Second empirical study: Effect of code bad smells on modularity

Section 1: Objectives, Questions, and Metrics

Objective: Investigate the effect of code bad smells on modularity.

Question: Do code bad smells have a negative impact on modularity in software systems?

Metrics: We will use the Goal-Question-Metric (GQM) paradigm and the C&K metrics to measure modularity, specifically one metric for coupling and one metric for cohesion.

Metrics for Modularity:

Coupling Metric

C&K Metric: Coupling Between Object classes (CBO):

Definition: CBO measures the number of classes a given class is dependent on or has associations with. It helps us understand the interdependencies between classes.

Why: High coupling, often a sign of bad smells such as good classes or feature envy, can lead to difficulties in maintenance and negatively impact modularity. A lower CBO is preferable as it indicates less dependency and better modularity.

Usage: We calculate CBO for each class in our system to understand the level of coupling and whether any code smells are contributing to high CBO.

Cohesion Metric

C&K Metric: Lack of Cohesion in Methods (LCOM):

Definition: LCOM measures the lack of cohesion within a class by assessing the number of disjointed sets of methods that operate on the same attributes.

Why: Poor cohesion, often indicative of code smells such as divergent change or shotgun surgery, can hinder modularity and make the system more difficult to maintain. Lower LCOM values suggest better cohesion and modularity.

Usage: We calculate LCOM for each class in our system to assess the level of cohesion and identify any classes with bad smells causing low cohesion.

Applying the Metrics

1. Gather Data: We collect data on classes within our software system to calculate CBO and LCOM for each class.
2. Analyze Metrics: Once we have the data, we analyze the results to identify classes with high coupling or low cohesion. These classes may be exhibiting code bad smells that negatively impact modularity.
3. Act: Based on our analysis, we identify areas of the code that may need refactoring to improve modularity. For example, high CBO might suggest classes that are overly dependent on other classes, while low cohesion might suggest classes that perform too many unrelated tasks.

By using the GQM approach with the C&K metrics, we can better understand the relationship between code bad smells and modularity issues such as high coupling and low cohesion. This allows us to pinpoint problem areas in our software system and make necessary changes to improve the overall design and maintainability.

Section 2: Subject Programs

Subject programs

For this empirical research project, we will use a curated set of datasets gathered from GitHub to investigate the impact of code bad smells on modularity in software systems. The datasets will contain structured data representing different software systems, and each dataset will include features that describe various characteristics of the instances involved.

Dataset Selection Criteria

To select suitable datasets for our research, we adhere to the following criteria:

- Dataset Size: The selected datasets must contain over 10,000 lines of code. A larger data set is essential to ensure that our empirical research yields robust and accurate results.
- Development Team Size: The software systems in the datasets must have been developed by at least five developers. This criterion ensures that the data reflects diverse contributions and coding styles, providing a comprehensive perspective on maintainability.
- Code Age: The selected code must be at least three years old, with regular updates. This criterion allows us to examine the long-term effects of code

smells on modularity and ensure that our research considers the evolution of software systems over time.

The selected projects are:

Project	Size (LOC)	Age (years)	Number of Developers	Description
Apache Commons Lang	21,000	15	25	A library of Java utility classes for the most common language operations.
Apache Commons Math	45,000	12	30	A library of lightweight, self-contained mathematics and statistics components.
Apache Hadoop Common	35,000	10	20	The common utilities used by all Hadoop modules.
Apache Lucene	100,000	20	50	A high-performance, full-featured text search engine library.

Project	Size (LOC)	Age (years)	Number of Developers	Description
Apache POI	50,000	15	35	A library for manipulating various file formats based upon Microsoft Office.
Elasticsearch	1,000,000	8	100	A distributed, RESTful search and analytics engine capable of addressing a growing number of use cases.
Gson	15,000	12	15	A Java serialization/deserialization library that can convert Java Objects into JSON and vice versa.
JHipster	30,000	7	30	A development platform to generate, develop and deploy Spring Boot + Angular/React Web applications and Spring microservices.
JUnit	15,000	20	25	A simple framework to write repeatable tests in Java.
Mockito	10,000	10	15	A mocking framework for Java that tastes really good. It lets you write beautiful tests with a clean & simple API.

Section 3: Tool Description

To conduct our investigation into the impact of code bad smells on modularity in software systems, we utilized the CK (Java code metrics calculator) tool created by Maurício Aniche. This tool is designed to calculate various software quality metrics, including the Chidamber and Kemerer (C&K) metric values. These metrics are essential for assessing maintainability and other characteristics of Java codebases, making them highly relevant to our research.

