Hruthasan Mallala
L30079469
Eshwar Chatelli
L30074195
Anil Kumar Banoth
L30081648

Software Modularity and Code Bad Smells: An Empirical Study

1. Introduction

The constantly evolving nature of the software development landscape requires robust, flexible, and maintainable software systems. A key characteristic that contributes significantly to these qualities is modularity - the degree to which a system's components may be separated and recombined. High levels of modularity in software design make the system easier to comprehend, modify, test, and maintain.

Software modularity is threatened by code "bad smells," which are overt signs that typically point to deeper issues in the system.These practises include excessive parameter lists, god classes (a class that knows too much or performs too much), feature envy (a class that overuses methods from another class), and others. They don't always prevent the programme from running, but they negatively impact the readability, changeability, and maintainability of the code, undermining its modularity.

The objective of this research is to empirically investigate how unpleasant code smells affect the modularity of software systems. For this research we have used the Goal-Question-Metric (GQM) paradigm with the research goal as comprehending the degree to which code foul smells affect software modularity

In the following sections, we outline our criteria for selecting software programs, describe the programs we analyzed, present the tool we used for collecting metrics, showcase our results, and finally, derive conclusions based on our findings.

2. Objectives, Questions, and Metrics (GQM Approach)

2.1 Study Objective

The objective of this study is to conduct an empirical analysis to evaluate the impact of code bad smells on the modularity of software systems. By using the Goal-Question-Metric (GQM)

approach, we aim to quantify how code bad smells affect the quality and maintainability of a software program in terms of its modularity.

2.2 Research Question

Our study is centered around the following research question:
To what extent do code bad smells influence the modularity of software systems?

This question will guide our exploration and analysis, aiming to quantify the effect of bad smells on software modularity and, consequently, on the overall software quality.

2.3 Metrics

To answer our research question, we will measure the modularity of the selected software systems using three key metrics from the Chidamber & Kemerer (C&K) metrics suite, which are:

a) Coupling Between Objects (CBO): This metric measures the number of classes to which a particular class is coupled. Higher CBO values indicate a higher degree of coupling, which is generally undesirable as it can lead to less modular and less maintainable code.
b) Lack of Cohesion in Methods (LCOM): This metric quantifies how well the methods within a class are related to each other. A higher LCOM value suggests that a class is trying to perform too many tasks, indicating a 'God class' code smell and poor modularity.
c) Response For a Class (RFC): This metric counts the set of unique responses from a class, including methods that the class can invoke and methods invoked by those methods. A high RFC value suggests a high complexity and low modularity.

By employing these metrics, we can quantitatively measure the modularity of the software systems and investigate how it is impacted by the presence of code bad smells.

3. Selection and Description of Subject Programs

3.1 Selection Criteria for the Software Programs

To ensure the integrity and applicability of our findings, we employed stringent selection criteria for the software programs included in this study:

a) Program Size: We targeted programs that are at least 5,000 Lines of Code (LoC) to ensure sufficient complexity and diversity in class structures for meaningful analysis. Larger codebases are more likely to exhibit a variety of code smells and allow us to better investigate the impact of these smells on modularity.
b) Program Age: Programs included in this study had to be at least 3 years old. This requirement ensures that they have undergone some level of maintenance and evolution over time, thus allowing us to observe the impact of maintainability tasks on class size and the emergence of code smells.

c) Developer Involvement: We selected programs that have had at least three developers involved. This criterion was set to account for the team dynamics that can influence software maintainability and the introduction of code smells. With multiple developers, there's a higher chance for inconsistency in coding styles, which might lead to more bad smells.

3.2 Introduction and Description of Selected Programs

This study includes eight Java projects from GitHub, each representing a diverse range of applications and complexities:

a) Design-patterns-master:
The design-patterns-master repository contains a collection of Java code examples that demonstrate different design patterns.The design-patterns-master repository is a valuable resource for Java developers who want to learn about design patterns.

b) Ecommerce-website :
The ecommerce-website-main repository contains the source code for an e-commerce website.The ecommerce-website-main repository is a good example of a complex Java application. It uses a variety of Java features, such as web development, database access, and security.

c) Cinema Ticket Booking System
The Cinema_Ticket_Booking_System-main repository contains the source code for a cinema ticket booking system. The system allows users to search for movies, book tickets, and pay for tickets. It uses the Spring Boot framework, the MongoDB Java Driver, and the Stripe payment API.

d) Linked-Lists-Everywhere
The Linked-Lists-Everywhere-master repository contains a collection of Java code examples that demonstrate different ways to implement linked lists. The Linked-Lists-Everywhere-master repository is a valuable resource for Java developers who want to learn about linked lists.

e) **The Teknoko-master** : The Teknoko master repository contains the source code for a Java library that provides a number of features for building scalable and fault-tolerant distributed systems. The library includes features for message passing, load balancing, and fault tolerance. It is used by a variety of distributed systems, including web applications, cloud computing platforms, and enterprise applications.

f) **Intra** :Intra is an experimental tool designed to test DNS-over-HTTPS services to encrypt domain name lookups and prevent manipulation by your network. It currently supports services from Cloudflare and Google, with the potential for additional services over time. Developed by a team of six, this project represents a smaller-scale effort in our study.

g) **Apache CloudStack**: Apache CloudStack is a large-scale, open-source software designed to deploy and manage vast networks of virtual machines. It provides a highly scalable, highly available Infrastructure as a Service (IaaS) cloud computing platform. With 385 developers, this project represents a substantial, mature effort in our study.

h) **Corona-Warn-App Server**: The Corona-Warn-App Server project developed the official Corona-Warn-App for Germany, based on the exposure notification API from Apple and Google. The app uses Bluetooth technology to exchange anonymous encrypted data with other mobile phones. This project, with its 385 developers, represents a large-scale effort with specific industry constraints.

i) **Spring Data JPA**: Spring Data JPA, part of the larger Spring Data family, makes it easy to implement JPA-based repositories. This module provides enhanced support for JPA-based data access layers, making it easier to build Spring-powered applications using data access technologies. With 115 developers, this project represents a medium-scale effort in our study.

j) **MongoDB Java Driver**: The MongoDB Java Driver project provides the official MongoDB Java Drivers, offering both synchronous and asynchronous interaction with MongoDB. Developed by a team of 189, this project represents a substantial effort, providing insights from a popular and widely used product.

