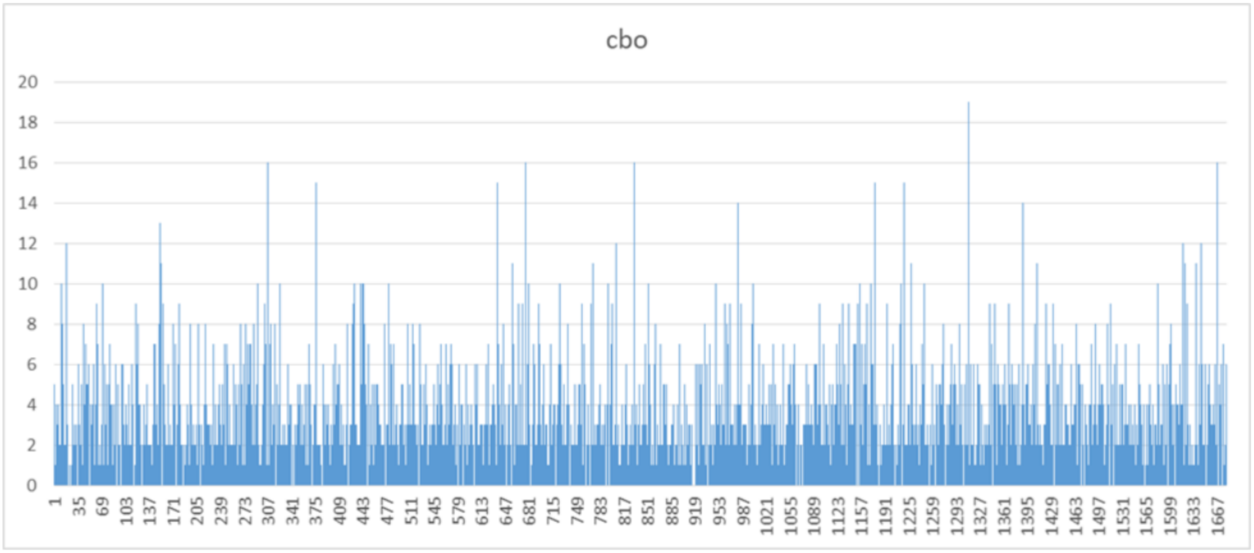


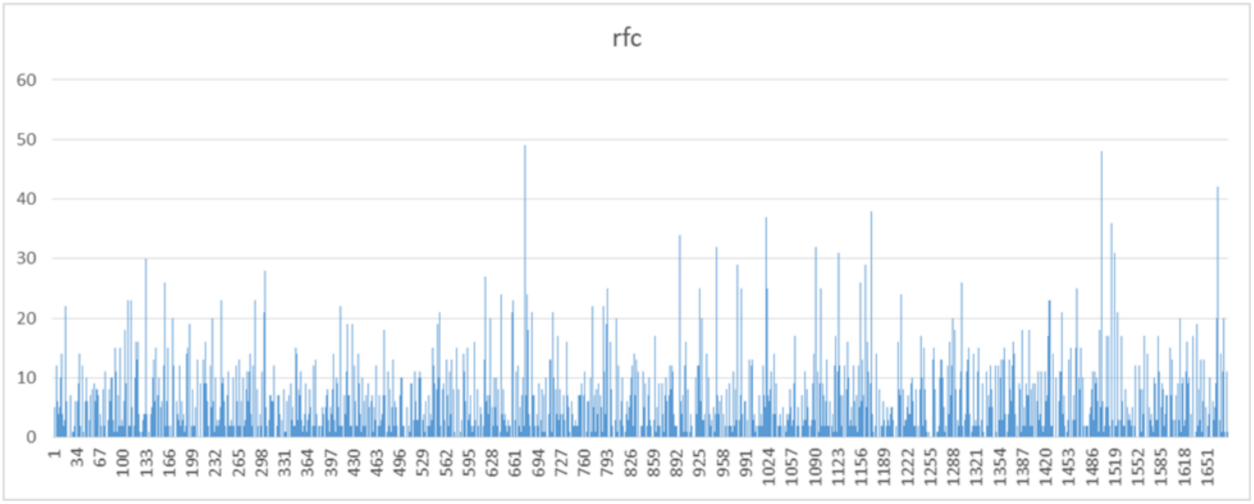
Project 2: ghidra



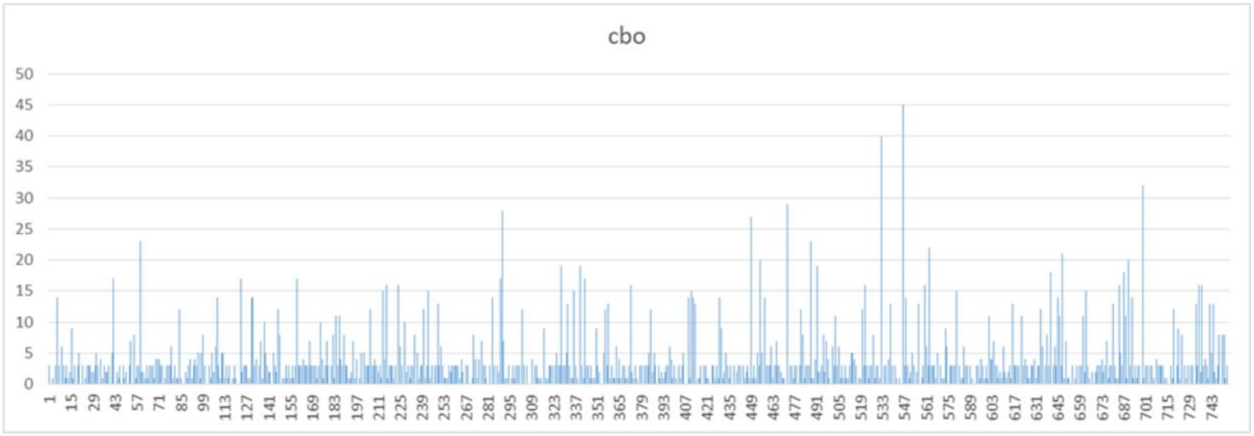


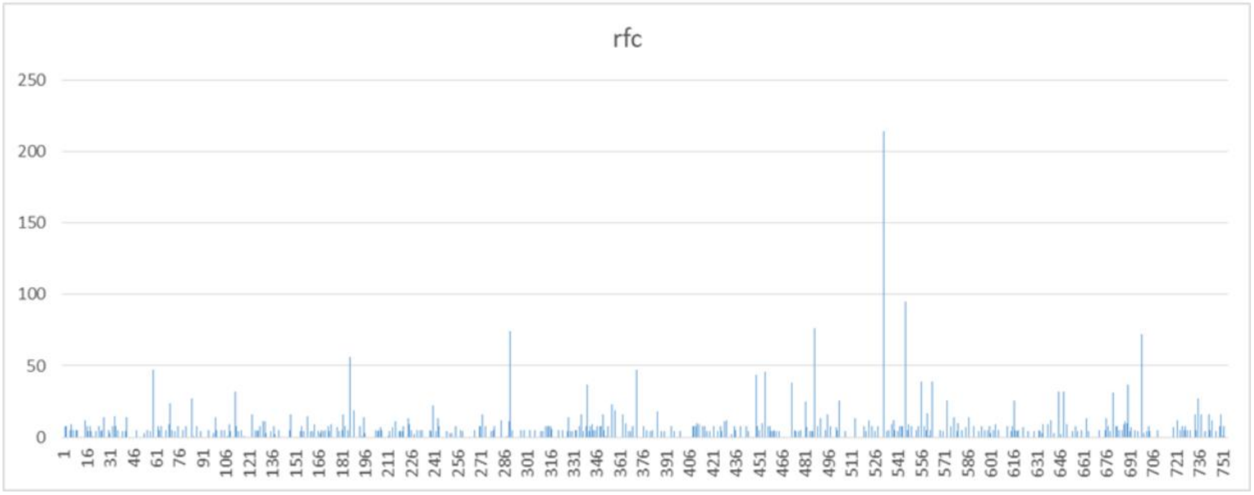
Project 3 : java-design-patterns:-



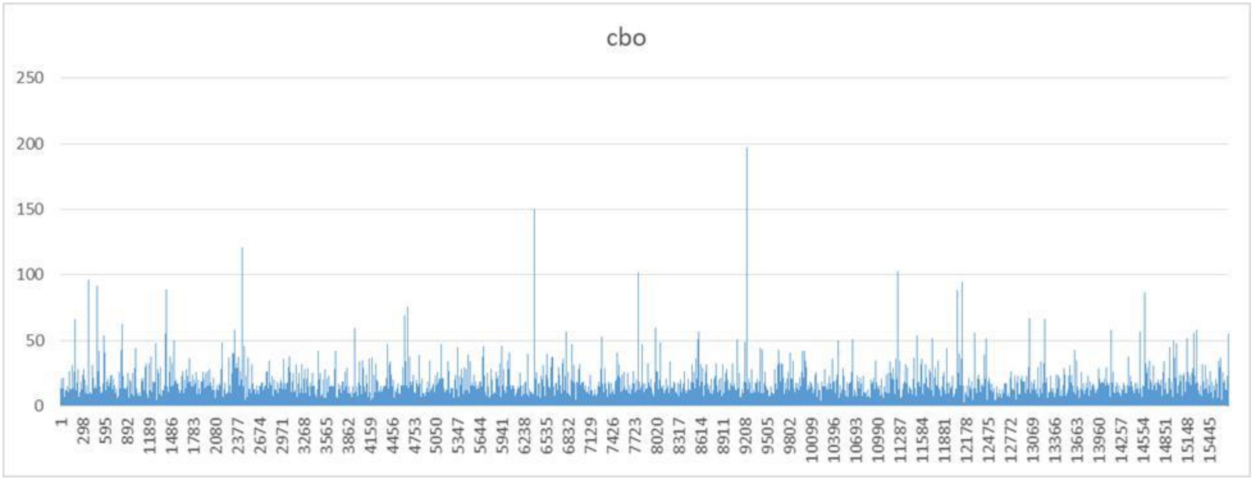


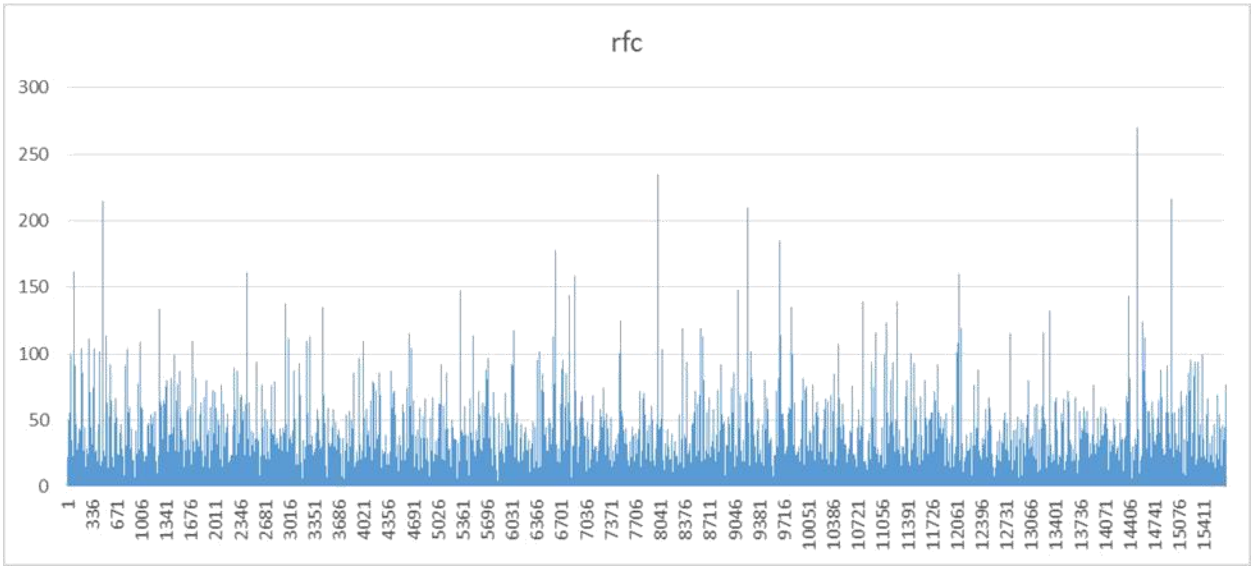
Project 4: mall





Project 5: spring-framework





Section 5:

Conclusions

In this empirical study, our objective was to explore the impact of code bad smells on the modularity of Java projects, utilizing specific metrics such as the number of inactive classes, project size, and a designated range of bad smells. The investigation involved a careful selection of programs based on these metrics, followed by a detailed analysis of the obtained measurements.

The results of our study revealed noteworthy insights into the relationship between code bad smells and modularity in the selected Java projects. The examination of metrics across different programs allowed us to discern patterns and trends that shed light on the potential influence of code bad smells on modularity.

