DELHI TECHNOLOGICAL UNIVERSITY

SOFTWARE QUALITY & METRICS (SE-411)

STEP 1

Submitted To:

Dr. Marouane Kessentini

Submitted By:

Ankit(2K18/SE/021)

Ankit Kumar Yadav(2K18/SE/024)

Anmol Yadav(2K18/SE/028)

Ashish Kumar(2K18/SE/041)

Palak Yadav(2K18/SE/092)

Team Introduction:

In order to do the Software Quality & Metric Project, a group of 5 members is formed. The group members are:

Ankit	2K18/SE/021
Ankit Kumar Yadav	2K18/SE/024
Anmol Yadav	2K18/SE/028
Ashish Kumar	2K18/SE/041
Palak Yadav	2K18/SE/092

Github Repositories:

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

Tools Used:

1. DesigniteJava

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.

2. 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 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. SonarQube provides fully automated analysis and integration with Maven, Ant, Gradle, MSBuild, and continuous integration tools (Atlassian Bamboo, Jenkins, Hudson, etc.). It pairs up with the existing software pipeline and provides clear remediation guidance for developers to understand and fix issues.

The 27 languages supported by SonarQube include 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.

Use cases:

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 included an actor as well and it is named “Testing Team”.

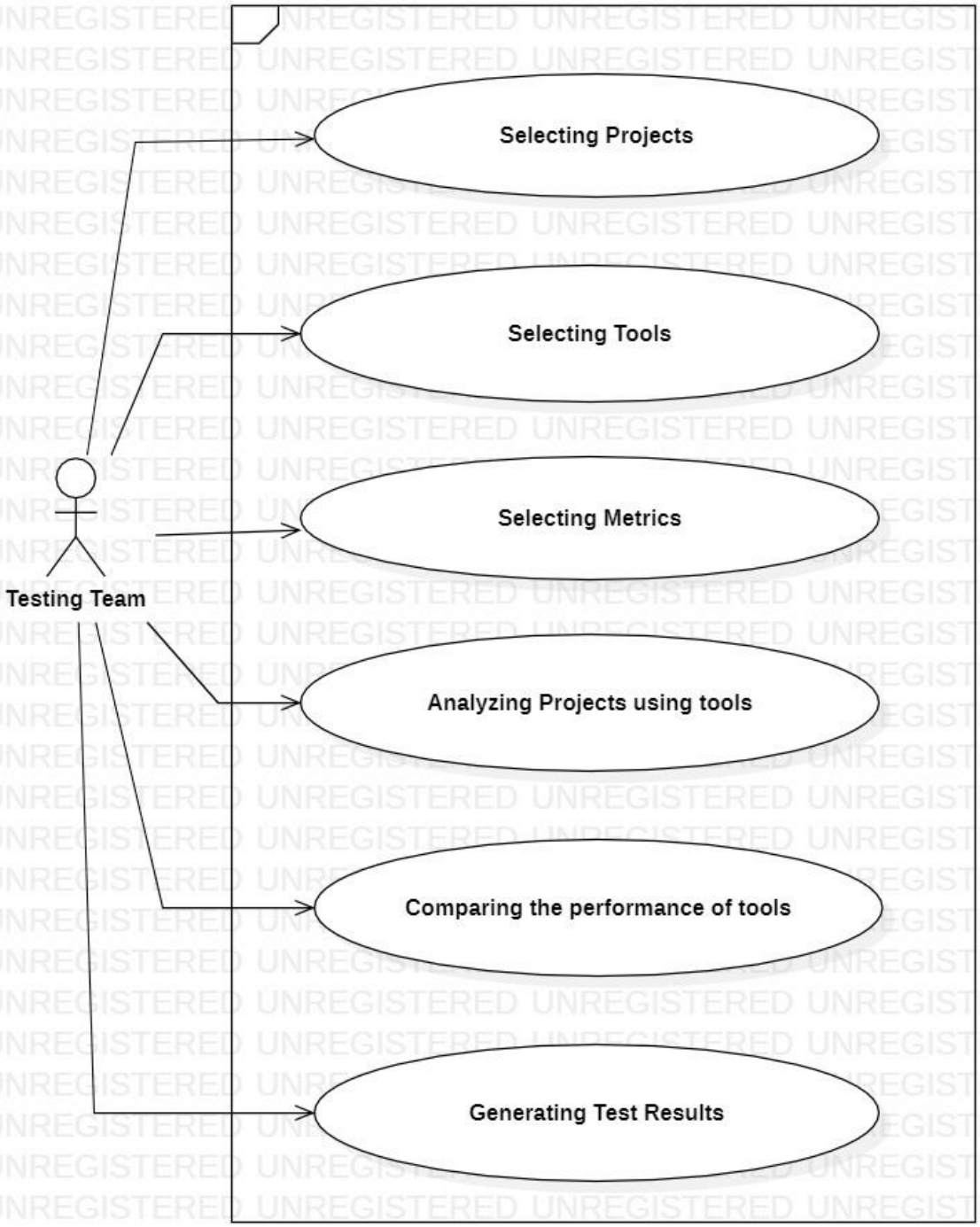

Fig. Use Case Diagram

Use Case Descriptions:

1. Selecting Projects

Introduction: This use case documents the steps that must be followed to select various repositories.
Actors: Testing Team.
Pre-Condition: The project must have at least 5000 LOC.
Post-Condition: If the use case is successful, then the project is selected for code quality assessment.
Event Flow: **Basic Flow:** 1) Tester login into GitHub. 2) Search project using Search bar. 3) Select the project and count its lines of code using an online tool. 4) If LOC is greater than 5000, select the repository, otherwise search for another project. **Alternate Flows:** **Alternate Flow 1: Repositories less than 5000 LOC.** If the Repository is less than 5000 LOC, then we do not select that repository, and the actor returns to the basic flow, and the basic flow is restarted.
Special Requirement: None
Associated Use Case: None

2. Selecting Tools

Introduction:

This use case documents the steps that must be followed to select two code quality assessment tools.

Actors:

Testing Team.

Pre-Condition:

The tool is easily accessible and free.

Post-Condition:

If the use case is successful, then the tool is selected.

Event Flow:

Basic Flow:

1) Tester searches the web for the tool.
2) Check if the tool is compatible with the repository, depending on the programming language.
3) The tool is compatible.
4) We select that tool for analysing our selected projects.

