UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

SonarQube

Sergio Di Meglio
sergio.dimeglio@unina.it

Nov 17,2023

Software Engineering

Software Quality Measurement

- *It is a process of quantifying and assessing various aspects of software to ensure that it meets specified requirements and conforms to predefined standards.*
- **GOAL:** evaluate and improve the quality of the software throughout its development life cycle.
- To implement software quality measurement effectively, **Static and Dynamic analysis** approaches are required.

Dynamic Analysis

- Dynamic analysis is a larger term that encompasses various techniques and methods used to **analyze and evaluate software during its execution.**
- Here are some key aspects of dynamic software analysis:
  - **Debugging**
  - **Profiling**
  - **Software Testing**
  - **Etc…**

Static Analysis

- Static code analysis is done **without executing any of the code**.
- Here are key aspects of static software analysis:
  - **Source code analysis,** involves examining source code to find issues such as code smells, potential bugs and security violations.
  - **Code metrics,** generate metrics related to code complexity, code duplications, and other quality indicators.
  - **Dependency analysis.**
  - **Etc…**

Static vs Dynamic Analysis

1. Performs at non-runtime.
2. Works on source code
3. White Box
4. Large amount of time and resources
5. A preventive action
6. Provides more defects
7. Performed before Dynamic analysis

1. Performs at runtime
2. Works on executed code
3. White and Black Box
4. Less compare to
5. A corrective action
6. Lesser defects as compare
7. Performed after Static analysis

Static Analysis Challanges

- **Performance Impact:** Analyzing large codebases or complex alogrithms can be time-consuming and resource-intensive.
- **Complexity Handling**: Complex codebases with intricate structures and numerous dependencies might be challenges to accurately comprehend.
- **DOING IT MANUALLY IS IMPOSSIBLE!**

What is SonarQube?

- Self-managed **automatic code review tool**.
- Provides a dashboard where you can monitor:
  - Quality of your code.
  - Track changes over time
  - Identify potential issues
- It easily integrates into existing workflow (CI pipeline and DevOps platform).

Issues

- During code inspection sonarQube detects three types of issues:
- **BUG:** a coding mistake that can lead to an error or unexpected behavior at runtime. (*Reliability*)
- **VULNERABILITY:** a point in your code that's open to attack (*Security*)
- **CODE SMELL:** a maintainability issue that makes your code confusing and difficult to maintain (*Maintainability*).

Bug Example

A "NullPointerException" could be thrown; "datainizio" is nullable here.

Get permalink

Null pointers should not be dereferenced java:S2259

1 month ago ▾ L61

Bug ▾ Major ▾ Open ▾ Not assigned ▾ 10min effort 0 comments

cert, cwe ▾

Where is the issue? | Why is this an issue?

OOBD_2223_S_17 src/main/java/project/controller/ModificaConferenzaController.java

See all issues in this file

```
55        1 String datainizio = null;
56          String datafine = null;
57          if (DataInizio.getValue() != null && DataFine.getValue() != null) {
58              datainizio = DataInizio.getValue().toString();
59              datafine = DataFine.getValue().toString();
60          }
61          if (nomeconferenza.equals("") || costototale.equals("") || 2 datainizio.equals("") || datafine.equals("")
 || descrizione.equals("") || ente.equals("") || codice.equals("")) {
```

Vulnerability Example

Code Smell Example

Remove this useless assignment to local variable "codConfInfo".

Get permalink

Unused assignments should be removed java:S1854

1 month ago ▾ L169

Code Smell ▾ Major ▾ Open ▾ Not assigned ▾ 15min effort 0 comments

cert, cwe, unused ▾

Where is the issue? | Why is this an issue?

OOBD_2223_S_17 src/main/java/project/controller/InserisciConferenzaController.java

See all issues in this file

```
167         if (selectedItem != null) {
168             String[] SedeSelezionata = selectedItem.split(", ");
169             String codConfInfo = SedeSelezionata[0];
```

Remove this useless assignment to local variable "codConfInfo".

```
170             return SedeSelezionata[0];
```

Issue Severity

Each issue has five different degrees of severity:

- **BLOCKER**
- **CRITICAL**
- **MAJOR**
- **MINOR**
- **INFO**

OLD SEVERITY	MAPPED to
Blocker, Critical	High
Major	Medium
Minor, Info	Low

The severity feature evolved in SonarQube 10.2. The following mapping was applied.

Blocker Issue

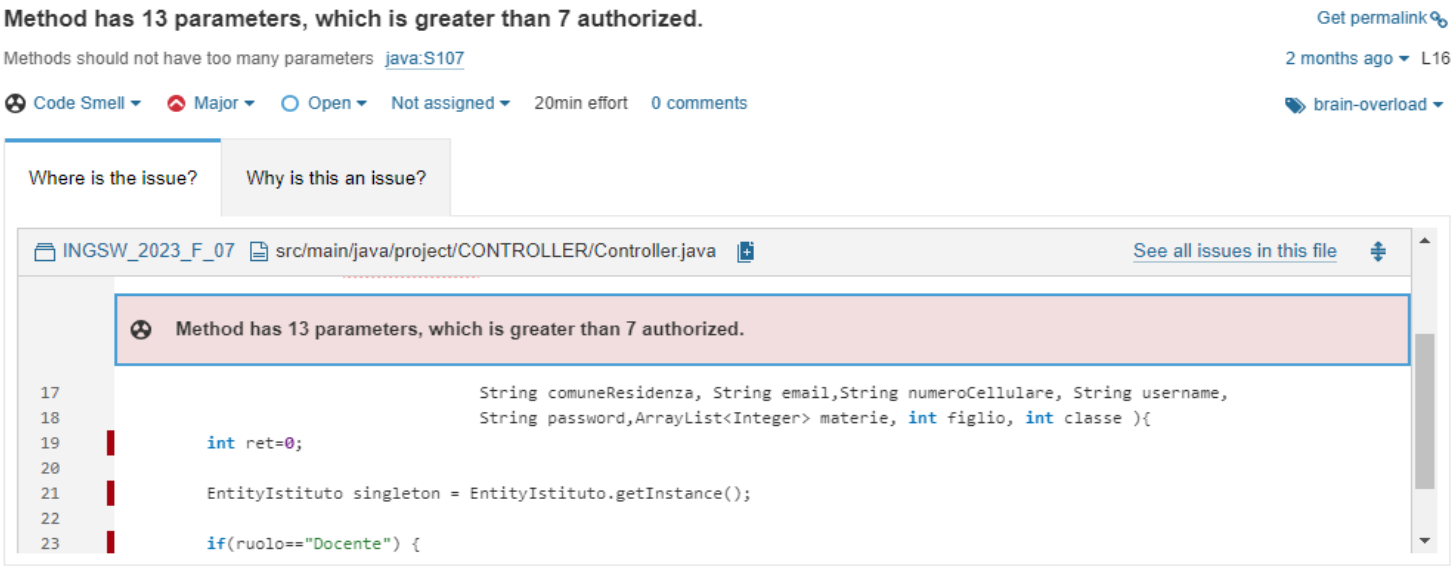
- This is a bug with a high probability of impacting application behavior in production.
- For example, a memory leak, JDBC connections not closed.
- **They need to be fixed immediately.**

