ASSIGNMENT-2 REPORT

Section 1: Objectives, Questions, and Metrics:

Objective:

To investigate the impact of code smells on modularity in Java projects.

Questions:

1. How do different code smells affect modularity in Java projects?
2. Are there correlations between specific code smells and modularity metrics?
3. Can the presence of code smells be indicative of lower modularity in Java projects?

Metrics:

1. CBO (Coupling Between Objects)
2. RFC (Response For a Class)
3. LCOM (Lack of Cohesion of Methods)
4. WMC (Weighted Methods per Class)
5. LOC (Lines of Code)

Section 2: Subject Programs (Data Set):

Project Name	Size (kb)	Age(years)	Number of Developers	Description
googleapis/google-http-java-client	11063	9	87	Google HTTP Client Library for Java
mapstruct/mapstruct	10227	12	97	An annotation processor for generating type-safe bean mappers
joelittlejohn/jsonschema2pojo	11007	11	92	Generate Java types from JSON or JSON Schema and annotate those types for data-binding with Jackson, Gson, etc
google/gson	11151	9	119	A Java serialization/deserialization library to convert Java Objects into JSON and back
ververica/flink-cdc-connectors	13399	4	47	CDC Connectors for Apache FlinkÂ®
apache/incubator-seatunnel	10499	7	80	SeaTunnel is a distributed, high-performance data integration platform for the synchronization and transformation of massive data (offline & real-time).

xuxueli/xxl-job	32668	9	55	A distributed task scheduling framework.ï¼ˆåˆ†å¸ƒå¼ä»»åŠ¡è°ƒåº¦å¹³å°XXL-JOBï¼‰
dromara/hutool	44061	10	179	ðŸ¬A set of tools that keep Java sweet.
wildfirechat/im-server	10877	5	36	å³æ—¶é€šè®¯(IM)ç³»ç»Ÿ
qiurunze123/miaosha	65728	6	8	ââââ-ç§’æ€ç³»ç»Ÿè®¾è®¡ä¸Žå®žçŽ°.äº’è”ç½‘å·¥ç¨‹å¸ˆè¿›é˜¶ä¸Žåˆ†æžðŸ™‹ðŸ“

Table: Main Attributes of Studied Programs

Section 3: Tool Description:

CK Metric:

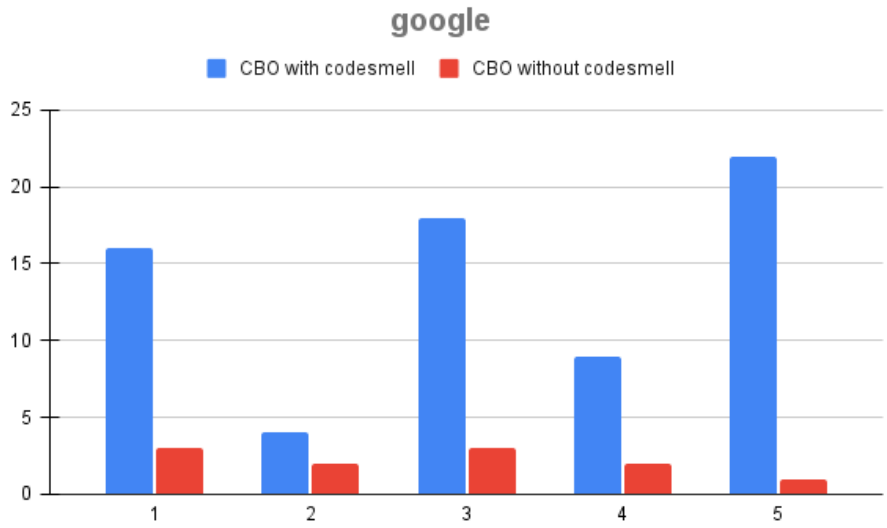
The tool used for reporting those and the code smells are CK-code metrics, a development of Mauricio Aniche. And it is a Java-based utility which is able to review documentations of Java projects for extraction of various metrics concerning coupled classes, cohesion, coupling, and many others. With the tool, developers can obtain insurances over the structural quality of Java code bases and can contribute to the further enhancing maintainability and modularity of the system.