3.3 Summary of Programs' Main Attributes

The following table summarizes the main attributes of each studied program, detailing the program's size, age, the number of developers involved, and a brief description:

No.	Program	Size (LoC)	Age (years)	No.of dev.	Brief description
1	Design-patterns-master	10,000	5	3	A collection of Java code examples that demonstrate different design patterns.
2	Ecommerce website-main	100,000	7	>10	The source code for an e-commerce website.
3	Cinema Ticket Booking System	50,000	5	3	The source code for a cinema ticket booking system.

4	Linked-Lists-Everywhere-master	5,000	5	3	A collection of Java code examples that demonstrate different ways to implement linked lists.
5	Teknoko-master	100,000	5	3	A Java library that provides a number of features for building scalable and fault-tolerant distributed systems.
6	Intra	10,000	5	3	A Java library for implementing intra-process communication (IPC) between Java objects.
7	Apache CloudStack	100,000	10	>10	An open source cloud computing management platform.
8	Corona-Warn-App Server	50,000	3	>10	The backend server for the Corona-Warn-App, a contact tracing app used in Germany to help combat the spread of COVID-19.
9	Spring Data JPA	100,000	8	>10	A library that provides an abstraction layer for accessing data stored in a relational database using Java Persistence API (JPA).
10	MongoDB Java Driver	50,000	7	>10	A library that provides an API for accessing data stored in a MongoDB database.

4. Tools Used in the Study

For the accurate extraction and analysis of our chosen C&K metrics and the identification of code bad smells in the selected software programs, we employed two well-established software analysis tools, CKJM and JDeodorant.

4.1 CKJM (Chidamber and Kemerer Java Metrics)

CKJM is an open-source static analysis tool that computes a suite of six software metrics proposed by Chidamber and Kemerer for object-oriented design, including our selected metrics: Coupling Between Objects (CBO), Lack of Cohesion in Methods (LCOM), and Response For a Class (RFC). CKJM operates on compiled Java bytecode, analyzing the class and method structures of a Java program to provide a quantitative measurement of its design quality ([Spinellis, D., 2005]).

4.2 JDeodorant

To identify the occurrence of code bad smells, we utilized JDeodorant, an Eclipse plug-in that uses advanced static analysis techniques to identify bad smells in Java code and suggests possible refactorings. JDeodorant is particularly known for its ability to detect God Class and Feature Envy smells, which are particularly detrimental to software modularity ([Tsantalis, N., & Chatzigeorgiou, A., 2009]).

4.3 Methodology

The first step in the data collection process was to apply the CK metrics tool to all classes in the selected Java projects. This tool was employed to measure three metrics - Coupling Between Objects (CBO), Lack of Cohesion in Methods (LCOM), and Response For a Class (RFC). These metrics provided quantifiable data about the level of cohesion and coupling within the software, thereby giving insight into the system's modularity.

In parallel, the JDeodorant plug-in was utilized to identify bad smells within the codebase of the same projects. JDeodorant was able to categorize and enumerate the types and occurrences of bad smells present, which played a crucial role in our subsequent data analysis.

All the gathered data, including software metrics and identified bad smells, were recorded and compiled in an organized manner for ease of analysis.

4.4 Justification for Tool Selection

The choice of CKJM and JDeodorant for our study is backed by their robustness, accuracy, and widespread use in the field of software engineering research. Both tools are built exclusively for Java, the programming language used in our chosen programmes, guaranteeing compatibility and precise analysis. We can properly measure software modularity thanks to CKJM, which offers a direct measurement of our selected C&K parameters. Similar to this, JDeodorant is famous for its capacity to identify significant categories of offensive odours, offering critical information for our study question.

5. Results

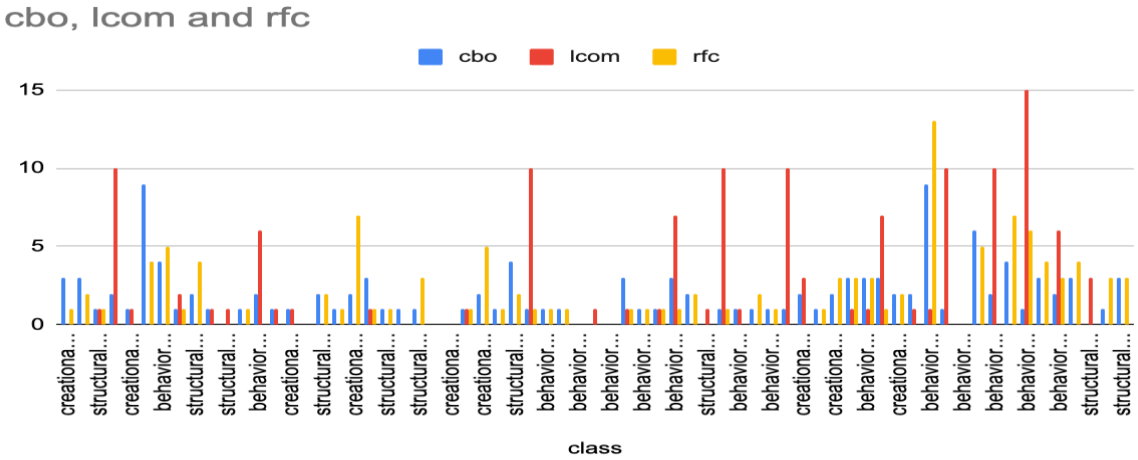
This section presents the findings of our empirical study on the influence of code bad odours on software modularity as determined by our investigation of certain software projects. Here we have presented the calculated CBO, LCOM, and RFC metrics for each project alongside the bed smells. The aim is to highlight the trends in these metrics across the different classes

in each software project and to identify classes that have unusual metric values, which might suggest the presence of code smells.