Critical Issue

- This is a bug with a low probability to impact the behavior of the application in production or an issue that represents a security flaw. For example an empty catch block or SQL injection.

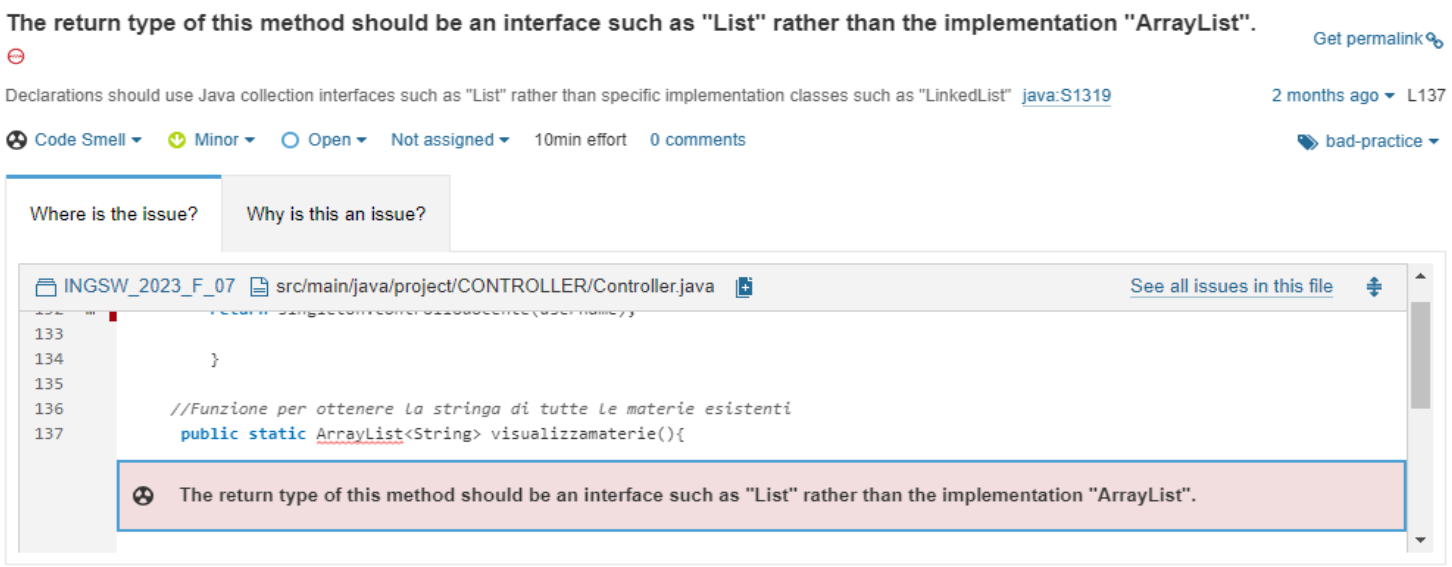
Major Issue

- A quality flaw that can highly impact the developer's productivity.
- For example duplicated code, unused parameters.



Minor Issue

- A quality flaw that can slightly impact the developer's productivity.
- For example too many lines of code, switch condition with less than two case instances.



Info Issue

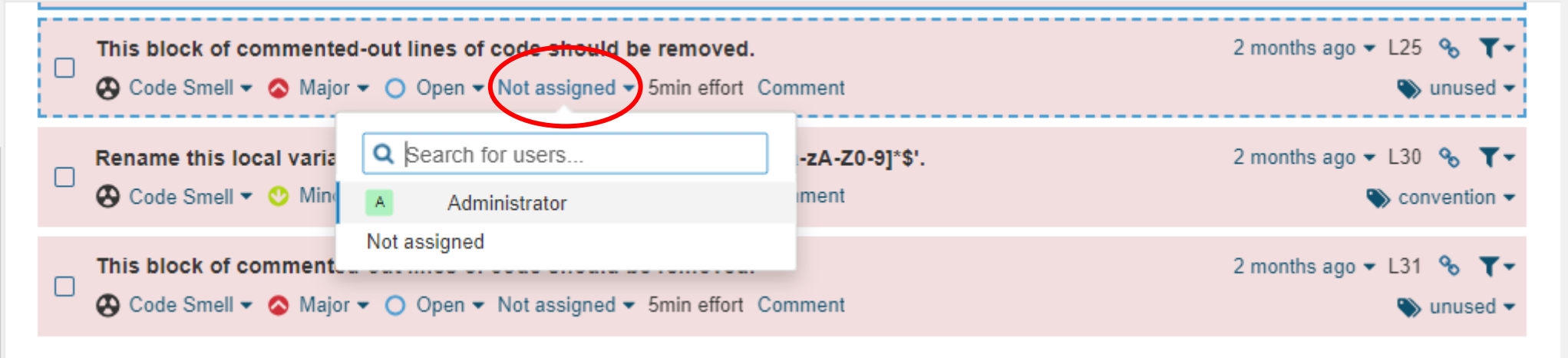
- This is neither a bug nor a quality flaw, but **just an observation**.

