Software Quality & Metrics (SE-411)
Project Step - 3

Dr. Marouane Kessentini
Software Engineering, DTU

Ankit
2K18/SE/021

Ankit Kr. Yadav
2K18/SE/024

Anmol Yadav
2K18/SE/028

Ashish Kumar
2K18/SE/041

Palak Yadav
2K18/SE/092

Abstract— This document is a project report for SQM Subject. This file contains the components of our project report i.e. Introduction, Problem statement, methodology, Results, Threats to validity, conclusion and references.

INTRODUCTION

This part includes all the concepts that were used in doing this project.

1. Software Quality Metric - The word 'metrics' refers to standards for measurements. Software Quality Metrics means measurement of attributes, pertaining to software quality along with its process of development. They help in identifying the number of defects and code smells present currently in the software. However, it is not limited to only this and further takes into consideration other software qualities like maintainability, reliability etc.

2. Code Smells- Also commonly known as antipatterns/design flaws/defects/bad smell now the first thing is that it is not a bug in the code and hence, do not affect the functionality instead they indicate the weakness or drawback in the project which can result to bad design practices and further result into bugs in future and hence a good project must be free of code smells. This is also why these become a point of concentration for project developers because requirements can change frequently and if the project is free from these types of smells from the very beginning then it becomes a comparatively easy task in future. Thus our project focuses majorly on this to find code smells in projects which are done using 2 different automated tools to improve the quality of the project by removing the code smells identified by the tool.

3. Software Refactoring- It is the process of restructuring the code without changing its external behaviour i.e. only internal design is changed without changing any of its functionalities this is used to remove code smells for the project.

Code Assessment Tools used:

Designite Java - DesigniteJava is a code quality assessment tool for any code that is written in Java. It detects a number of design, architecture and implementation smells that show issues related to maintainability, which is present in the analyzed code. It also computes many commonly used object-oriented metrics. It helps in reducing technical debt and improving the maintainability of software.

- DesigniteJava detects 17 design smells, including Imperative Abstraction, Unutilized abstraction, Deficient Encapsulation, Broken Modularization, Cyclic-Dependent Modularization, Cyclic Hierarchy, Wide Hierarchy, etc.
- It also detects 10 implementation smells, for example, Abstract Function Call From Constructor, Complex Conditional, Empty catch clause, Long Parameter List, Long Statement, Magic Number, Missing default, etc.
- DesigniteJava computes several object-oriented metrics, including LOC (Lines Of Code), CC (Cyclomatic Complexity), PC (Parameter Count), NOM (Number of Methods), WMC (Weighted Methods per Class), NC (Number of Children), DIT (Depth of Inheritance Tree), FANIN (Fan-in), FANOUT (Fan-out), etc.
- It supports trend analysis that reveals the evolution from the perspective of smells.
- It computes object oriented design metrics which are helpful to gauge structural health of software.

SonarQube - SonarQube is an open-source platform that is developed by SonarSource for regular inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities on 27 programming languages. SonarQube is also one of the code quality assurance tool which is used to not only

collect but also analyze source code. SonarQube offers reports on duplicated code, coding standards, unit tests, code coverage, code complexity, comments, bugs, and security vulnerabilities. SonarQube can record metrics history and provides evolution graphs. This tool also reduces risk which occurs in software development in a very short period of time. It does this by detecting bugs in our code and also alerts developers to fix the bugs before the software is rolled out for production. SonarQube provides fully automated analysis and integration with Maven, Ant, Gradle, MSBuild, and continuous integration tools. It pairs up with the existing software pipeline and provides clear remediation guidance for developers to understand and fix issues.
SonarQube supports programming languages like Java (including Android), C#, C, C++, JavaScript, TypeScript, Python, Go, Swift, COBOL, Apex, PHP, Kotlin, Ruby, Scala, HTML, CSS, ABAP, Flex, Objective-C, PL/I, PL/SQL, RPG, T-SQL, VB.NET, VB6, and XML.

Problem Statement

Detecting code smells manually was not an easy task. It requires a lot of time to read the code of the entire project and let's suppose a project has 10K Line of Code, so it becomes a very hectic task for the tester or developer to detect code smells manually. Nowadays, testers use automated testing tools to detect code smells very easily and so we also use automated testing tools in this project.

Our main motive in this project is to test various open source projects written in Java language which are available on github having lines of code (LOC) greater than 5 thousand using any two testing tools and then comparing both of the tools on the basis of their results. To compare the results we will create a table that contains number of code smells detected, number of bugs etc. We will also suggest various methods for refactoring the flaws of the code.

Finally, in conclusion part, we will be suggesting which tools is more useful and which one of them would be more beneficial to assess software quality.

Methodology

Our methodology will be to first choose 10 Java projects from github and then identify the use cases for this study. Then we use 2 automated testing tools to test our projects and then tools will give us a detailed report on various flaws in code. After gathering all the results generated by the tools we will suggest appropriate refactoring methods that can be used to eliminate that code smell.

The main motive of this project is to compare two automated testing tools on the basis of their results produced and features that they provide. It is aimed to detect code smells from any project and then by using refactoring methods to abolish the identified code smells.