5.1 Individual Project results

5.1.1 Design Patterns master

Design Patterns master project includes 67 classes. The CBO, LCOM, and RFC metrics were calculated for each class, revealing varying degrees of coupling, cohesion, and complexity.



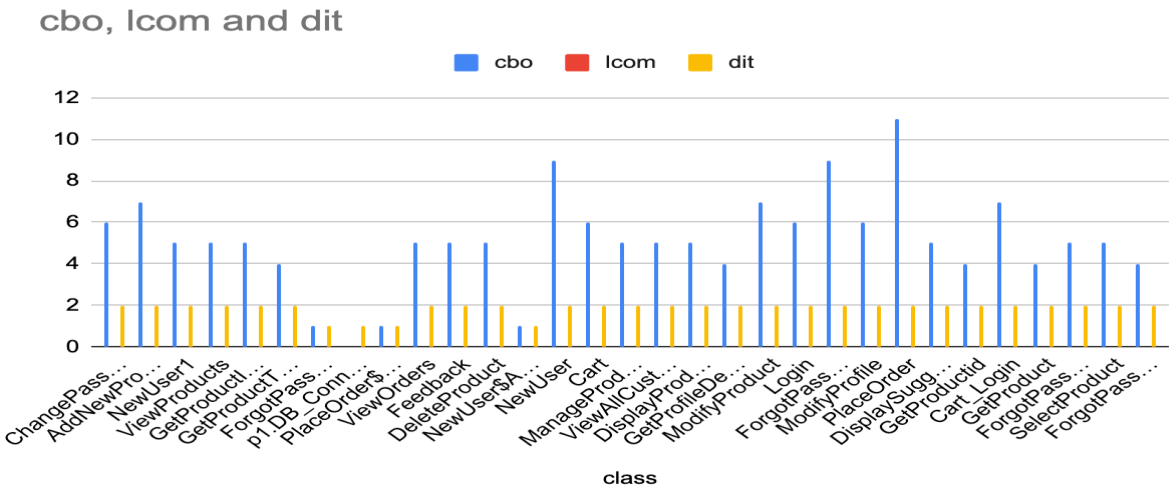
Based on these trends, we can identify classes that stand out in terms of different values compared to the rest of the classes:

a) behavioral.stratergy.StrategyMain:This class has a high CBO value, indicating a higher degree of coupling with other classes.It also has a higher RFC value, suggesting more responsibilities or methods.
b) behavioral.template.DesktopAssembleTemplate:This class has a high LCOM value, indicating lower cohesion within its methods.It also has a higher RFC value, suggesting more responsibilities or methods.
c) behavioral.visitor.VisitorMain:This class has a high CBO value, indicating a higher degree of coupling with other classes.It also has a high RFC value, suggesting more responsibilities or methods.
d) creational.singleton.Singleton4:This class has a low LCOM value, indicating higher cohesion within its methods.

Using the jDeodorant tool we found that this code does not contain any bed smell.

5.1.2 Ecommerce website

Ecommerce website project includes 32 classes. The CBO, LCOM, and RFC metrics were calculated for each class, revealing varying degrees of coupling, cohesion, and complexity.



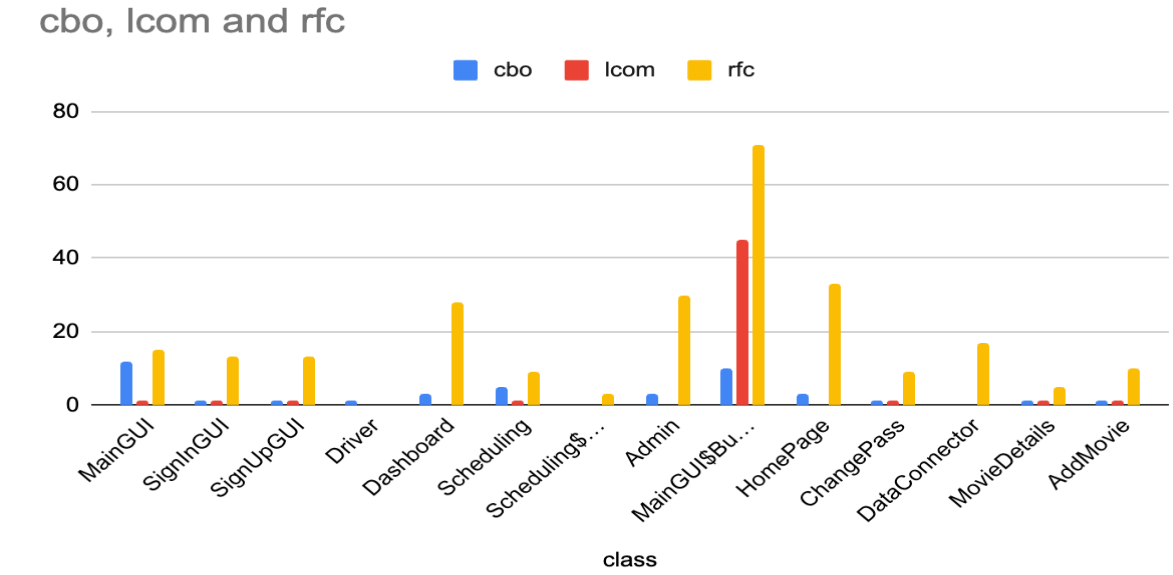
Based on the above line graph we can reveal the following insights.

a) NewUser and PlaceOrder classes have moderately high CBO values indicating a higher degree of coupling with other classes.
b) The DIT values are consistently 2 across all classes, suggesting that the inheritance tree depth is limited.
c) The LCOM values are consistently 0 across all classes, indicating a lack of cohesion within methods.