Issue Lifecycle

- After creation, issues flow through a lifecycle, taking one of the following statues:
  - **Open:** set by SonarQube on new issues.
  - **Confirmed:** set manually to indicate that the issues is valid.
  - **Resolved:** set manually to indicate that the next analysis should close the issue.
  - **Reopened:** set automatically by SonarQube when a resolved issues hasn't actually been corrected.
  - **Closed:** set automatically by SonarQube for automatically created issues.

Issue Lifecycle

Issue Assignment



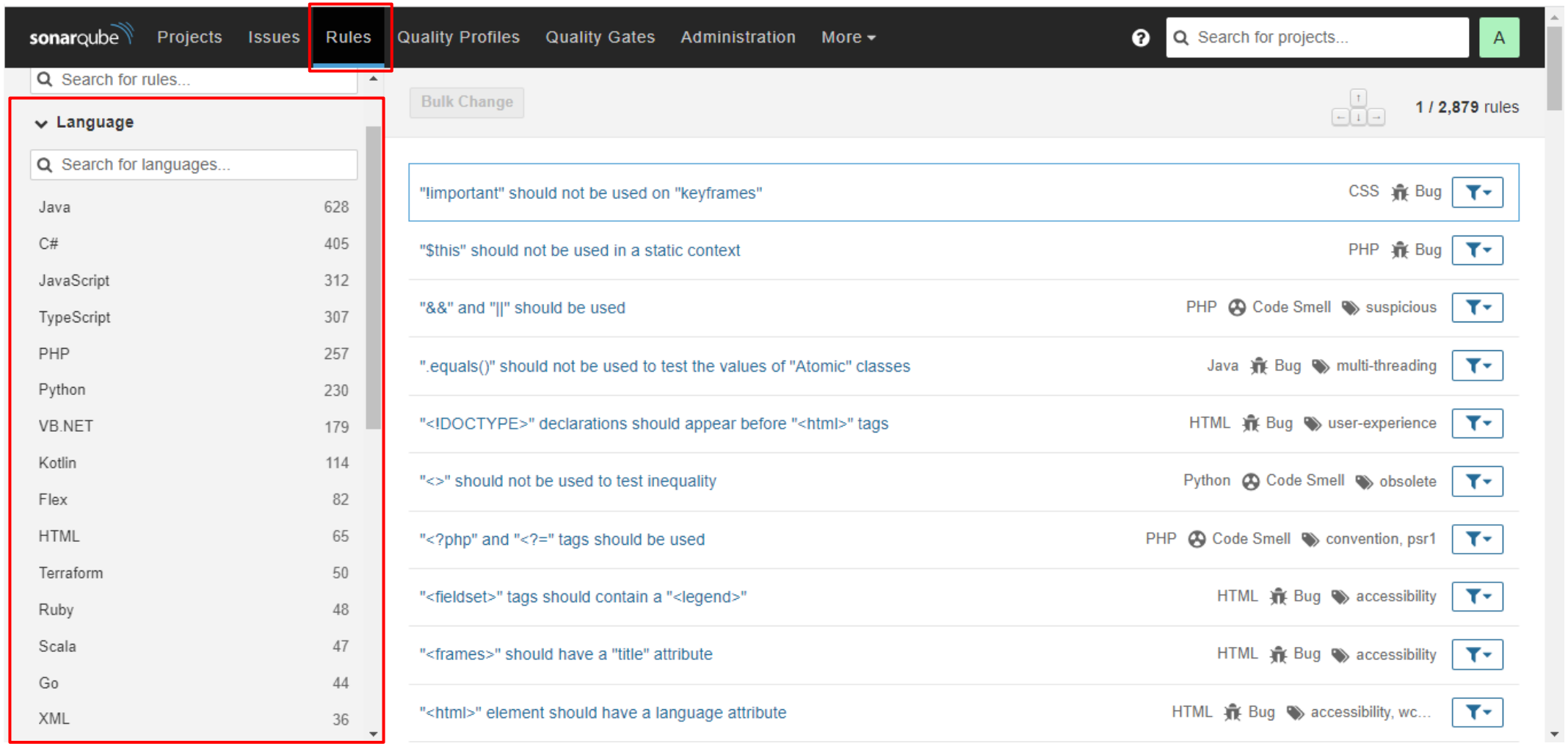
- The issues are automatically assigned during analysis to the last committer on the issues line.
- **Only if the commiter can be correlated to a SonarQube user.**
- Otherwise, it can be assigned manually.