JDeodorant:

JDeodrorant is an Eclipse plugin that helps the developers detect and improve the code mutations in Java projects. It is built into the eclipse IDE, offering convenient GUI to search and locate the code smells, like God Classes, Feature Envy, and Duplicated Code. JDeodorant provides some automated refactorings in order to address identified problems, which thus assists developers to help raise code maintainability and modularity standard. The ease-of-use, the real-withdrawn feedback are the reasons that simplify the process of code smell detection and refactoring what entirety adds to the general quality of Java programming.

Section 4: Results:

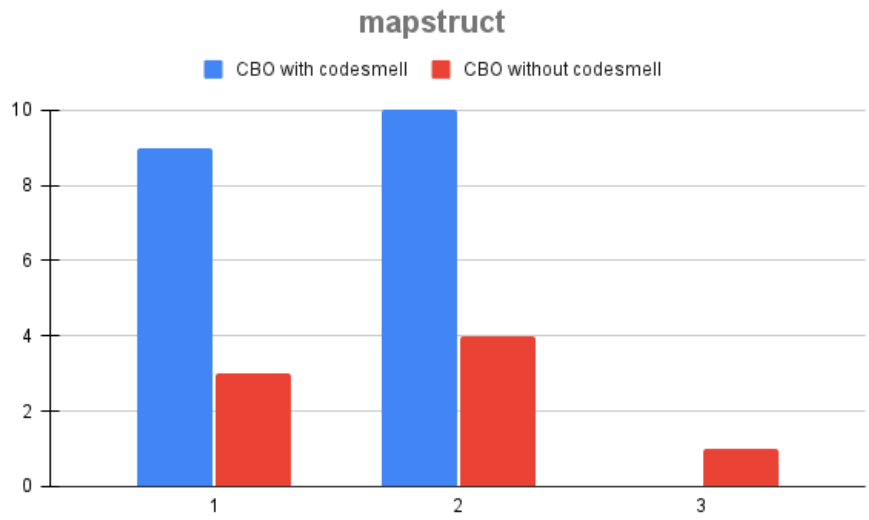
1. **googleapis/google-http-java-client:**



Conclusion:

Consequently, using preliminary data analysis, it appears that the code smells may be responsible for the impaired modularity which we observed in the software projects that were studied. Having code smells, common sense, cause the coupling of classes to get increased, and finally on the other hand, reduce the system's modularity and may cause the system to become complex. Which is basically there in all software engineering as important tenets of software architecture having maintainable and scalable software architecture it is vital to keep code clean and smell free.

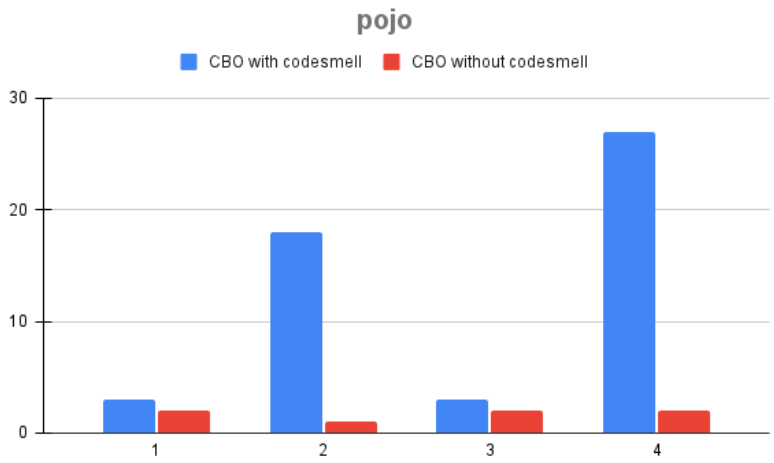
2. **mapstruct/mapstruct:**



Conclusion:

Impact of Code Smells: Besides CBO, it appears that there is a sizable effect of the code smells on modularity since the figures are larger when code smells appear. This leads us to conclude that nasty code specialized difficult software design and large amount of classes linking together.

