8 Lab: Clean code and static analysis

v2025-04-02

8.0 Introduction

Learning objectives

- Use static code analysis to inspect code quality.
- Apply quality gates and quality dashboards to track code quality (using Sonar Qube)

Key points

- Static code quality can assess a code base to produce quality metrics. These metrics are based on the occurrence of known "bad smells" and weaknesses. In this kind of analysis, the solution is not deployed, nor is the code executed (thus the name static analysis).
- Static code analysis can be run locally using a "linter" (usually integrated in the IDE) but it would be even more import to implement code analysis in the team development infrastructure, i.e., at the continuous integration pipeline, using specialized services.
- Key code quality measures include occurrences of problematic code ("bugs" in logic/flow), vulnerabilities (security/reliability concerns), code smells (bad/poor practice or coding style); coverage (ratio tested/total); and code complexity.
- The estimated effort to correct the vulnerabilities is called the technical debt. Every software quality engineer needs tools to obtain realistic information on the technical debt.

Explore

- public projects on Sonar cloud that you can browse and learn.
- Webinar on static code analysis (promoted by Codacy)

8.1 Local analysis

Static code analysis is typically integrated into the build cycle using a cloud service or a dedicated team server. In this lab we will use (containerized) SonarQube locally (SonarQube Cloud is the SaaS version).

a) Prepare/copy a Maven-based Java project to analyze with SonarQube. The project should contain tests and the Jacoco plugin configured (in POM). You may reuse one from previous labs, for example, the *Euromillions* from Lab 1.2

b) Prepare a local instance of SonarQube server (using the Docker image).

For this lab, you do not need to configure a production database (an embedded database is used by default; but for a production scenario you should use a more demanding configuration).

Confirm that you can access the Sonar dashboard (default : http://127.0.0.1:9000) and change the default credentials (admin / admin).

Note1: you may get a **conflict on port 9000** in your host, as it is also commonly picked for other services (e.g.: Portainer); use another port mapping, if needed.

c) Complete the initial configuration to create a local project.
   - Select "Create a local project" link
   - Give a name to the project and, next, choose a baseline method to detect changes (default "Use the global setting").
   - hit "Create project".

d) Now, let us configure the analysis settings.
   - Your should be in the "How do you want to analyze your repository?" page. If not, select Settings for the newly create project.
   - Complete the Analysis method configuration. In step 1, set "Analysis method" to "Locally".
   - Next, generate an access token to be provided in client tools. Be sure to write in down (you will need it later).
   - In step 2, chose Maven as the build tool. Copy the sample maven command generated for you. You will use this command to run the analysis from command line.

e) In your project, lock the Sonar Maven plugin version in your POM, using an updated version.

Run the code analysis from Maven build (highlighted parts should be changed as needed):

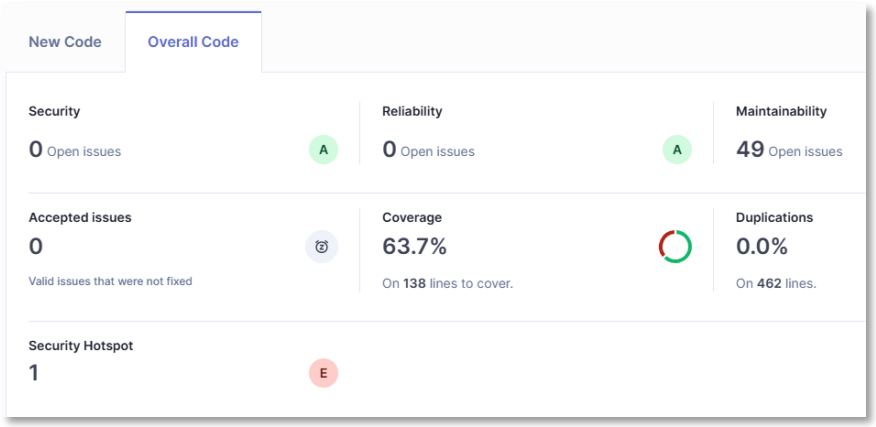
```
$ mvn clean verify sonar:sonar -Dsonar.projectKey=eurom1 -Dsonar.projectName='eurom1'
-Dsonar.host.url=http://127.0.0.1:9000 -
Dsonar.token=sqp_487a32fdbab1f8ac2b37251294ea9d30000003
```

Note: alternatively, you can save part of the Sonar configuration as "global settings", shortening the required maven command.

f) Confirm that Sonar analysis was executed. Access the SonarQube dashboard (default : http://127.0.0.1:9090/projects).

   What was found in the analysis (copy the dashboard)?

   Has your project passed the defined quality gate? Elaborate your answer (in the **Readme** document).



g) Explore the analysis results and complete with a few sample issues, as applicable.