Rules

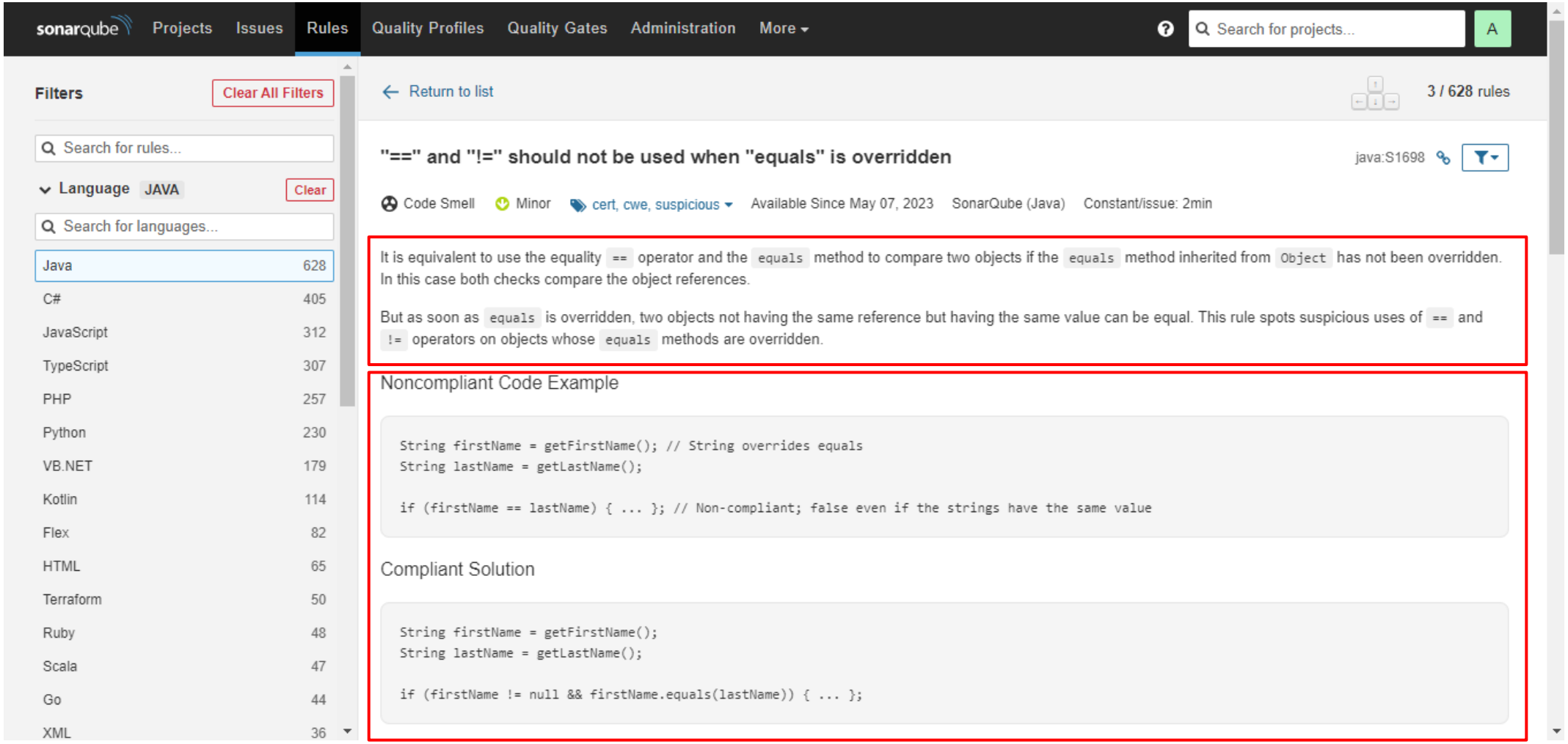
- SonarQube identifies these issues through the use of a specific **rule sets**.
- **What is a rule?** A coding standard or practice which should be followed.
- In case the analyzed source code violates a coding rule, SonarQube generates an issue.