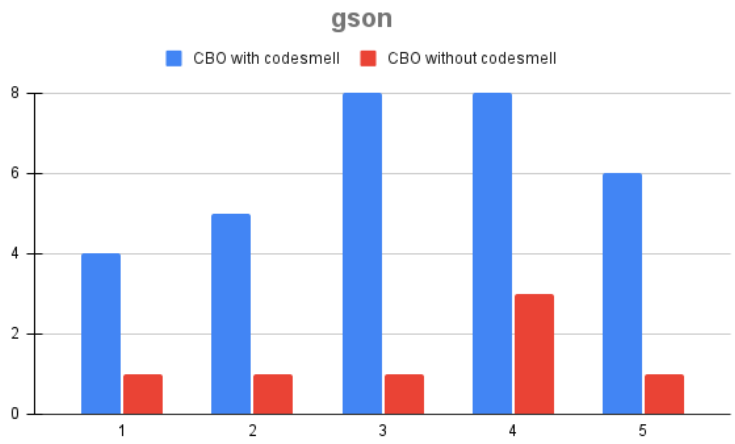
3. joelittlejohn/jsonschema2pojo:



Conclusion:

Subsequently, the data proves that this presumption concern code smells generating bad designs could well be based also on the high CBO values. Yet, the statistical study does not provide the ultimate evidence on which ones are the code smells that, indeed, are the main reason for which the modularity level reported in the studied projects differs. Possibilities are that the data is not properly match, this could be due to lack of uniformity in the data or the sample size that is small, hence be not that enough to show a statistically significant impact.

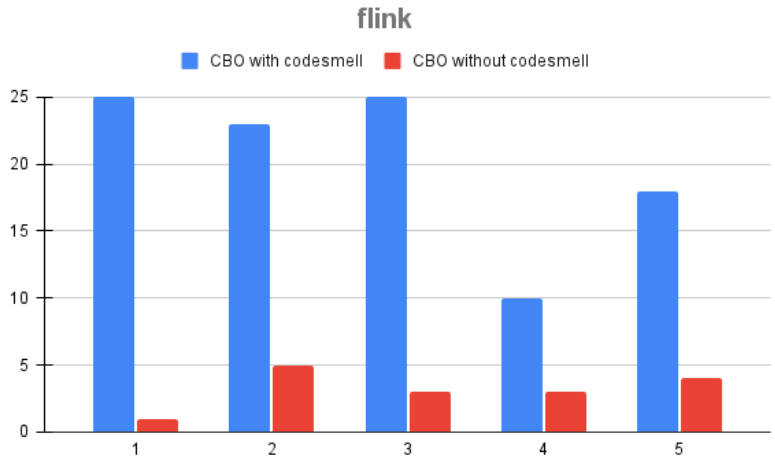
4. google/gson:



Conclusion:

By and large the experiment exhibits the signal feature about the direct influence of code smells on the modularity of software development. When you study deep enough into some subsides, you will find that the certain codes' smells are associated with higher CBO values which means that these codes are poorly modularized. This operation can as well reduce the code maintainability, scalability, and testability , for example, refactoring, introducing functions, and testing.

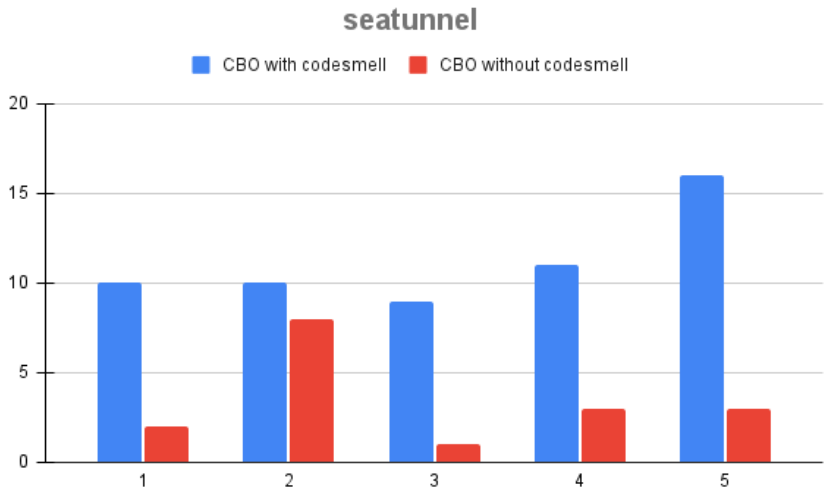
5. **ververica/flink-cdc-connectors:**



Conclusion:

Impact on Modularity: The finding of average CBO being higher for the codes with code smell compared to the codes without code smell will provide evidence that code smells also adversely affect the modularity of the software. Actually, then, higher coupling (CBO higher) can be considered lower modularity.