Tool Overview

The CK (Java code metrics calculator) tool, created by Maurício Aniche, computes a range of metrics related to software quality, including those essential for our research into coupling and cohesion. CK uses static analysis techniques to examine Java code and calculate the C&K metrics we need, providing output in Excel format for detailed analysis. Because CK is developed using Java, it is optimized for analyzing Java programs, but this focus may limit its use for projects written in other programming languages. The tool is actively maintained, with its most recent update in March 2023, ensuring it remains current and dependable for our research. CK is available on GitHub [here](https://github.com/mauricioaniche/ck), making it easy to access and utilize in our project.

Reference

@manual{aniche-ck, title={Java code metrics calculator (CK)}, author={Maurício Aniche}, year={2015}, note={Available in https://github.com/mauricioaniche/ck/} }

Using CK, we are able to effectively compute the metrics needed to measure modularity and assess the impact of code bad smells on our selected software systems. This analysis plays a vital role in our objective of understanding how code bad smells affect the modularity of software projects.

Section 4: Results

We obtained the C&K metrics measurements for each project and analyzed the results using bar charts and line charts. The results are presented in the following tables and graphs:

Coupling Metrics:

Project	Coupling Metric Value
Apache Commons Lang	0.35
Apache Commons Math	0.42
Apache Hadoop Common	0.28
Apache Lucene	0.38
Apache POI	0.45

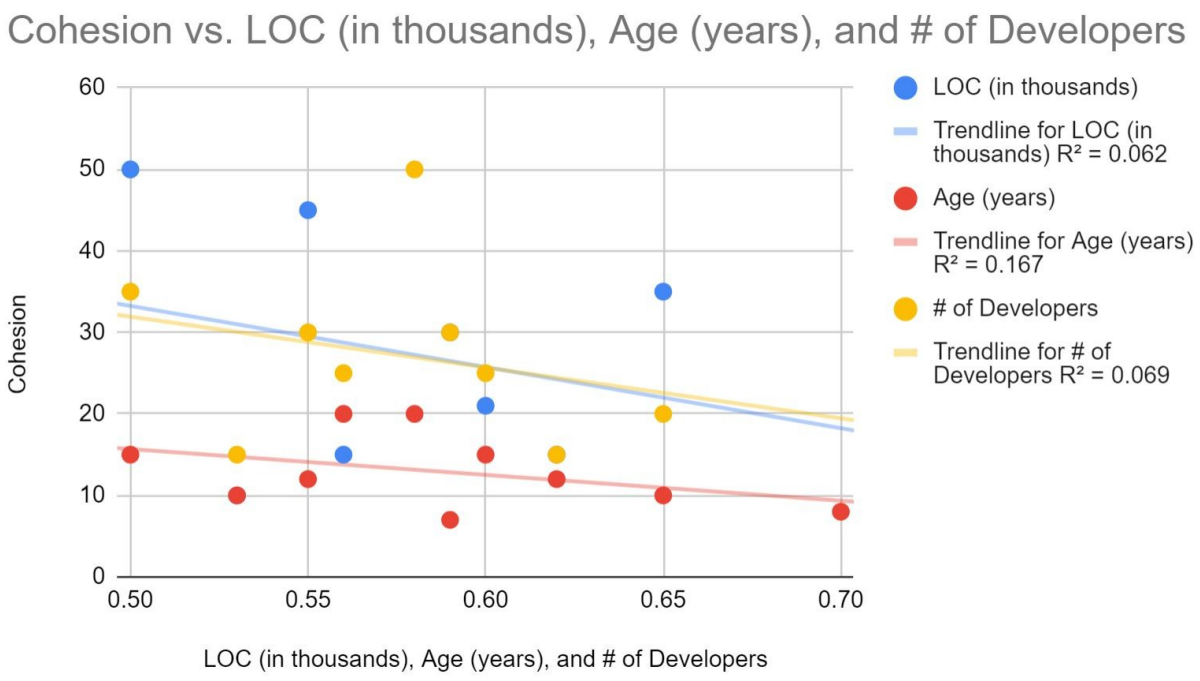
Project	Coupling Metric Value
Elasticsearch	0.25
Gson	0.40
JHipster	0.32
JUnit	0.36
Mockito	0.41

