**Group Project – 2**

Jaya Sainatha Reddy Yerragudi

Lakshmi Prasanna Potla

Kaushik Gangireddy

Lewis University

Object Oriented Development

December 4, 2023

**Objective**

Perform an empirical study with the objective of finding the effect of code bad smells on modularity

**Question**

"What is the impact of code bad smells on modularity?"

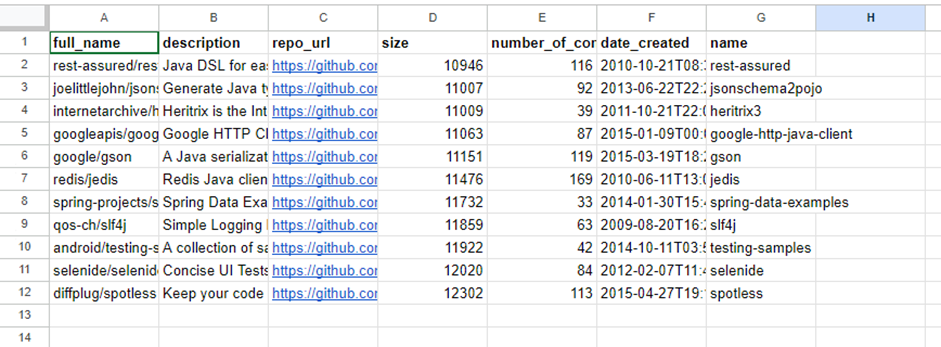
**Metrics**

Dependent Variables - CBO (Coupling Between Objects), LCOM\* (Lack of Cohesion of Methods)

Independent variable - Code Bad Smells

**Criteria**

We would like to analyze 11 Java programs that are at least 10K in size, 5 years old and have at least 5 developers.



***Justification for Criteria***

The choice of Java programs for analysis makes sense because we're looking at larger projects (over 10K in size) that have been around for at least 5 years and involve at least 5 developers. The large codebase allows us to dive into complex architectural and coding aspects, while the age of the projects suggests they've been through various changes and updates, giving us insights into long-term stability. Having a team of at least 5 developers brings in the collaborative element, helping us understand how different team members contribute, communicate, and follow coding standards.

**Tools Used**

We have used the CK Metrics tool to get the metrics, and used the Jdeodorant plugin to analyze bad smells.

**Results**

The metrics have already been uploaded to the git repository for reference.