Multi-Language Support

Multi-Language Support

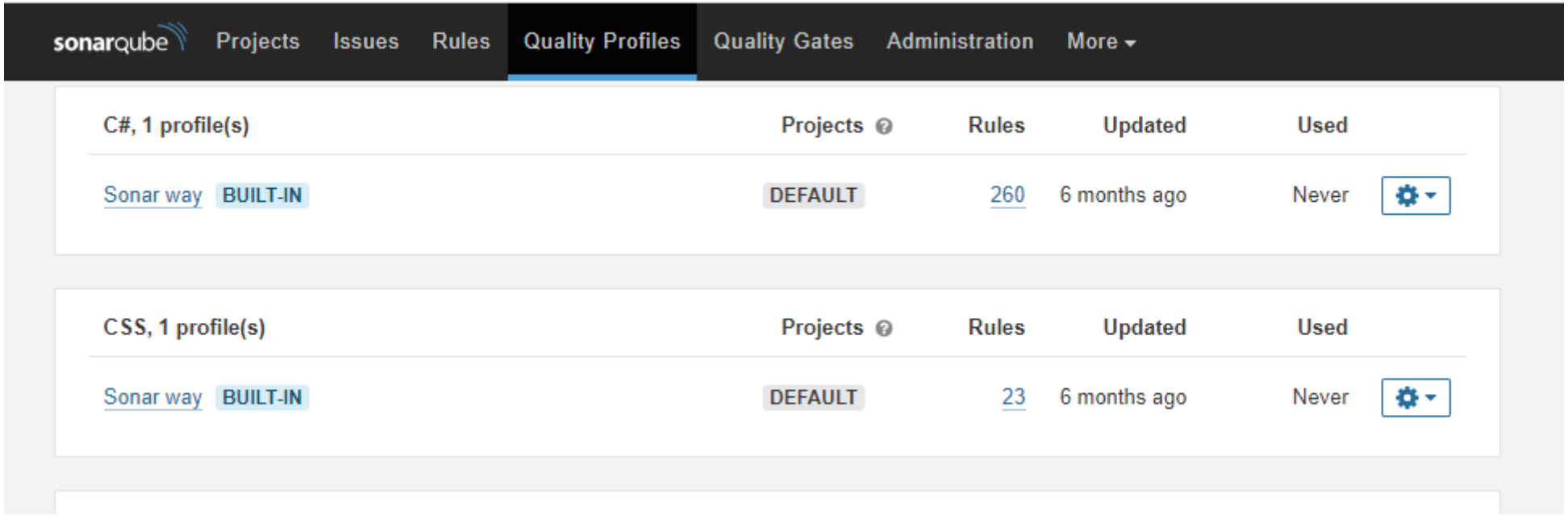


Rules Details

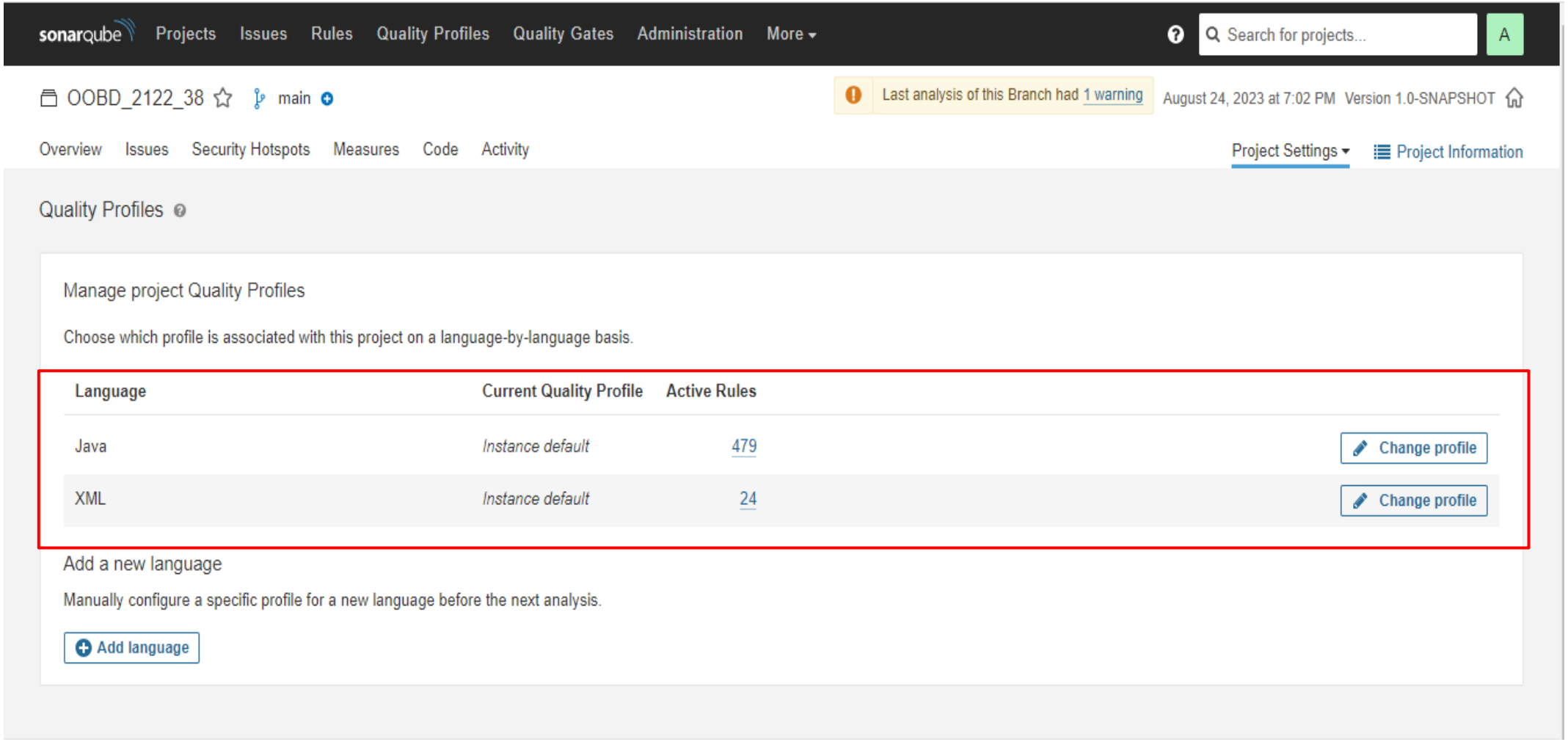


Quality Profiles

- They defines the set of **rules** to be applied during code analysis.
- Every project has a quality profile for each supported language.



Quality Profiles



Quality Gate

- **Is my project ready for release?**
- Quality gates is a set of condition to enforce a quality policy in your organization.
- For example:
  - No new blocker issues.
  - Code coverage on new code greater than 80%.

Defining Quality Gate (1)

- Each quality gate condition is a combination of:
  - a measure.
  - a comparison operator.
  - an error value.
- For instance, a condition might be
  - measure: Blocker issue.
  - comparison operator: >
  - error value: 0