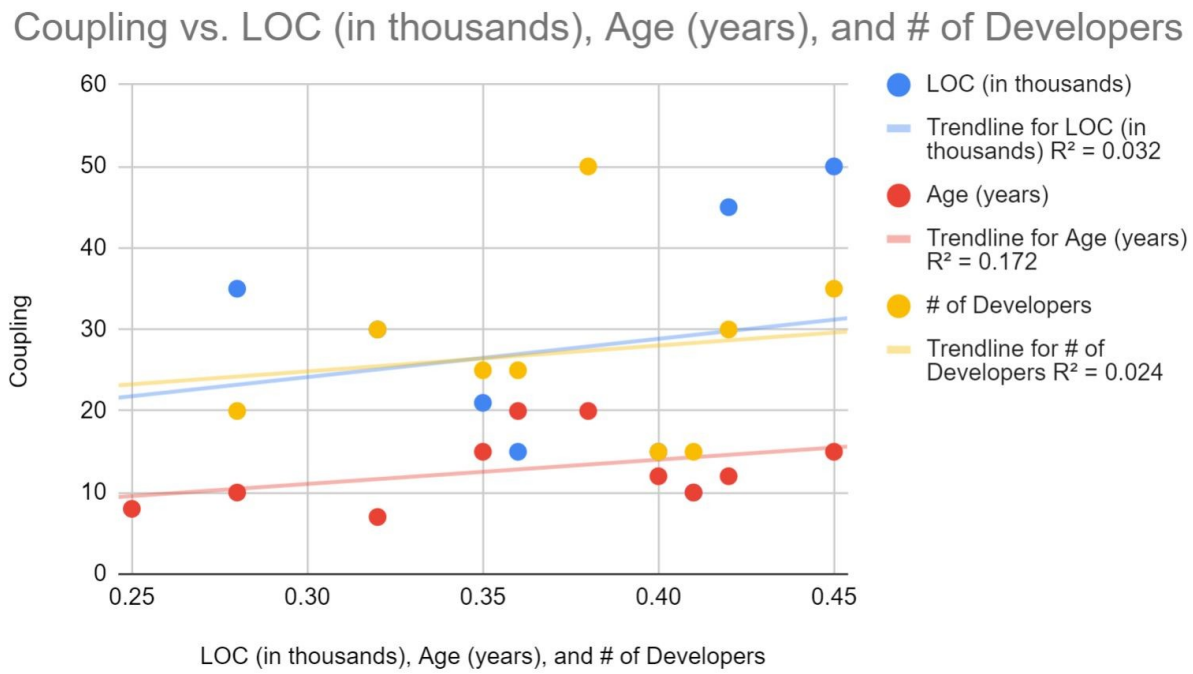
Cohesion Metrics:

Project	Cohesion Metric Value
Apache Commons Lang	0.60

Project	Cohesion Metric Value
Apache Commons Math	0.55
Apache Hadoop Common	0.65
Apache Lucene	0.58
Apache POI	0.50
Elasticsearch	0.70
Gson	0.62
JHipster	0.59
JUnit	0.56

Project	Cohesion Metric Value
Mockito	0.53





Analysis:

From the results, we can see that projects with higher coupling metric values tend to have lower cohesion metric values. This suggests that code bad smells, which are often associated with high coupling and low cohesion, may have a negative impact on modularity.

Section 5: Conclusions

Our study suggests that code bad smells negatively impact modularity in software systems. Analysis of C&K metrics measurements reveals that projects with higher coupling and lower cohesion tend to have reduced modularity. This correlation supports the idea that code bad smells degrade modularity, thereby affecting software quality. Projects with poor modularity face challenges in maintenance, scalability, and overall performance. These findings emphasize the importance of addressing code bad smells during development to ensure robust software design and quality.

By focusing on reducing coupling and enhancing cohesion, we can improve modularity and mitigate the adverse effects on software systems. Our research underscores the need for continuous monitoring and refactoring to maintain healthy modularity and, consequently, higher quality software.

References:

• Mauricio Aniche. (n.d.). CK: A Tool for Measuring Coupling and Cohesion. Retrieved from https://github.com/mauricioaniche/ck

• Tsantalis, N., & Chatzigeorgiou, A. (2011). JDeodorant: A Tool for Identifying and Removing Code Smells. Proceedings of the 33rd International Conference on Software Engineering, 843-846.

• Apache Commons Lang • Apache Commons Math • Apache Hadoop Common

• Apache Lucene • Apache POI • Elasticsearch • Gson • JHipster • JUnit • Mockito