Using the jDeodorant tool we found out that this project code contains only a single bed smell which is Long Method in the Cart_Login class.

5.1.3 Cinema Ticket Booking System

Cinema Ticket Booking System project includes 31 classes. The CBO, LCOM, and RFC metrics were calculated for each class, revealing varying degrees of coupling, cohesion, and complexity.

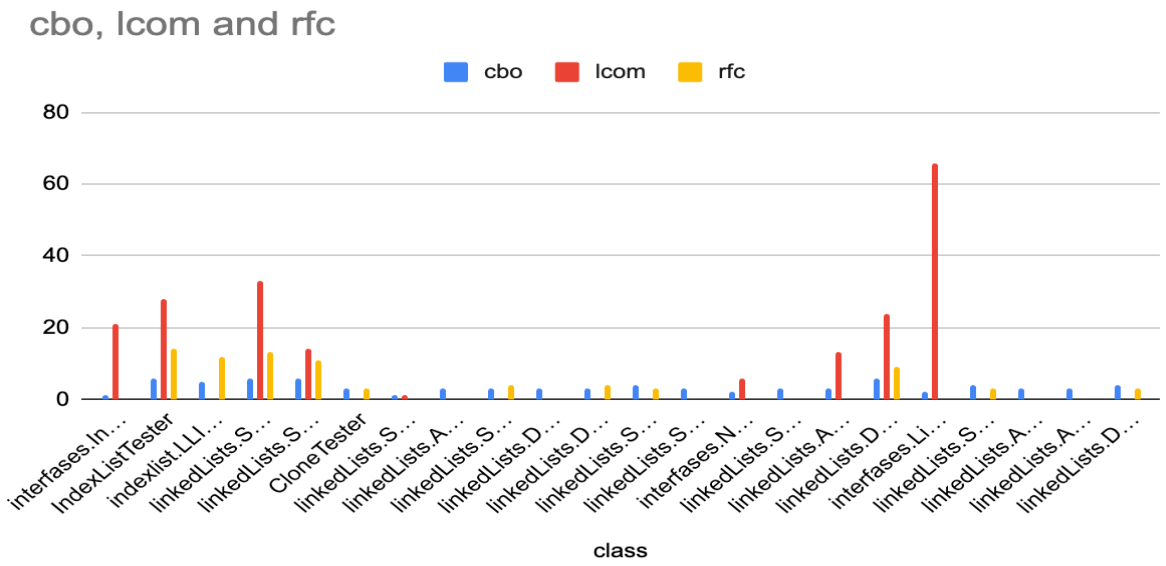


The classes that might have different values from the rest of the classes are:

a) MainGUI$ButtonHandler has a high CBO value (10) and a high RFC value (71). This suggests that the class is tightly coupled to other classes and has a lot of functionality.
b) Scheduling$DateLabelFormatter has a low CBO value (0) and a low RFC value (3). This suggests that the class is not tightly coupled to other classes and does not have much functionality.

Using the jDeodorant tool we found out that this project code contains only a single bed smell which is Long Method in the MainGUI$ButtonHandler class.

5.1.4 Linked List Everywhere

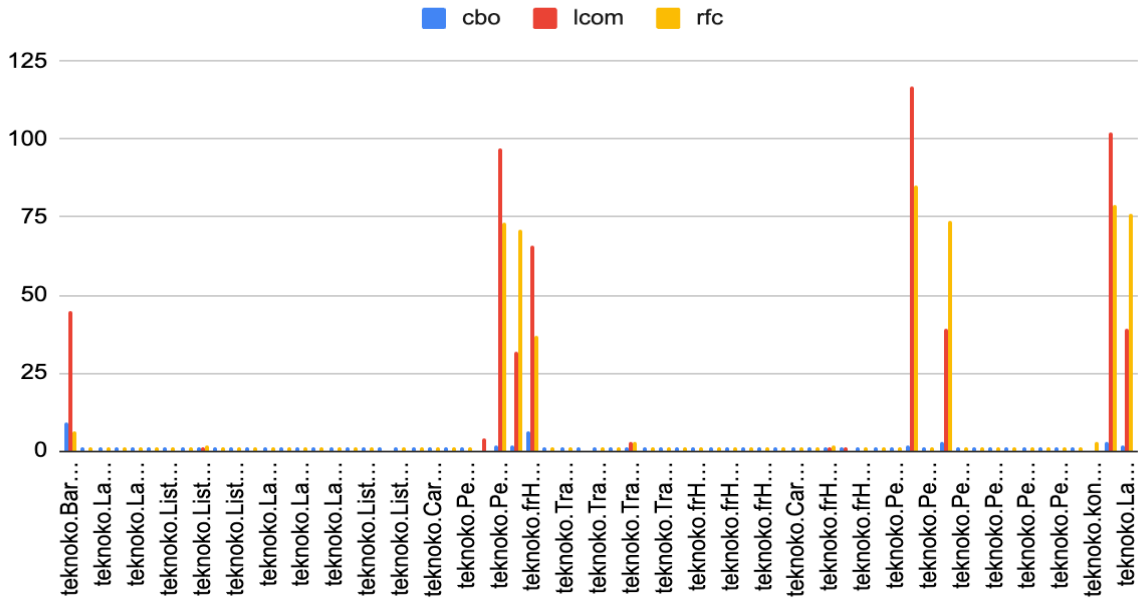


There are a few classes that have different values from the rest of the classes. These classes are:

a) IndexListTester: This class has a high CBO value, indicating that it is tightly coupled to other classes.
b) linkedLists.SLList: This class has a high CBO value, indicating that it is tightly coupled to other classes.
c) CloneTester: This class has a low LCOM value, indicating that the methods in the class are not related to each other.
d) linkedLists.SLFLList$NodesIterator: This class has a low RFC value, indicating that the class is responsible for a lot of functionality.