Alternate Flows:

Alternate Flow 1: The tool is not compatible.

If the Tool is not compatible with Repositories, then do not select that Tool, and the actor returns to the basic flow of the execution.

Special Requirement:

None

Associated Use Case:

None

3. Selecting Metrics

Introduction: This use case documents the steps that must be followed to select various metrics.
Actors: Testing Team.
Pre-Condition: Metrics should be available in both tools.
Post-Condition: Metrics have been finalized.
Event Flow: **Basic Flow:** 1) Search for metrics in DesigniteJava. 2) Search for metrics in SonarQube. 3) Selecting the metrics that will be used in generating the results. **Alternate Flows:** None
Special Requirement: None
Associated Use Case: None

4. Analyzing Projects

Introduction: This use case documents the steps that must be followed to analyze projects.
Actors: Testing Team.
Pre-Condition: Projects are compatible with the tools.
Post-Condition: Projects have been successfully analyzed.
Event Flow: **Basic Flow:** 1. Running the tool on the respective repository i.e projects. 2. Going through the metrics generated by the tool. 3. Verifying the results. **Alternate Flow:** None
Special Requirement: None
Associated Use Case: None

5. Comparing Tools

Introduction:

This use case documents the steps that must be followed to compare the tools.

Actors:

Testing Team.

Pre-Condition:

Repositories are successfully analyzed and metrics are ready to be compared.

Post-Condition:

Metrics have been successfully compared and a report is generated.

Event Flow:

Basic Flow:

1. Go through the generated metrics.
2. For every metric, verify whether the result is correct as per the other tool or from code perspective.
3. If the value of the respective metric is correct, then mark it as “True”.
4. Compare the number of True of both the tools.

Alternate Flow:

Alternate Flow 1: incorrect result

1. If the value of the respective metric generated by the tool is incorrect, then mark it as “False”.
2. Compare the number of False of both the tools.

Special Requirement:

None

Associated Use Case:

None

6. Generating Result

Introduction:

This use case documents the steps that must be followed to generate the result.

Actors:

Testing Team.

Pre-Condition:

All the repositories have been analyzed successfully.

Post-Condition:

Result is generated.

Event Flow:

Basic Flow:

1. Going through the previous generated reports.
2. Note down the values of these following mentioned factors:
    - Number of metrics generated.
    - Number of types of metrics generated like code smells, architecture smells, cohesion etc.
    - Number of metrics that were true or false.
3. Compute the final results of both tools.

Alternate Flow: None

Special Requirement:

None

Associated Use Case:

None