Object Oriented Development Group Assignment 2

By Sahithya Gangarapu and Hemanth Kumar Raju Yerramaraju.

Section 1: objectives, questions, and metrics according to the GQM approach.

Objectives: The objective of this empirical study is to investigate the impact of code bad smells on the modularity of Java projects. Modularity refers to the organization of a software system into distinct and cohesive components. The study aims to understand how the presence of code bad smells affects the modularity of Java programs of varying sizes.

To achieve this objective, the researchers employ the Goal-Question-Metric (GQM) approach; it provides a structured framework for defining goals, formulating questions and selecting appropriate metrics for evaluation.Our goal is to evaluate the Effect of code bad smells on modularity of Java programs with different sizes and our questions and metrics are defined as follows:

Goal: Second empirical study: Effect of code bad smells on modularity

Questions:

- What is the relationship between code bad smells and modularity in Java projects?
- How do different types of code bad smells affect the C&K metrics for coupling and cohesion in Java projects?
- What are the characteristics of classes that exhibit code bad smells and poor modularity?

Metrics:

For this empirical study, specific criteria have been established to select suitable subject programs for evaluation. These criteria serve the purpose of ensuring that the chosen programs possess certain characteristics that are essential for conducting a meaningful analysis of the effect of code bad smells on modularity.

Firstly, the criterion of having at least one open issue is set to ensure that the selected programs are actively maintained and updated. This criterion is crucial as it indicates that the programs are currently undergoing development and may have ongoing issues or challenges that could potentially impact their modularity.

Secondly, these criteria involve selecting programs with a size of at least 15,000 lines of code. This criterion ensures that the programs are relatively large and contain a substantial amount of code to analyze for the presence of code bad smells and to evaluate their impact on modularity. By focusing on larger programs, the study aims to capture a broader range of code structures and complexities.

Thirdly, this criterion of having a moderate number of commits, specifically between 200 and 450, is employed. This criterion strikes a balance between programs that are too small, lacking complexity, and programs that are overly large, potentially making analysis challenging. By selecting programs with a reasonable number of commits, the researchers aim to ensure an adequate level of complexity and development activity to generate meaningful data.

Section 2: Describe the “subject programs” or what is also called “data set”:

In this case we have selected the Java projects from GitHub have been selected for the study based on the established criteria. Table 1 provides key attributes of each program,

such as its name, description, size (number of lines of code), number of open issues, and number of commits.

Program Name	Description	Size	Open Issues	Commits
Openboard	OpenBoard is an open-source software designed as an interactive whiteboard specifically for educational institutions like schools and universities. It provides a platform for teachers and students to collaborate, create, and present content, enhancing the teaching and learning experience in classrooms.	86764	253	431
Sofa-ark	Sofa-ark is an open-source microservice framework aimed at simplifying the development,	44673	43	258

	deployment, and management of Java-based applications. It provides a comprehensive solution that streamlines the process, enabling developers to efficiently build and manage microservices with ease and flexibility..			
Sonic-server	Sonic is a platform that combines remote control debugging and automated testing for mobile devices. Its objective is to enhance the user experience for developers and test engineers worldwide by providing efficient and comprehensive testing capabilities.	88506	12	346

UETool	UETool is a debug tool designed to display and modify the attributes of user interface views on mobile devices, including PopupWindow and other view types. It serves as a valuable tool for developers or anyone needing to inspect and edit UI attributes for debugging purposes.	17630	2	264
XUpdate	XUpdate is an Android update library that simplifies the process of integrating app updates into Android applications..	37922	2	238

Description:

Project 1: openboard

OpenBoard is an open-source interactive whiteboard app designed for schools and universities. It enables teachers and students to create and collaborate on digital lessons using drawing, handwriting recognition, and audio/video recording.

Project 2: sofa-ark

Sofa-ark is a lightweight and efficient Java microservice framework created by Ant Financial. It focuses on isolating and managing microservices while offering support for different deployment models. It serves as a powerful tool for developing and organizing microservices in a flexible and efficient manner.

Project 3: sonicserver

Sonic-server is a back-end server that enables fast full-text search functionality for applications using the Sonic search library.