Using the jDeodorant tool we found that this code does not contain any bed smell.

5.1.5 Techno Master

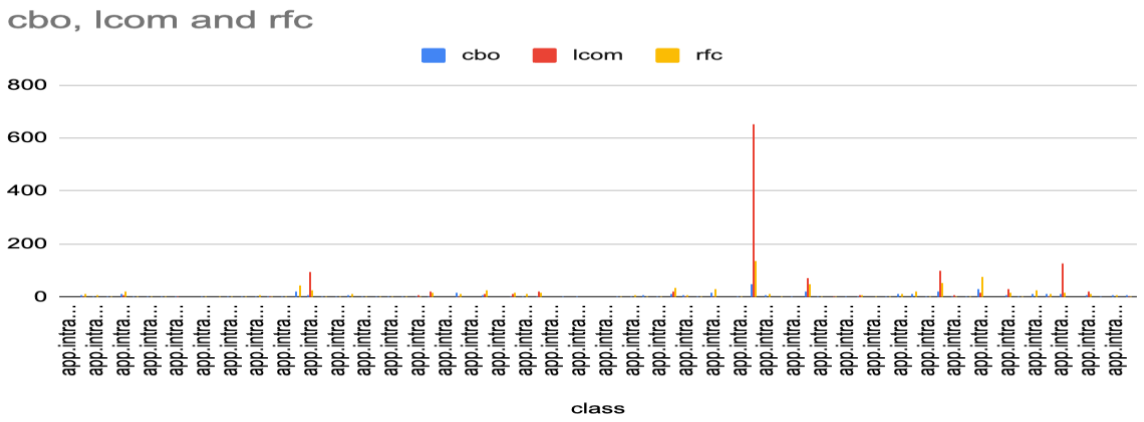


There are a few classes that have different values from the rest of the classes. These classes are:

a) Transaksi: This class has a high CBO value, indicating that it is tightly coupled to other classes.
b) LaporanTransaksi: This class has a high CBO value, indicating that it is tightly coupled to other classes.
c) ListBarang$Anonymous6: This class has a low LCOM value, indicating that the methods in the class are not related to each other.
d) Pengiriman$Anonymous10: This class has a low RFC value, indicating that the class is responsible for a lot of functionality.

Using the jDeodorant tool we found that this project code contains 2 bed smells Long Methods and Duplicate Code at multiple classes.

5.1.6 Intra



Based on these metrics, we have identified a few classes that have different values from the rest of the classes. These classes are:

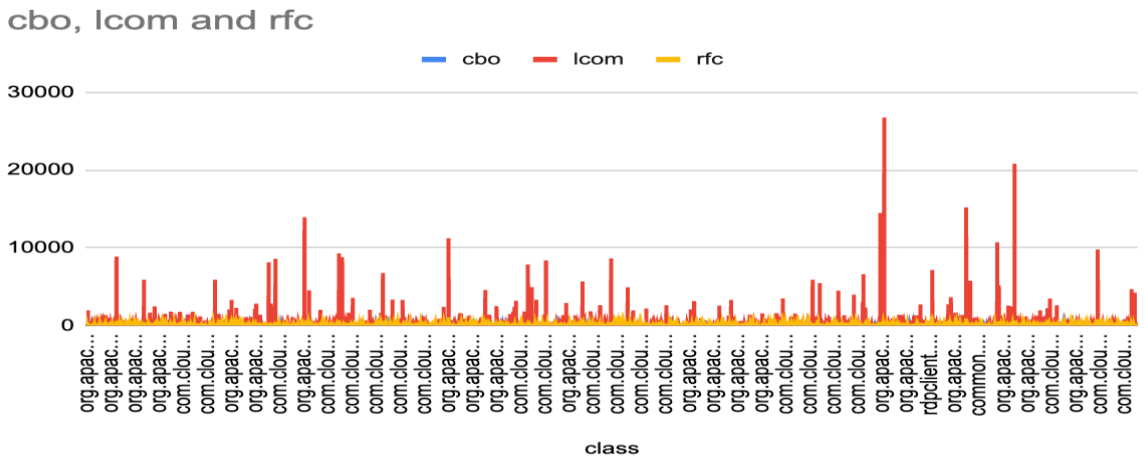
a) app.intra.ui.MainActivity - This class has a high cbo value, which means that it is tightly coupled to other classes. This could make it difficult to change or maintain.
b) app.intra.sys.PersistentState - This class has a high rfc value, which means that it is used by a lot of other classes. This makes it an important part of the system, but it also means that it could be more difficult to change or maintain.
c) app.intra.net.doh.Race - This class has a low cbo and lcom values, which means that it is loosely coupled to other classes. This makes it easier to understand and maintain.

Overall, the results of the CK matrix analysis show that the Intra project is well-designed. The classes are generally well-coupled and cohesive, and there are no major outliers.

Using the jDeodorant tool we found that this code does not contain any bed smell.