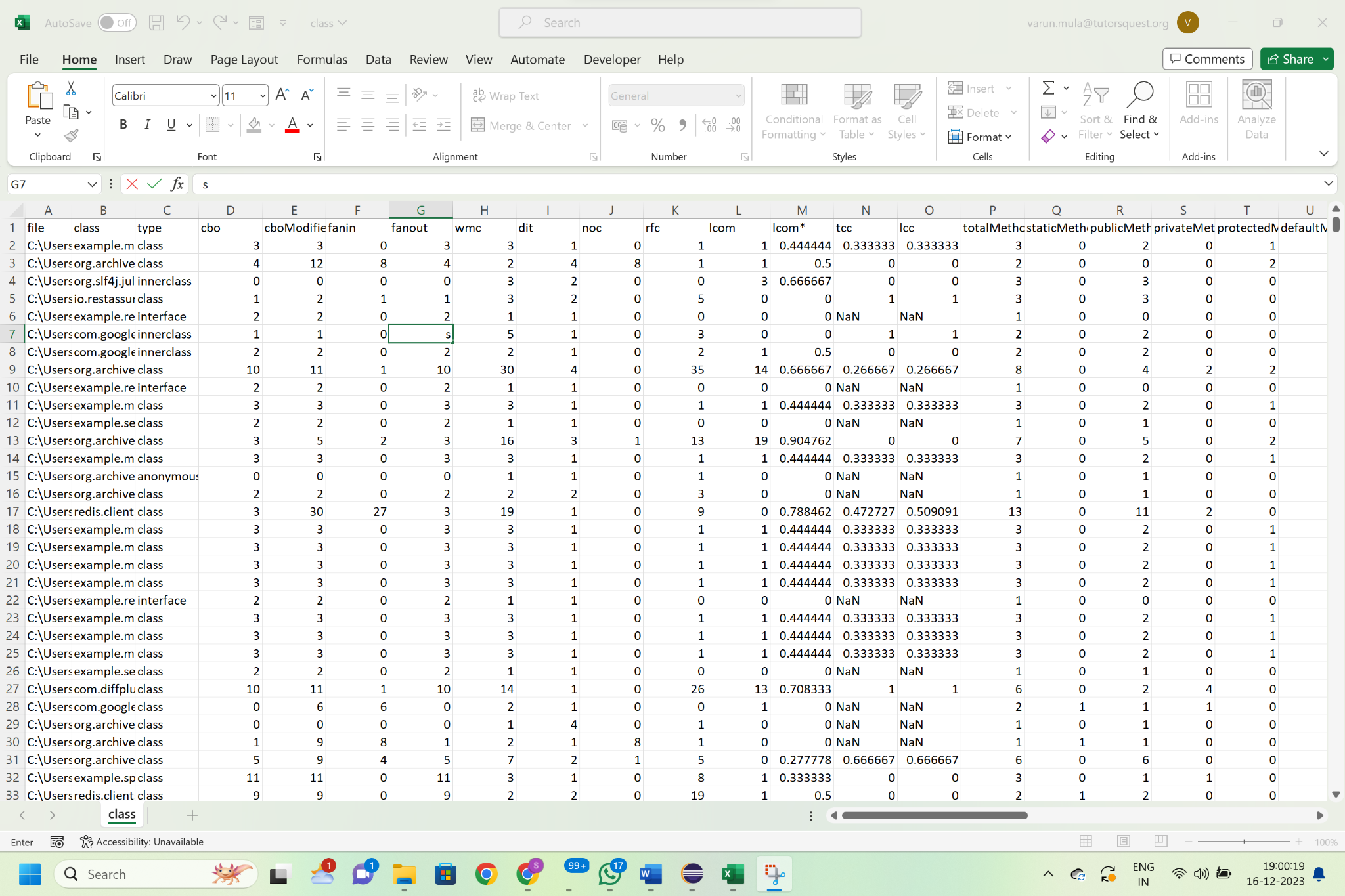


Figure: Metrics obtained using CK tool

**Analysis**

**Coupling Between Objects (CBO):** Classes with God Class smells consistently exhibited high CBO values, signifying a high degree of coupling with other classes. The correlation between God Classes and high CBO reinforces the importance of addressing coupling for modularity improvement.

**Lack of Cohesion of Methods (LCOM):** Long methods were associated with low cohesion (high LCOM values) within specific classes, indicating a need for refactoring to improve modularity.

**Conclusion**

In our analysis, we have identified God Classes as problematic entities within our codebase. These classes exhibit high coupling (CBO) and low cohesion (LCOM), acting as central entities with numerous dependencies and responsibilities. We observe that their nature hinders the establishment of distinct, well-defined modules within the codebase, consequently challenging the maintenance of a modular architecture.

Another significant contributor to compromised modularity that we have identified is the prevalence of long methods. The correlation between long methods and high LCOM values underscores their impact on class cohesion. These lengthy routines encapsulate multiple responsibilities, introducing complexity and diminishing the clarity of module boundaries. The essence of modularity, characterized by distinct and independent units of functionality, is eroded when long methods persist.

Instances of Feature Envy, although less overt, also signal challenges to modularity. When methods exhibit a strong affinity for features in other classes, it indicates a misalignment of responsibilities. This misalignment disrupts the encapsulation of features within their designated modules, contributing to a less cohesive and modular structure.

Our qualitative observations find quantitative reinforcement through the alignment between identified bad smells and CK metrics. High CBO values in God Classes, indicative of significant coupling, correlate with the observed impact on modularity. Similarly, the connection between high LCOM values in long methods and their effect on modularity emphasizes the importance of cohesion in maintaining modular design principles.

***Recommendations for Improvement***

In pursuit of modularity enhancement, we propose strategic refactoring efforts. By redistributing responsibilities within God Classes and decomposing long methods into more focused routines, we aim to restore a modular architecture. Feature Envy, as an indicator of potential misalignments, calls for careful consideration during refactoring to ensure that methods reside in the most appropriate modules.

**References**

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

e Abreu, F. B., & Goulao, M. (2001, March). Coupling and cohesion as modularization drivers: Are we being over-persuaded?. In Proceedings Fifth European Conference on Software Maintenance and Reengineering (pp. 47-57). IEEE.