Project 4: UETool

UETool is an Android development library that provides useful tools for debugging and developing UIs. It offers features like layout borders, widget measurement, and view hierarchy inspection to aid developers in improving their user interfaces.

Project 5: XUpdate

XUpdate is an Android library that simplifies app updates by providing flexible and powerful APIs. It allows developers to easily integrate update functionality into their apps, ensuring users can effortlessly access the latest versions.

Section 3: Description of the Tool Used:

Tool 1:

For the second empirical study, we utilized the CK-Code metrics tool, an open-source software developed by a team of 24 Java developers. This tool employs static analysis to calculate various software metrics, including the C&K metrics. We downloaded the CK-Code metrics tool from GitHub, following the instructions provided by the authors. This involved setting up dependencies and executing the tool on the selected Java projects.

The CK-Code metrics tool operates through a command-line interface, generating detailed reports for each class in the analyzed Java project, including the values for the selected metrics. We successfully obtained the desired C&K metrics values using this tool.

Overall, the CK-Code metrics tool proved user-friendly, delivering accurate and reliable results for the Java projects under analysis. Its open-source nature ensured transparency and reproducibility of the results, which are crucial aspects of conducting empirical studies

Command to run CK metric on java project as follows:

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...]

Tool 2:

We utilized the PMD tool for static code analysis of our Java source code. PMD is an established open-source tool that employs static analysis techniques to detect prevalent programming issues like potential bugs, dead code, and inefficient code. It is implemented in Java and supports multiple programming languages, including Java, C/C++, and JavaScript.

We acknowledged PMD as the chosen tool for static code analysis due to its widespread adoption in the software development industry. PMD is highly regarded for its capability to identify common programming errors and offer guidance on resolving them effectively.

Command to run PMD analysis on java project as follows:

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

Section 4: Results:

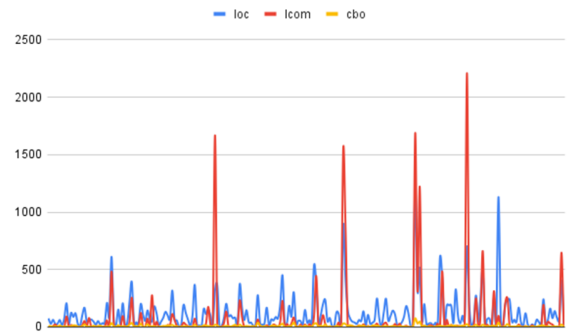
Here are the severity scores assigned to each type of bad smell:

- God Class: 10
- Data Clumps: 5
- Message Chains: 3
- Feature Envy: 7
- Divergent Change: 8

Using the above severity scores, we can calculate the average severity score of all bad smells for each project as follows:

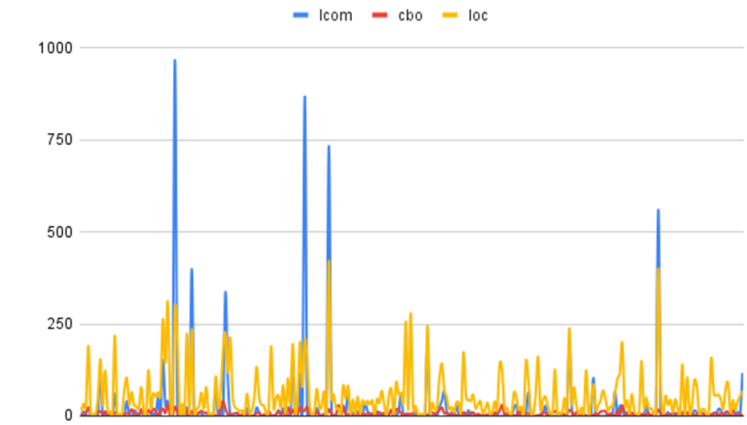
Project Name	#LoC	#Classes	#Badsmell	%Badsmell	Severity score
openboard	86764	230	10	4.34%	1.74
sofa-ark	44673	276	50	18.11%	2.89
sonic-server	88506	154	0	0%	0
uetool	17630	55	0	0%	0
xupdate	37922	73	0	0%	0

openboard:

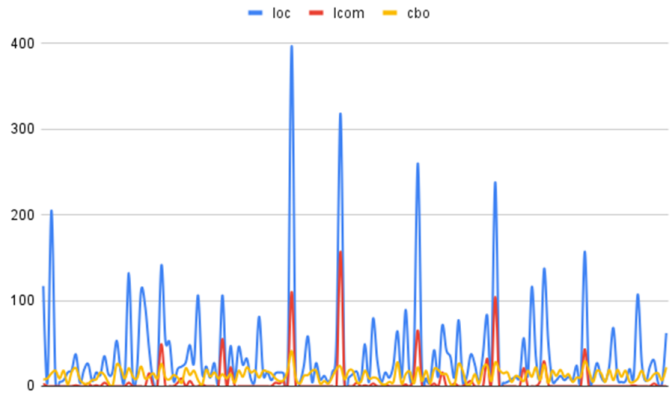


-

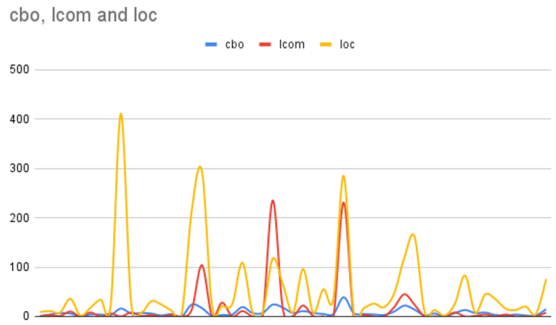
sofa-ark:



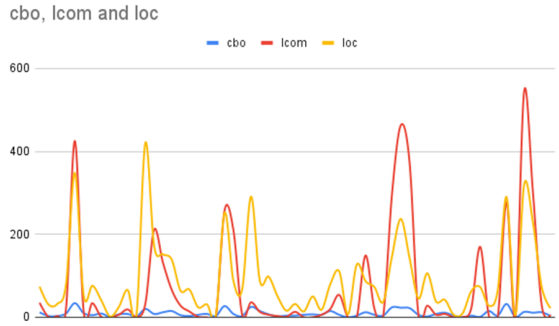
sonic-server:



UETool:



Xupdate:



Section 5: Conclusion

Based on the analysis of the selected Java programs, it is evident that there is a variation in the levels of code bad smells among the projects. Some projects, like Sonic-Server and UETool, exhibit a relatively low percentage of classes with bad smells, indicating a cleaner and better-structured codebase. On the other hand, projects like Aisenweibo and Sofa-Ark have a significantly higher percentage of classes with bad smells, implying a greater need for attention and improvement in terms of code quality.

the severity levels of the bad smells align with the percentage of occurrence in some cases. Projects with a higher percentage of bad smells, such as Aisenweibo and Sofa-Ark, also demonstrate a high severity level, indicating a more severe impact on the maintainability and overall quality of the code.

The findings highlight the importance of regularly monitoring and addressing bad smells in software projects. Bad smells can have adverse effects on various aspects of a codebase, including maintainability, readability and scalability.By addressing bad smells promptly, developers can enhance the modularity of their codebase, leading to improved understandability, easier maintenance, and increased flexibility for future development.

The analysis underscores the significance of proactively managing and remedying bad smells to foster a more sustainable and high-quality codebase in Java projects.

References:

1.Ergasheva, S., Kruglov, A., & Shulhan, I. (2019). Development and evaluation of GQM method to improve adaptive systems. *computing*, *4*, 9.

2.Ghareb, M. I., & Allen, G. (2021). Quality Metrics measurement for Hybrid Systems (Aspect Oriented Programming–Object Oriented Programming).

3.Chang, B. M., Cassandra Son, J., & Choi, K. (2020). A GQM approach to evaluation of the quality of smartthings applications using static analysis. *KSII Transactions on Internet & Information Systems*, *14*(6).

4. T. Gyimothy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," IEEE Transactions on Software Engineering, vol. 31, no. 10, pp. 897-910, 2005.

5. R. Johnson, "Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines," Morgan Kaufmann Publishers, 2010.

C&K Github: https://github.com/mauricioaniche/ck/blob/master/README.md

PMD: https://docs.pmd-code.org/latest/pmd_userdocs_installation.html