Steps that we followed to complete this project are as follows:

Step-1:

1. We first meet our group members on zoom, and then we discuss on document software requirements & develop initial Use-Case Diagram.

2. We learned what quality assessment/code-smells detection tool is and what code smells is.

3. We select 10 github Repositories that are written in Java programming language with minimum 5000 lines of code and have multiple contributors.

4. We select 2 quality automated assessment tools. We chose DesigniteJava & SonarQube.

Step-2:

1. We give a brief overview about the project with the help of Requirements' modeling.

2. We have made 6 Sequence diagrams (of basic flows of each use case), 1 Class diagram and 1 Activity diagram (of the whole scenario).

Step-3:

1. Run tools on selected projects.

2. Gather and analyze the results.

3. Compare both the tools on the basis of results.

4. Generation of final report.

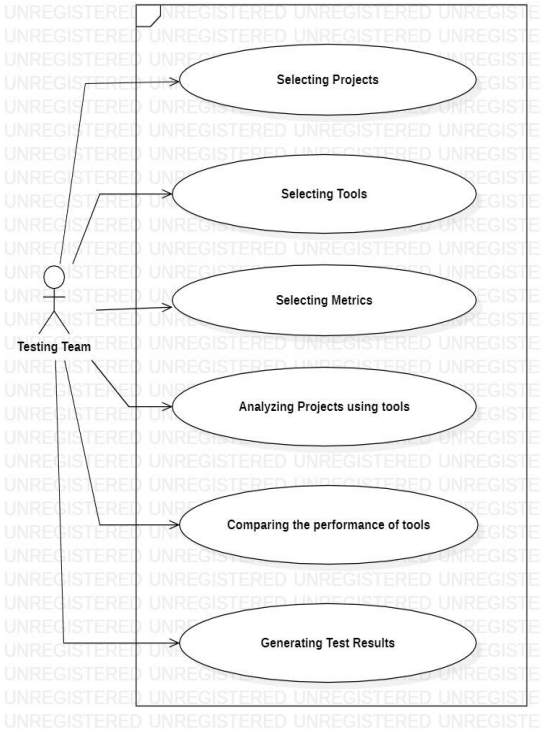
From Github, 10 repositories were selected for the project, each of them is written in java programming language. According to the criteria, each of them has more than 5000 Lines of Code (LOC) and has multiple contributors. The github link of each repository is given below:

1. https://github.com/SkyTubeTeam/SkyTube
2. https://github.com/ZieIony/Carbon
3. https://github.com/iluwatar/java-design-patterns
4. https://github.com/PhilJay/MPAndroidChart
5. https://github.com/square/retrofit
6. https://github.com/naman14/Timber
7. https://github.com/timusus/Shuttle
8. https://github.com/amirzaidi/Launcher3
9. https://github.com/Rohitjoshi9023/KBC--Kaun-Banega-Crorepati
10. https://github.com/IrisShaders/Iris

Following use cases have been identified:

- Selecting projects
- Selecting tools
- Selecting metrics
- Analyzing projects using tools to gather metrics
- Comparing the performance of tools
- Generating test result report

We have also made use-case diagram in starUML software on the basis of identified use cases.



Note: We have made sequence diagrams and an activity diagram in Project Step-2, but not included in this final report.

This project would help us to understand different types of code smells that can occur during project development and would also provide refactoring techniques that would be helpful to remove code smells. We can compare between different projects on the basis of how well they performed in terms of software quality and then can give ratings to different projects.

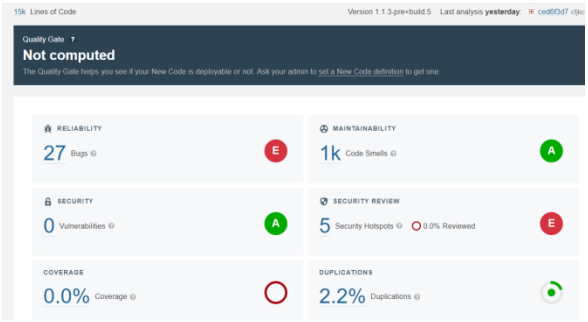
RESULTS

We tested our github repositories on the two tools that we selected and it produced results and we summarize them into below table:

DesigniteJava

Repo's Name	LOC	Architecture smells	Design smells	Implementation smells	Total code smells
SkyTube	17K	66	141	665	872
Carbon	38K	49	254	2.1K	2.4K
java-design-patterns	41K	64	450	2K	2.5K
MPAndroidChart	26K	56	219	2742	3K
retrofit	25K	16	462	767	1.2K
Timber	20K	55	178	1154	1.3K
Shuttle	91K	190	659	4.7K	5.5K
Launcher3	62K	124	245	1.9K	2.2K
KBC--Kaun-Banega-Crorepati	4.7K	45	2	1K	1.04K
Iris	97K	266	389	1.5K	2.1K

This is the screenshot of SonarQube after successful executing on Iris repository:



SonarQube

Repo's Name	LOC	Code Smells Detected	Bugs	Security Reviews (secuity hotspots)	Duplicacy
SkyTube	30K	2.5K	63	12	6.2%
Carbon	65K	1.7K	85	7	3.8%
java-design-patterns	33K	659	56	34	0.7%
MPAndroidChart	25K	1.2K	22	59	10.2 %
retrofit	30K	352	20	6	1.6%
Timber	27K	1.6K	76	8	3.6%
Shuttle	65K	3.6K	17	42	23.4 %
Launcher3	72K	965	58	15	4.2 %
KBC--Kaun-Banega-Crorepati	6K	2.3K	79	48	8.9%
Iris	14K	1K	27	5	2.2%

Results Comparison:

1st repository "SkyTube" has 31K LOC. In this case, SonarQube is successful in calculating exact number of LOC. In terms of code smells, there is a difference of 1.5K code smells between both of the tools.