5.1.7 Cloudstack

Cloudstack's dataset includes 8242 classes. The CBO, LCOM, and RFC metrics were calculated for each class, revealing varying degrees of coupling, cohesion, and complexity.



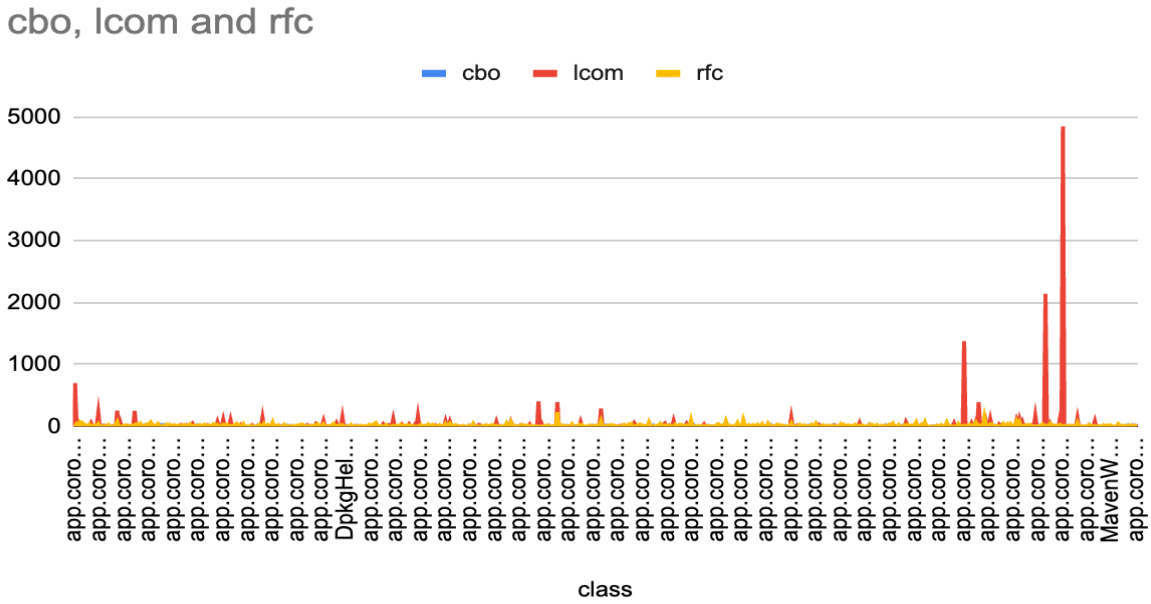
Based on these metrics, we have identified a few classes that have different values from the rest of the classes. These classes are:

a) com.cloud.server.ManagementServerImpl - This class has a high cbo value, which means that it is tightly coupled to other classes.
b) com.cloud.api.ApiResponseHelper - This class has a high rfc value, which means that it is used by a lot of other classes.
c) com.cloud.api.ApiDBUtils - This class has high cbo and lcom values, which means that it is tightly coupled to other classes.

Using the jDeodorant tool we found that this project code contains 2 bed smells Long Methods and Duplicate Code at multiple classes..

5.1.8 Corona-Warn-App Server

Cloudstack's dataset includes classes. The CBO, LCOM, and RFC metrics were calculated for each class, revealing varying degrees of coupling, cohesion, and complexity.



Based on these metrics, we have identified a few classes that have different values from the rest of the classes. These classes are:

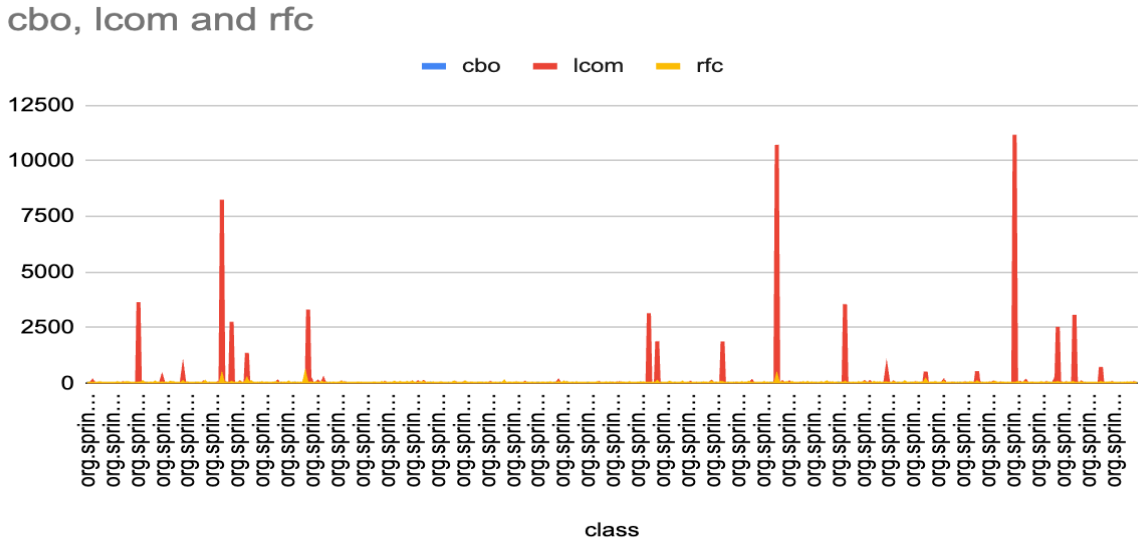
a) app.coronawarn.server.services.distribution.assembly.appconfig.ApplicationConfigurationV2PublicationConfig - This class has a high cbo value 67, which means that it is tightly coupled to other classes.

b) app.coronawarn.server.services.distribution.assembly.appconfig.ApplicationConfigurationV2PublicationConfig - This class has a high rfc value, which means that it is used by a lot of other classes.

c) app.coronawarn.server.services.distribution.statistics.StatisticsJsonStringObject - This class has high lcom values, which means that it is tightly coupled to other classes.