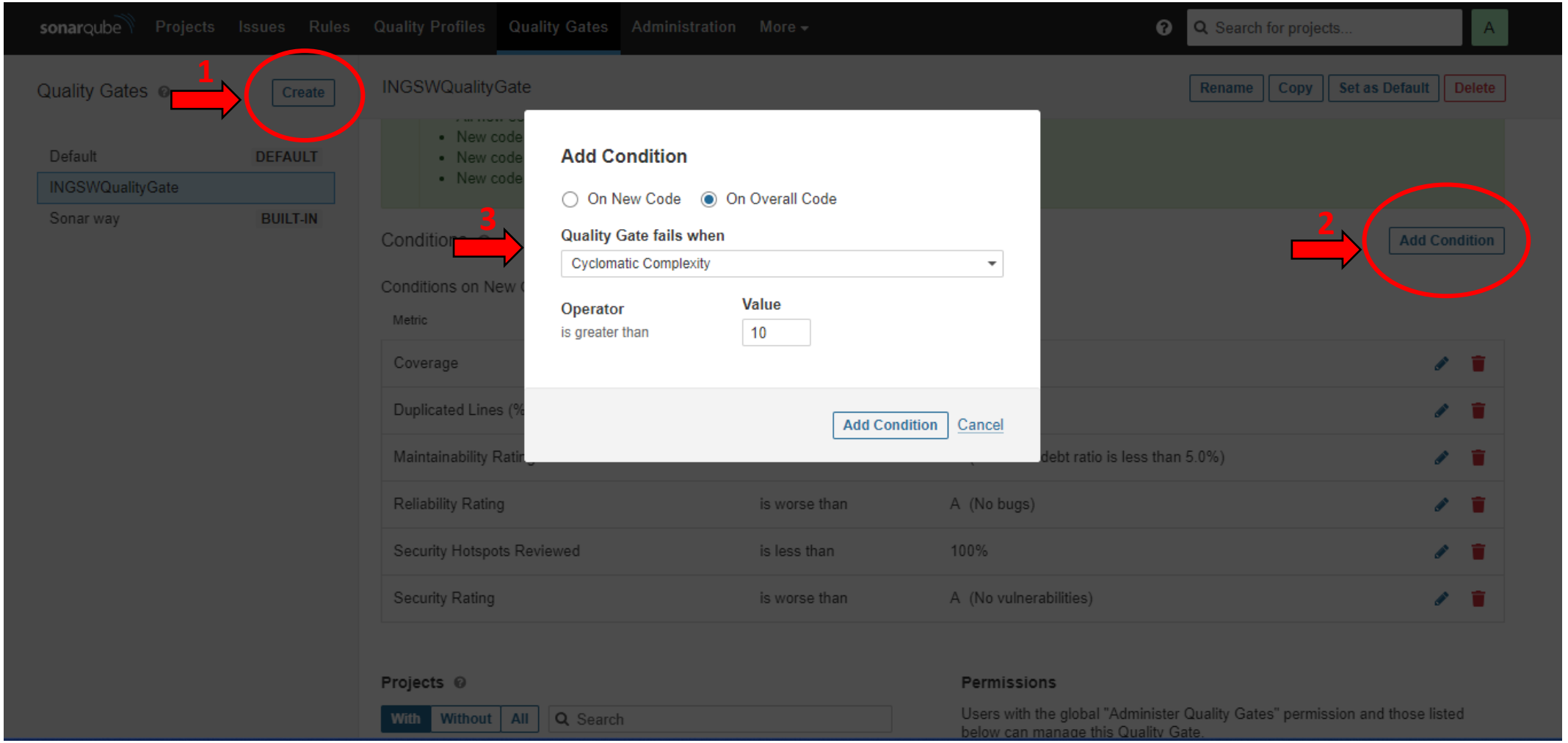
Default Quality Gate

Conditions

Conditions on New Code

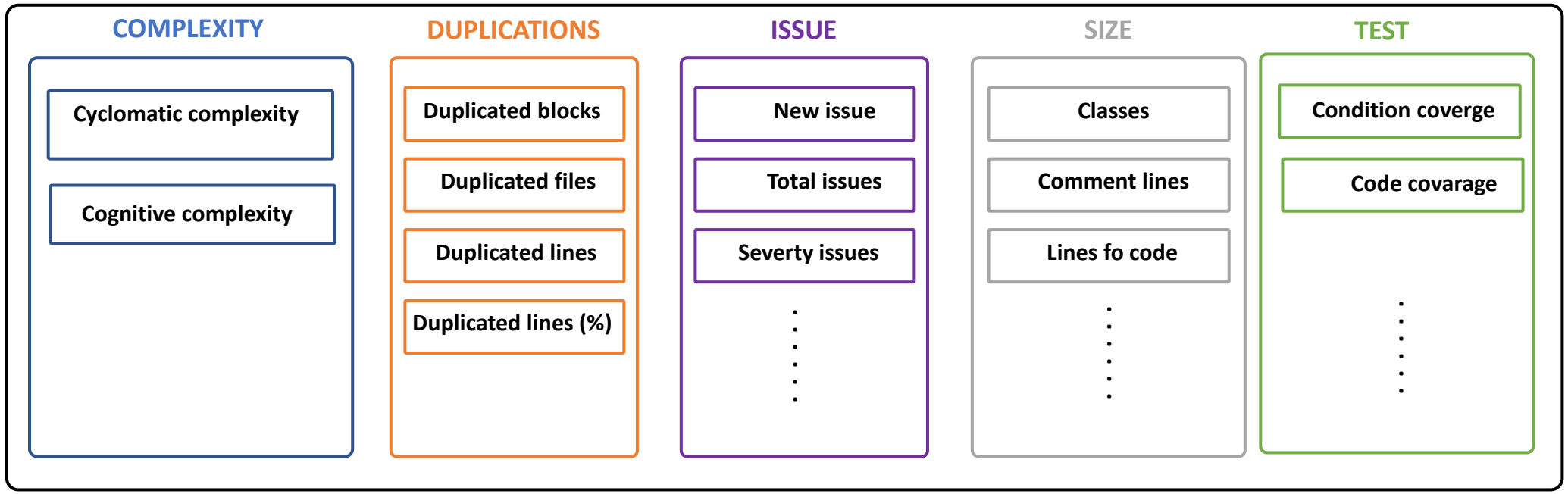
Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A (Technical debt ratio is less than 5.0%)
Reliability Rating	is worse than	A (No bugs)
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A (No vulnerabilities)

Defining Quality Gate (2)



Metrics

- SonarQube calculates a number of useful metrics.



- https://docs.sonarsource.com/sonarqube/latest/user-guide/metric-definitions/

Technical Debt

- A measure of **effort** to fix all issues.