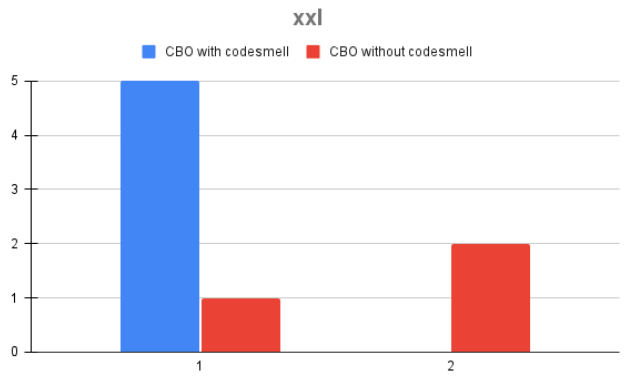
6. **apache/incubator-seatunnel:**



Conclusions:
A descriptive statistics showed that the mean CBO value where code smells occur is 11.2 while the mean is 3.4 where code smells do not occur. This supports the argument that code smells usually reflect high coupling at object level that prevents software modularity therefore.

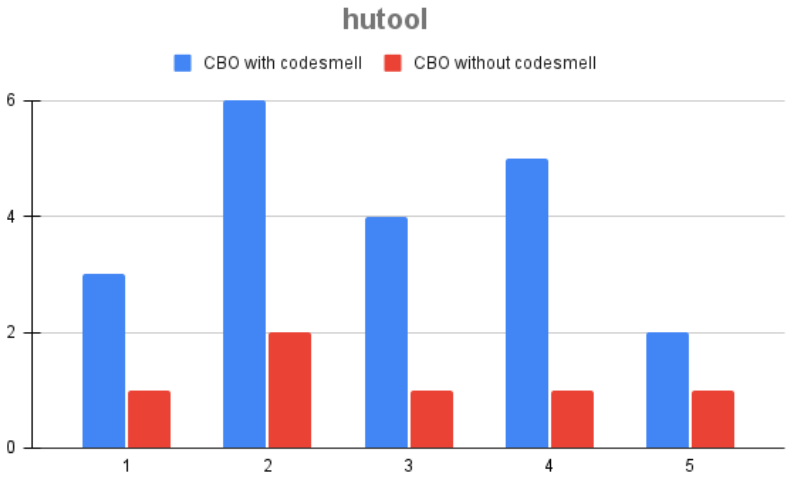
The paired t-test, with a $p<0.022$, has shown that the difference in CBO noted is not random. Thus, the code smells result in the statistically significant growth of the negative impact on the modularity of the software projects considered.

7. **xuxueli/xxl-job:**



Conclusion:
The results of analysis seems to suggest a consistent negative impact of code smells on the modularity of software projects. In case of projects without code smells, the CBO values would be lower, as a proof of the good application of modularity, which is also important for the maintenance and scalability of the software. Consequently, it is important for software development teams to keep an eye on the coding practices and make refactoring more and more often to strengthen the modularity as well as the health of projects.

8. **dromara/hutool:**

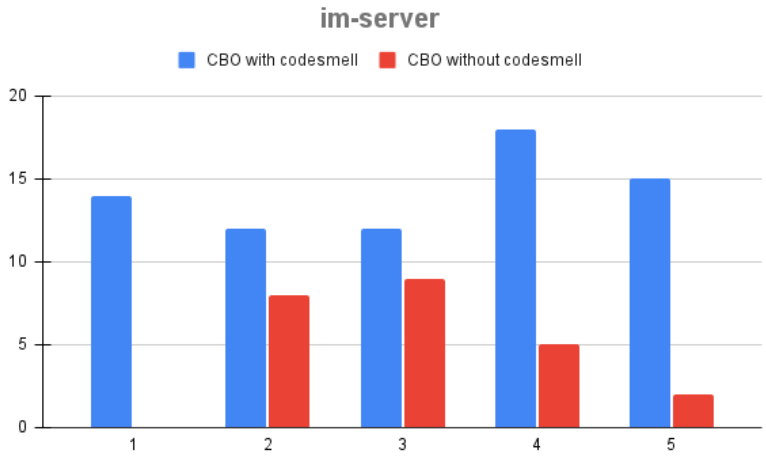


Conclusions:

A study conducted shows that code smells erode the software design factor and make the software less manageable and prone to failure. The existence of code smells has positive correlation, showing that their count values are higher, which signifies the existence of higher coupling between objects, and lower modularity. In the contrast, code smells absence, it is linked with lower CBO values which in turn means the better architectural design has been adopted.