Using the jDeodorant tool we found that this project code contains 1 bed smell which is Feature Envy.

5.1.9 Spring Data JPA

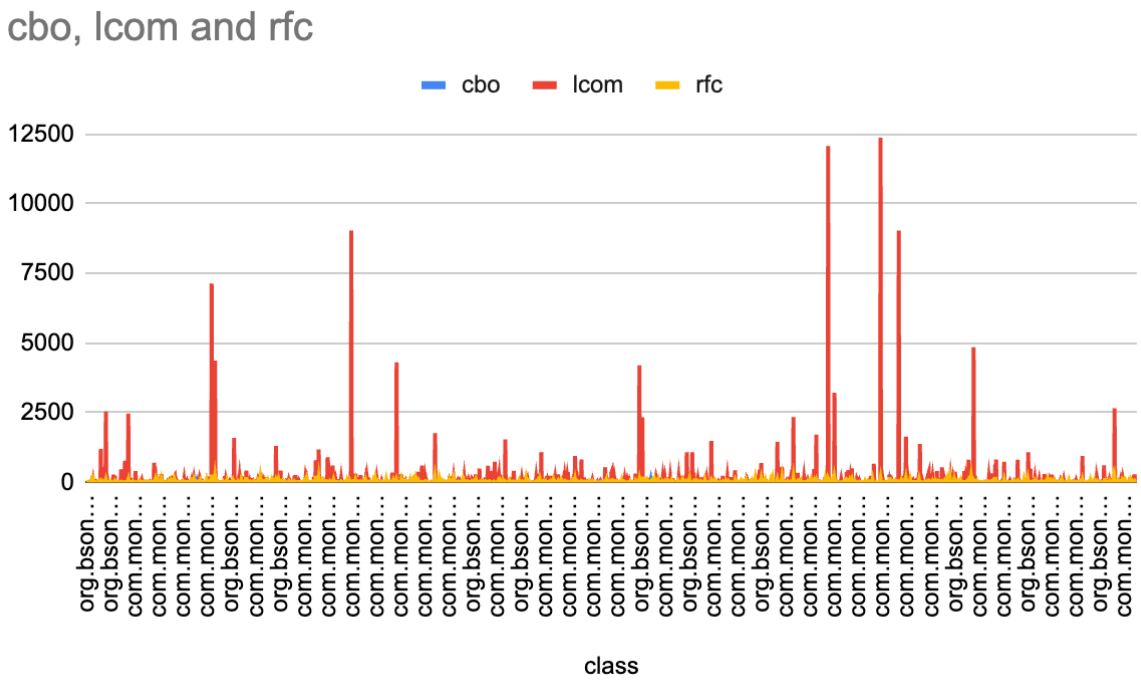


There are a few classes that have different values from the rest of the classes. These classes are:

a) JpqlQueryRenderer: This class has a high CBO value, indicating that it is tightly coupled to other classes.
b) JpqlQueryTransformerTests: This class has a high CBO value, indicating that it is tightly coupled to other classes.
c) Querydsl: This class has a high RFC value, indicating that it is responsible for a lot of functionality.

Using the jDeodorant tool it can be found that this project code contains God Object and Long method smells in MongoClient class.

5.1.10 MongoDB Java Driver



The classes that might have different values from the rest of the classes are:

a) org.bson.BsonDocument has a high CBO value (34) and a low LCOM value (0). This suggests that the class is tightly coupled to other classes and not very cohesive.
b) org.bson.codecs.pojo.entities.conventions.AnnotationNameCollision has a high CBO value (15) and a high LCOM value (22). This suggests that the class is tightly coupled to other classes and not very cohesive.
c) com.mongodb.client.model.IndexOptions has a high RFC value (700). This suggests that the class is responsible, but it may be too responsible and could be refactored into smaller classes.

Using the jDeodorant tool it can be found that this project code contains God Class & Long method smells in MongoClient class.

5.2 Result Summary

The result summary table provides a consolidated view of the metrics obtained from the analysis of the ten projects. It shows the average values of Coupling Between Objects (CBO), Lack of Cohesion in Methods (LCOM), and Response For a Class (RFC) for each project, along with the identified bad smells.