SonarQube Installation (1)

- The easiest way is to use Docker images provided on the Docker Hub site.
- https://hub.docker.com/_/sonarqube/tags

- Pull the selected docker image

```
docker pull sonarqube:9.9.1-communit
```

SonarQube Installation (2)

- Creating the following volumes:

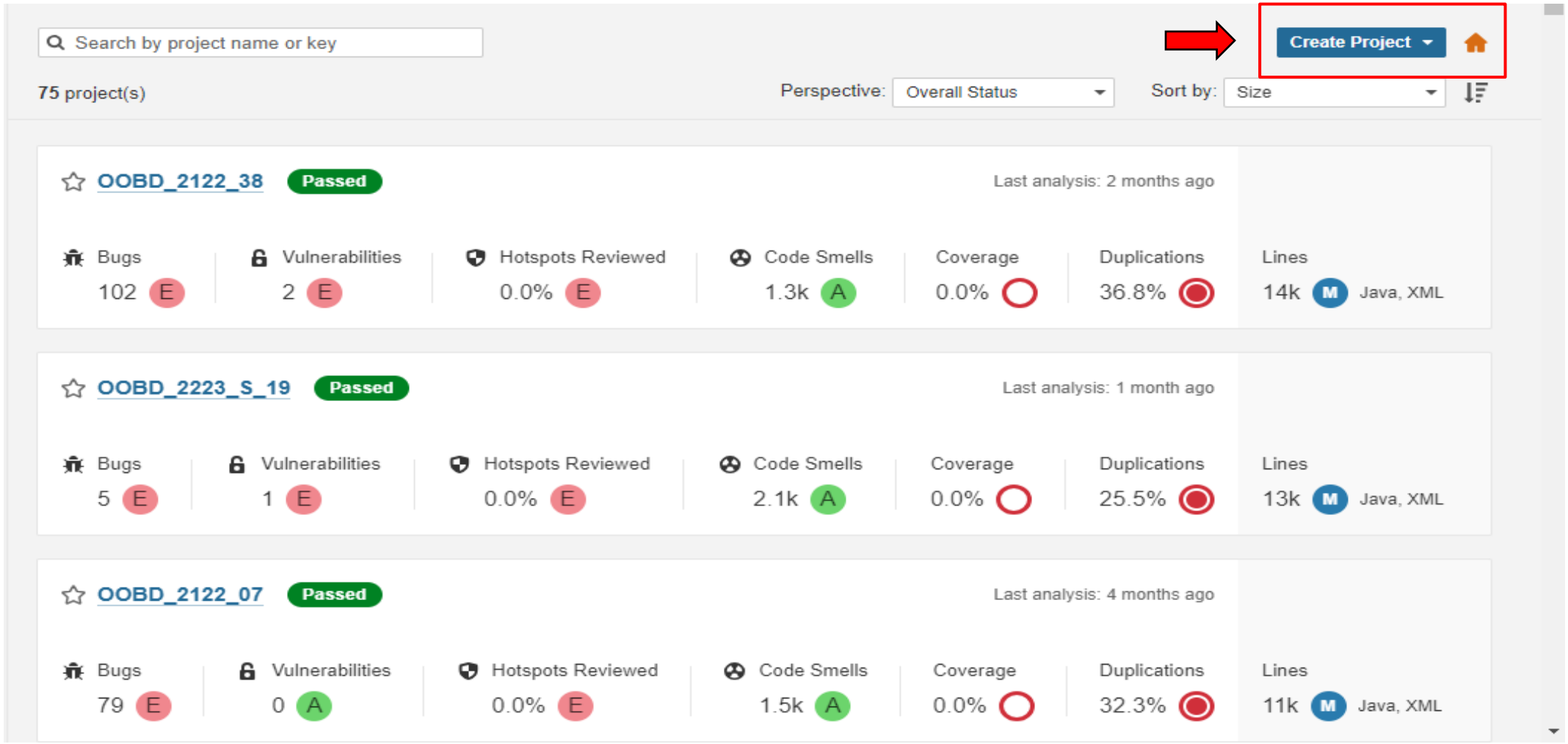
```
> docker volume create --name sonarqube_data
> docker volume create --name sonarqube_logs
> docker volume create --name sonarqube_extensions
```

- Run the image:

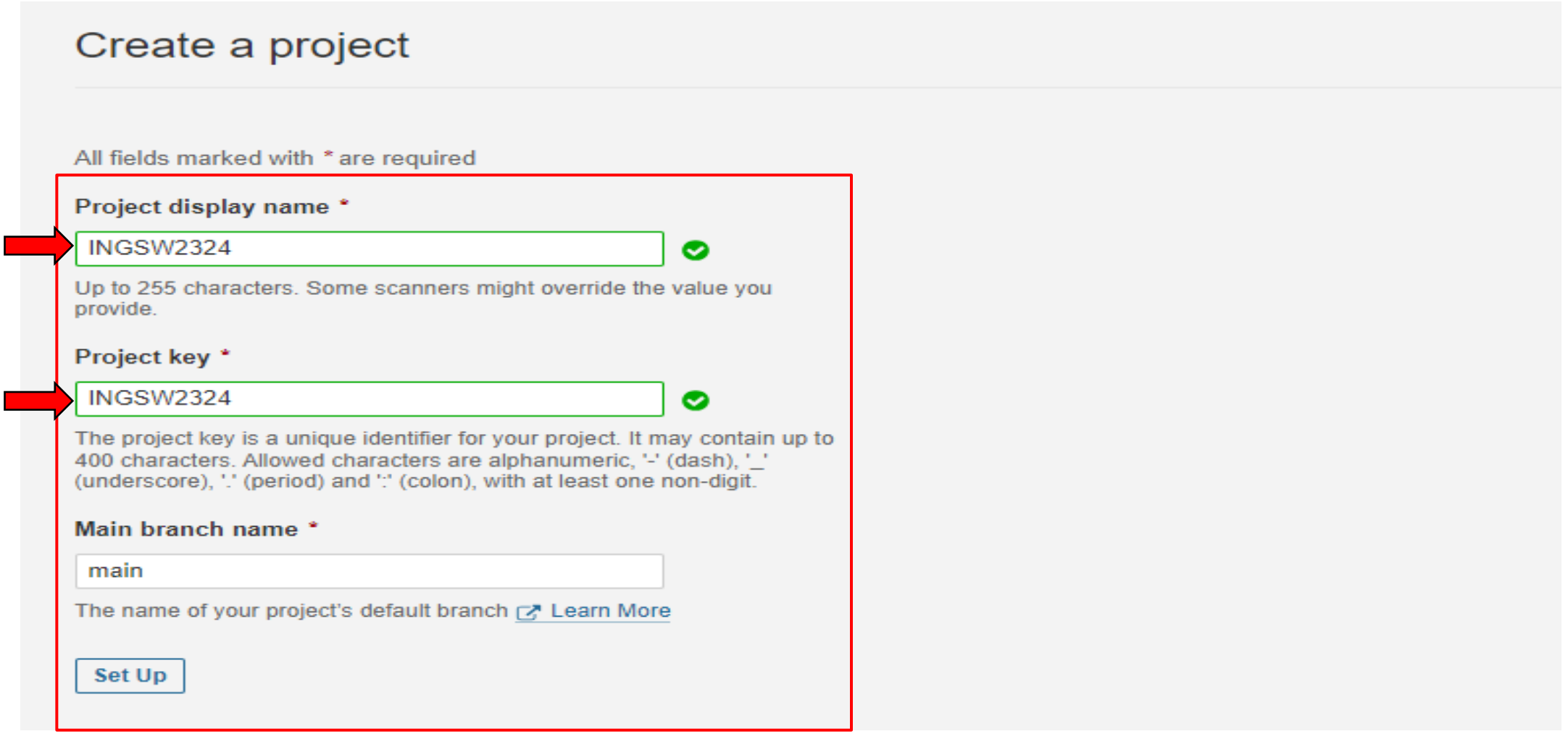
```
docker run –d --name sonarqube –p 9000:9000
-v sonarqube_data:/opt/sonarqube/data
-v sonarqube_extensions:/opt/sonarqube/extensions
-v sonarqube_logs:/opt/sonarqube/logs  sonarqube:9.9.1-community
```