Issue	**Problem description**	**How to solve**
Security	…	…
Reliability		
Maintainability		
Security hotspot		

h) In the configuration of Sonar, you can see references to external tools such as Checkstyle, PMD and SpotBugs related to Java analysis. What are they?

8.2 Technical debt (Cars)

Let's analyze a project with web endpoints. Make a copy of the "Cars" project from Lab 3.2

Be sure you are using a project with JUnit **tests implemented and passing**.

a) Default Spring Boot configuration does not include the JaCoCo plugin by default and the coverage information will not be picked up by Sonar.

   Be sure that you have tests in the project and the JaCoCo plugin configure in the Maven build (use an updated plugin version).

b) Analyze this project with SonarQube.

   Remember to create a new project in your Sonar instance.

c) Take note of the **technical debt found** (Measures > Overview). Explain the reported values.

d) Discuss the coverage values on the SonarQube dashboard (how many lines are "not covered"? And how many conditions? Are the values good?...)

e) Be sure to introduce changes in your code.

   - Analyze the reported problems and be sure to **correct the severe** code smells reported (critical and major).
   - Add some additional API endpoints.

Note: if you used the Entity data type as parameter in the API methods, you will likely get the vulnerability "Persistent entities should not be used as arguments".

Update (rerun) the analysis.

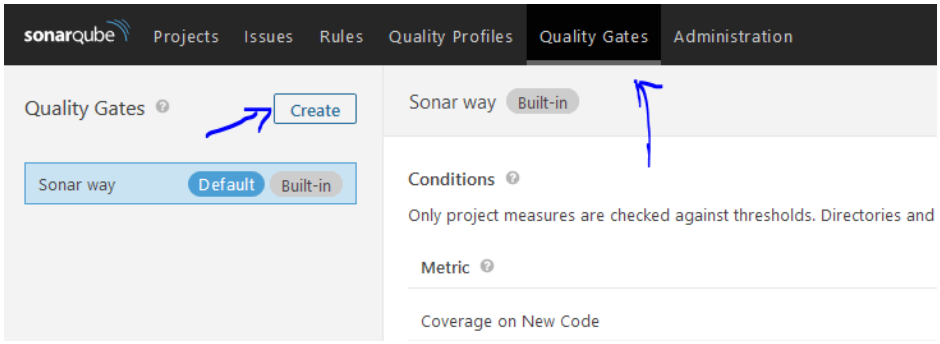
f) It is likely that your project is failing the Quality Gate using the default Sonar Wat configuration.

Which rule or rules are passing, which are failing?

g) Check the current conditions of the quality gate being applied (top bar > Quality Gates).

Create a new Quality Gate configuration by combining a set of criteria. Make the "no issues" rule on new code more flexible (not a best practice).

Explain your choice of metrics are limits.



h) Add some new code to the project and be sure to provide appropriate coverage.

You should run the analysis and have the Quality Gate for new code passing.

8.3 Smelly code hunting challenge

Search for "smelly code" in real projects and document some of the most critical/severe issues you found.

For that, use one of the following strategies:

- use your group project from IES course (from last semester) and conduct an analysis with Sonar. Browse the issues found, focus on those more severe, and **report examples** of "smelly code" for which you found the guidelines from Sonar **most "pedagogical"** (i.e., you were not aware, and you learned from Sonar feedback).
- If you don't have assess to the IES project, consider using an Open Source project or one of the public projects listed at Sonar Cloud (filtering for Java language).

You do not need to include the project used for the case study in the lab submission. Instead, you should discuss samples of "smelly code" found.

8.4 AI-assisted reviews

In this activity, you are expected to use some artificially intelligence engine specialized in coding.

a) If possible, configure GitHub Copilot extension in your IDE; otherwise, you can use an interactive webpage with another engine (Phind,...).

b) Use the IES project from the last semester, if available, or this sample project (*buser*), also implemented by students.

c) Assume you work in a team project and you ere requested to do a code review of code from a peer developer. You don't need to cover all project (suggestion: consider focusing on "REST

controllers" and "services".) You want code to be clean, easy to understand and maintain, and open to future changes.
Ask Copilot to review specific classes in the code.

d) Document the AI "collaboration": which *prompt* strategy did you use? Which points/changes did you find relevant/interesting from the AI suggestions? Which refactoring would you effectively recommend to your college to include in production?
Try to refer to the suggested refactoring practices using their "well-known" names in code refactoring catalogs[1] (e.g.: "Replace Magic Numbers").

e) Code reviews are expected to improve the code and the programmer. How did you feel about this last point, especially compared with the experience with SonarQube?

[1] A more detailed version of the refactoring methods catalogue can be found in Fowler book.