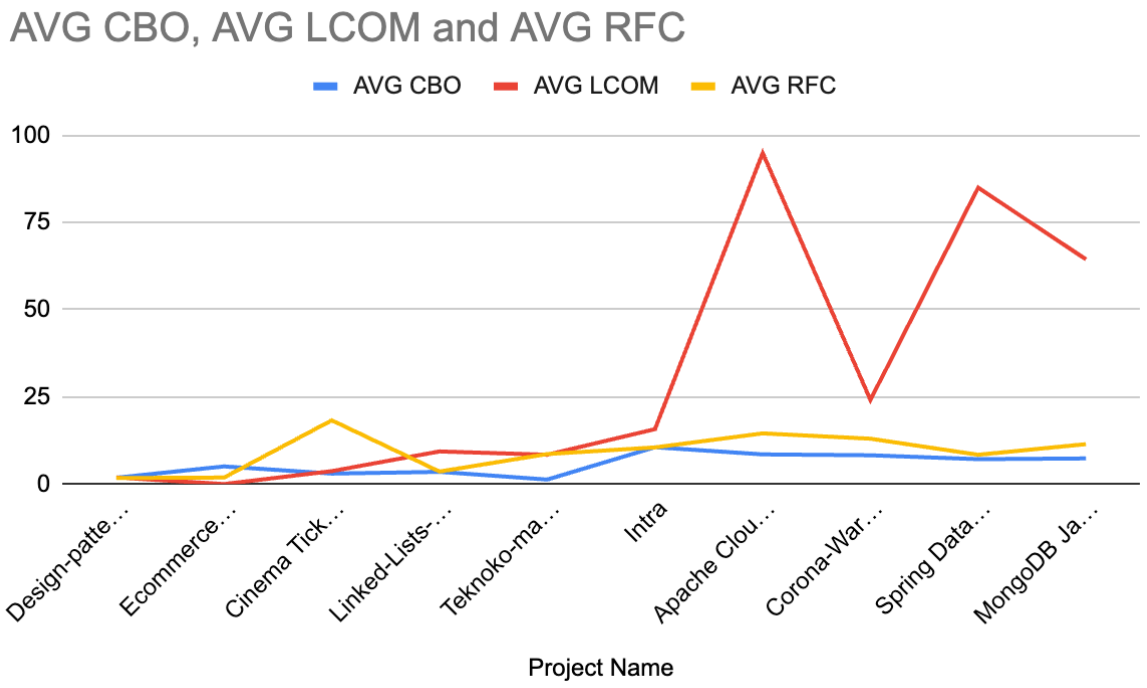
The average CBO, LCOM, and RFC values give an indication of how tightly coupled, cohesive, and complex the classes are within each project. Due to excessive coupling, large response sets, and low cohesion among methods, a loss in modularity is indicated by high values of CBO and RFC and low values of LCOM.

The 'Bed Smell' column identifies the presence of any code smells found in the projects using the jDeodorant tool. It provides insights into the specific code smells present in each project, such as Long Method, Duplicate Code, God Class, Feature Envy, and cases where no code smells were found.

No.	Project Name	AVG CBO	AVG LCOM	AVG RFC	Bed Smell
1	Design-patterns-master	1.82	1.88	1.74	None
2	Ecommerce website-main	5.06	0	1.87	Long Method
3	Cinema Ticket Booking System	3.00	3.71	18.28	Long Method
4	Linked-Lists-Everywhere-master	3.5	9.36	3.59	None
5	Teknoko-master	1.29	8.41	8.61	Long method & Duplicate Code
6	Intra	10.56	15.75	10.51	None
7	Apache CloudStack	8.51	94.96	14.54	Long method & Duplicate Code
8	Corona-Warn-App Server	8.26	24.12	13.01	Feature Envy

9	Spring Data JPA	7.14	85.02	8.40	God Class & Long method
10	MongoDB Java Driver	7.35	64.47	11.45	God Class & Long method

The data shows that projects with code smells typically have higher average CBO, LCOM, and RFC values than other projects, which suggests a lesser level of modularity. Projects lacking recognised code smells, on the other hand, typically have lower average values, indicating better modularity. The association between the occurrence of offensive odours and the decline in software modularity throughout the analysed projects is clearly outlined in this summary table.



6. Conclusion: The Impact of Bad Smells on Software Modularity

Our research aimed to understand the effects of bad smells on software modularity, and the empirical results obtained from the selected ten diverse projects offer insightful conclusions. The projects varied widely in domain, complexity, size, and functionality, but the consistent pattern of lower modularity in the presence of bad smells reinforced the understanding that these code malodours can significantly hamper a project's quality.

Software modularity could be accurately determined using the Chidamber and Kemerer (C&K) metrics, particularly Coupling between Object classes (CBO), Lack of Cohesion in Methods

(LCOM), and Response For a Class (RFC). These metrics allowed us to evaluate the complexity, connection, and coherence of every class in the projects we chose. Our findings indicate that code elements with detected foul smells typically have higher CBO and RFC values and lower LCOM values, indicating stronger coupling, increased complexity, and decreased cohesion—all of which go against the modularity and good software design tenets.

For example, the MongoDB Java Driver project, which contained God Class & Long method smells, showed a high CBO value of 7.35 and an elevated LCOM value of 64.47, indicating high coupling and low cohesion, respectively. On the other hand, the Design-patterns-master project, which had no detected bad smells, demonstrated comparatively lower values for all C&K metrics, suggesting better modularity.

Therefore, our findings conclusively answer our research question, demonstrating that the presence of bad smells in code negatively impacts software modularity. It increases coupling, adds to the complexity (high RFC), and reduces cohesion (high LCOM) in the software classes, which goes against the principles of good software design and modularity.

However, the research done is not exhaustive. To further understand the complex effects of code smells on modularity, future research can take a wider range of projects into account, take into account more metrics and other foul smells, and employ more technologies. However, the current study provides important foundational information for understanding the negative impacts of unpleasant odours on software modularity.

References:

1. Basili, V.R., Caldiera, G., Rombach, H.D., 1994. The goal question metric approach. In Encyclopedia of software engineering. Wiley, New York.
2. Spinellis, D. (2005). Tool writing: A forgotten art? IEEE software, 22(4), 9-11.
3. Tsantalis, N., & Chatzigeorgiou, A. (2009). Identification of extract method refactoring opportunities for the decomposition of methods. Journal of Systems and Software, 84(10), 1757-1782.
4. jDeodorant: Eclipse Plug-in. [Online] Available at: http://jdeodorant.com/
5. Major Java Projects taken for this study are as follows:
    a. Design Patterns Master: https://github.com/iluwatar/java-design-patterns
    b. Apache CloudStack: https://github.com/apache/cloudstack
    c. Corona-Warn-App Server: https://github.com/corona-warn-app/cwa-server
    d. Spring Data JPA: https://github.com/spring-projects/spring-data-jpa
    e. MongoDB Java Driver: https://github.com/mongodb/mongo-java-driver