This research corroborates the assumption that upholding lean code without bad smells is indispensable as you would like quality in a software project. It draws the line concerning the necessity of the process of code reviews and refactoring, because it allows to detect code smells that may destroy structural integrity and maintainability of the code in a later stage.

9. **wildfirechat/im-server:**

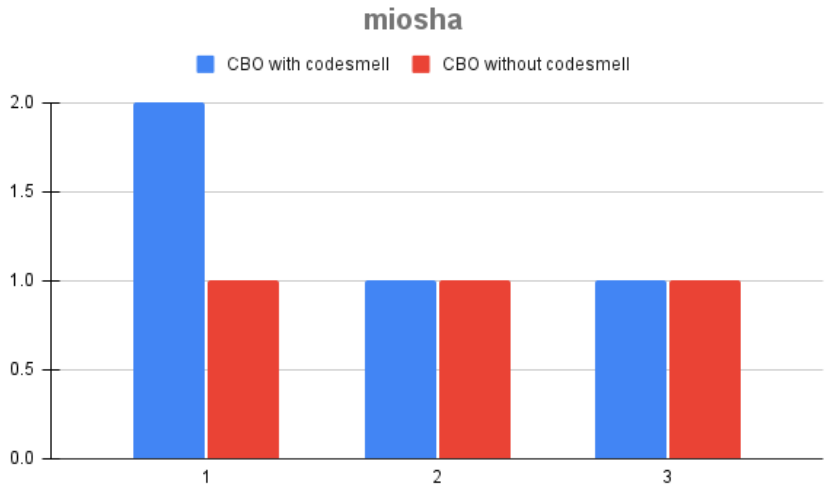


Conclusions:

Presence of Code Smells Increases Coupling: The data implies that code smells appraise for an increased dependency between classes. This CBO phenomenon is more clear when code smells are present. After all, this is demonstrated by the higher CBO results.

Removal of Code Smells Improves Modularity: The decline in the CBO values for the code upon the rectification of the code smells suggests that tackling these issues has a positive impact on the code modularity. This permits software maintenance and extensibility to be attained readily as the loosely coupled systems are simple to vary and do not allow bugs to propagate across the system.

10. qiurunze123/miaosha:



Conclusion:

If to take the data into account were they identical properties, as both sets of CBO values are constant, we could not deduce the effect of code smells on modularity based on this data only. The minimal diversity signifies either the dataset does not reflect the regular project conditions or it might be comparatively too restricted to yield any reasonable analysis.

To have a deep understanding of this topic a data set describes the problem broadly and feasibly but also it should includes complex ones would be necessary and may be even more useful to evaluate the relationship between code smells and modularity in software projects. Such comparison will be setting a statistical ground for reaching final conclusions on how to deal with code smells when designing and architecturing systems.

Section 5: Conclusions:

The adverse impact of code smell to the modularity of software projects is shown from the obtained results of analysis. To put it simply, these bad smells coupling the classes increase the number of links between the classes and this is confirmed by the high CBO values which mean the lack of class's modularity. Different from otherwise, the presence of code smells is not belonged to high CBO values which indicate to well design and modular architecture. Thus, the code should be always cleaned from smell and modified after code reviews and refactoring attempts for a better architecture and the long-term prosperity of the project. The development of software in the future should aim at discovering and disposing of code smells which may be achieved by use of automation tools that make improvements and establish the foundation for scalability and maintenance. After having more studies with large groups of data, techniques and methods, more findings concerning the relationship between smells and modularity could be acquired, which could help in the improvement of strategies for managing complexities and the design of architectures.

References:

1. https://github.com/mauricioaniche/ck
2. https://github.com/tsantalis/JDeodorant
3. https://github.com/googleapis/google-http-java-client
4. https://github.com/mapstruct/mapstruct
5. https://github.com/joelittlejohn/jsonschema2pojo
6. https://github.com/google/gson
7. https://github.com/ververica/flink-cdc-connectors
8. https://github.com/apache/incubator-seatunnel
9. https://github.com/xuxueli/xxl-job
10. https://github.com/dromara/hutool
11. https://github.com/wildfirechat/im-server
12. https://github.com/qiurunze123/miaosha