The choice of metrics, including the number of inactive classes, project size, and a specified range of bad smells, proved to be appropriate for capturing aspects related to modularity. These metrics provided a comprehensive view of the projects, enabling us to identify classes with distinct characteristics that may indicate the presence of code bad smells affecting modularity.

Our analysis highlighted the importance of considering metrics related to code bad smells when assessing the modularity of Java projects. Classes exhibiting deviations from the norm in terms of these metrics may warrant closer inspection and potential refactoring to enhance modularity.

While our study contributes valuable insights into the impact of code bad smells on modularity, it is essential to acknowledge certain limitations. The selected metrics, while informative, may not capture all nuances of modularity, and further research could explore additional metrics for a more comprehensive assessment. Additionally, the generalizability of our findings may be constrained by the specific criteria used to select programs.

In conclusion, this study provides a foundation for understanding the intricate interplay between code bad smells and modularity in Java projects. The insights gained contribute to the ongoing discourse on software quality, emphasizing the importance of addressing code bad smells to enhance the modularity and overall maintainability of software systems. Future research endeavors may build upon these findings, exploring additional metrics and refining our understanding of the dynamic relationship between code quality and modularity in software development

Reference

Kurmangali, A., Rana, M. E., & Ab Rahman, W. N. W. (2022, March). Impact of Abstract Factory and Decorator Design Patterns on Software Maintainability: Empirical Evaluation using CK Metrics. In *2022 International Conference on Decision Aid Sciences and Applications (DASA)* (pp. 517-522). IEEE.

Rathee, A., & Chhabra, J. K. (2022). Metrics for reusability of java language components. *Journal of King Saud University-Computer and Information Sciences*, *34*(8), 5533-5551.

Komolov, S., Dlamini, G., Megha, S., & Mazzara, M. (2022). Towards Predicting Architectural Design Patterns: A Machine Learning Approach. *Computers*, *11*(10), 151.