2nd repository "Carbon" has 58K LOC. In this case, SonarQube is successful in calculating approximate same number of LOC. In terms of code smells, there is a difference of 700 code smells between both of the tools.

3rd repository "java design patterns" has 71K LOC. In this case, both SonarQube & DesigniteJava is not successful in calculating exact number of LOC. In terms of code smells, there is a difference of 1.8K code smells between both of the tools.

4th repository "MPAndroidChart" has 42K LOC. In this case, both SonarQube & DesigniteJava is not successful in calculating exact number of LOC. In terms of code smells, there is a difference of 1.9K code smells between both of the tools.

5th repository "retrofit" has 30K LOC. In this case, SonarQube is successful in calculating exact number of LOC. In terms of code smells, there is a difference of 700 code smells between both of the tools.

6th repository "Timber" has 27K LOC. In this case, SonarQube is successful in calculating exact number of LOC. In terms of detecting total code smells, both tools are approximately same.

7th repository "Shuttle" has 71K LOC. In this, SonarQube is closer to exact LOC. In terms of code smells, there is a difference of 2K code smells between both of the tools.

8th repository "Launcher3" has 80K LOC. In this, SonarQube is closer to exact LOC. In terms of code smells, there is a difference of 1K code smells between both of the tools.

9th repository "KBC" has 6K LOC. In this case, SonarQube is successful in calculating exact number of LOC. In terms of code smells, there is a difference of 1K code smells between both of the tools.

10th repository "Iris" has 97K LOC. In this, DesigniteJava is closer to exact LOC, but SonarQube is very far from exact LOC. In terms of code smells, there is a difference of 1K code smells between both of the tools.

We are not able to remove each & every code smells manually that was detected by the tools, so we are specifying following Refactoring techniques that can be used to remove Code smells:-

Extract Method - The Extract Method refactoring is one of the most useful for reducing the amount of duplication in code. It also means that logic embedded in that method cannot be reused elsewhere.

Extract class - We can refractor a class into two separate classes, each with the appropriate responsibility.

Extract subclass - When a class has features (attributes and methods) that would only be useful in specialized instances, we can create a specialization of that class and give it those features. This makes the original class less specialized (i.e., more abstract), and good design is about binding to abstractions wherever possible.

Extract Superclass - When you find two or more classes that share common features, consider abstracting those shared features into a super-class. This makes it easier to removes duplicate code from the original classes.

Threats to Validity

1. One of the first problems that we faced while choosing online tools for this project is that some of them are paid while some of them are not quite good. So, choosing the right and free tool was not an easy task.
2. While selecting the github projects in a specific language, it is very hard to count LOC manually. So to overcome this, we used an online LOC count tool.
3. Both of the testing tools have different UI. DesigniteJava runs by executing some commands while SonarQube executes on sonarCloud and both of them use different criteria to evaluate code smells. So it becomes difficult to adapt with both the tool's execution process and both of the tools can generate different kinds of code smells, so during summarization of results, it was very hard to know which code smells to be written in the table.

4. In SonarQube, the execution process is quite complex and the configuration of the project took a lot of time because of the build time which was around more than 30min. So we had to wait for half an hour to get the results.

Conclusion

We conclude that we were able to carry out all the necessary operations for completing this project and hence we have successfully completed it.

We tested our 10 chosen github repositories on both of the two quality assessment tools and find out how much they vary in terms of results. We found architecture smells, design smells from Designite java and using SonarQube we found out code smells, bugs, duplicacy and security reviews.

Although SonarQube was more difficult to set up than DesigniteJava as it takes a lot of time (around 30 min.) to build the project while DesigniteJava is able to give results in just 1-2 min by using simple commands on the terminal. SonarQube gave more in depth results and a variety of options including documentations for each bug and code smells and also provides other useful things like duplicacy, security review etc.

At the end, after doing our research we would suggest using SonarQube for in depth overview of a project and for a simple overview of a project then a person can go for DesigniteJava.

For future work, we will expand this study to 4-5 quality assessment tools and then compare their results.

References

1. https://github.com/tushartushar/DesigniteJava
2. https://docs.sonarqube.org/latest
3. https://www.designite-tools.com/faq/
4. Sharma, Tushar & Singh, Paramvir & Spinellis, Diomidis. (2020). An empirical investigation on the relationship between design and architecture smells.Empirical Software Engineering. 25. 10.1007/s10664-020-09847-2.