SonarScanner for Maven

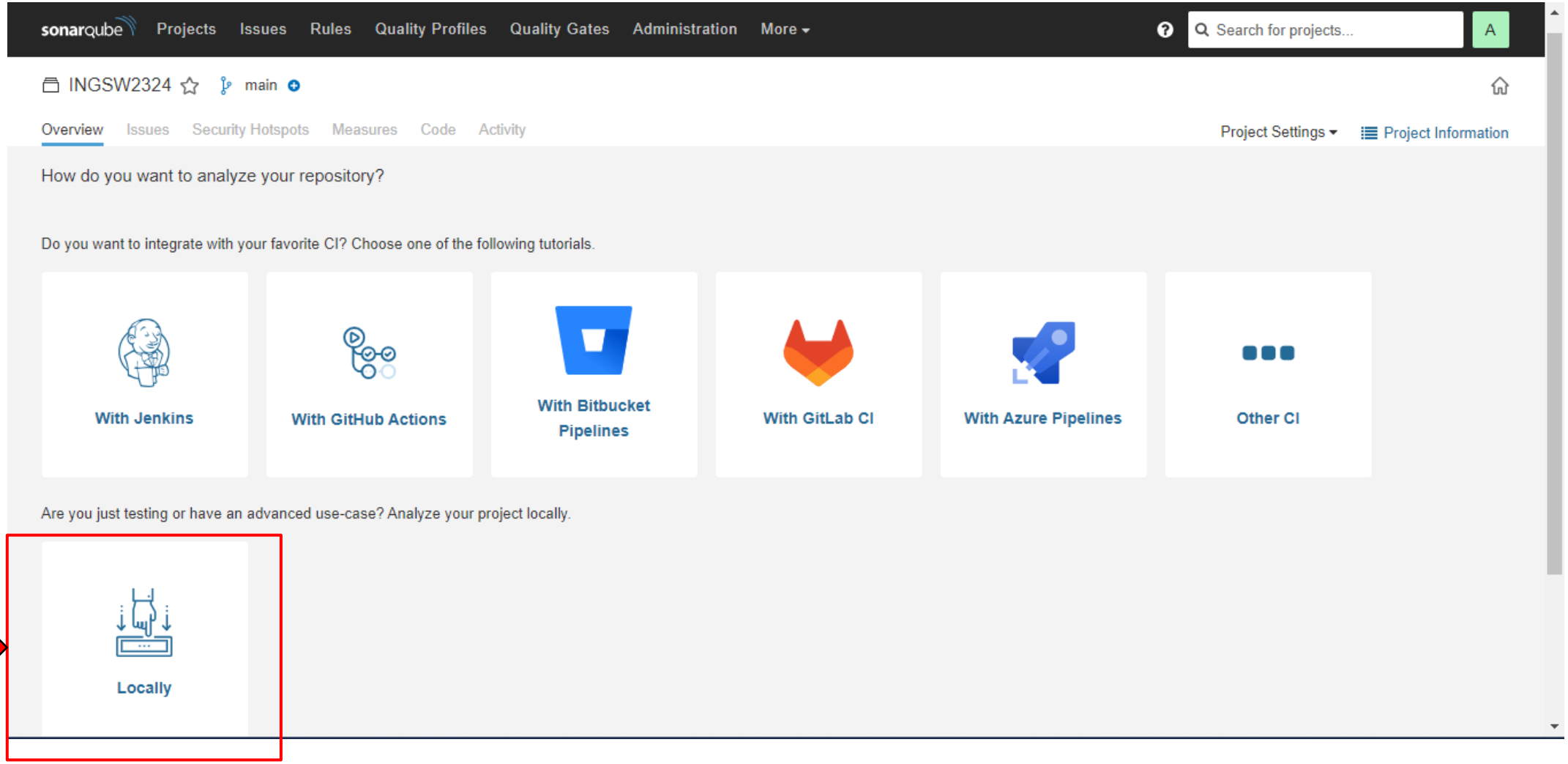
Create a New SonarProject (1)



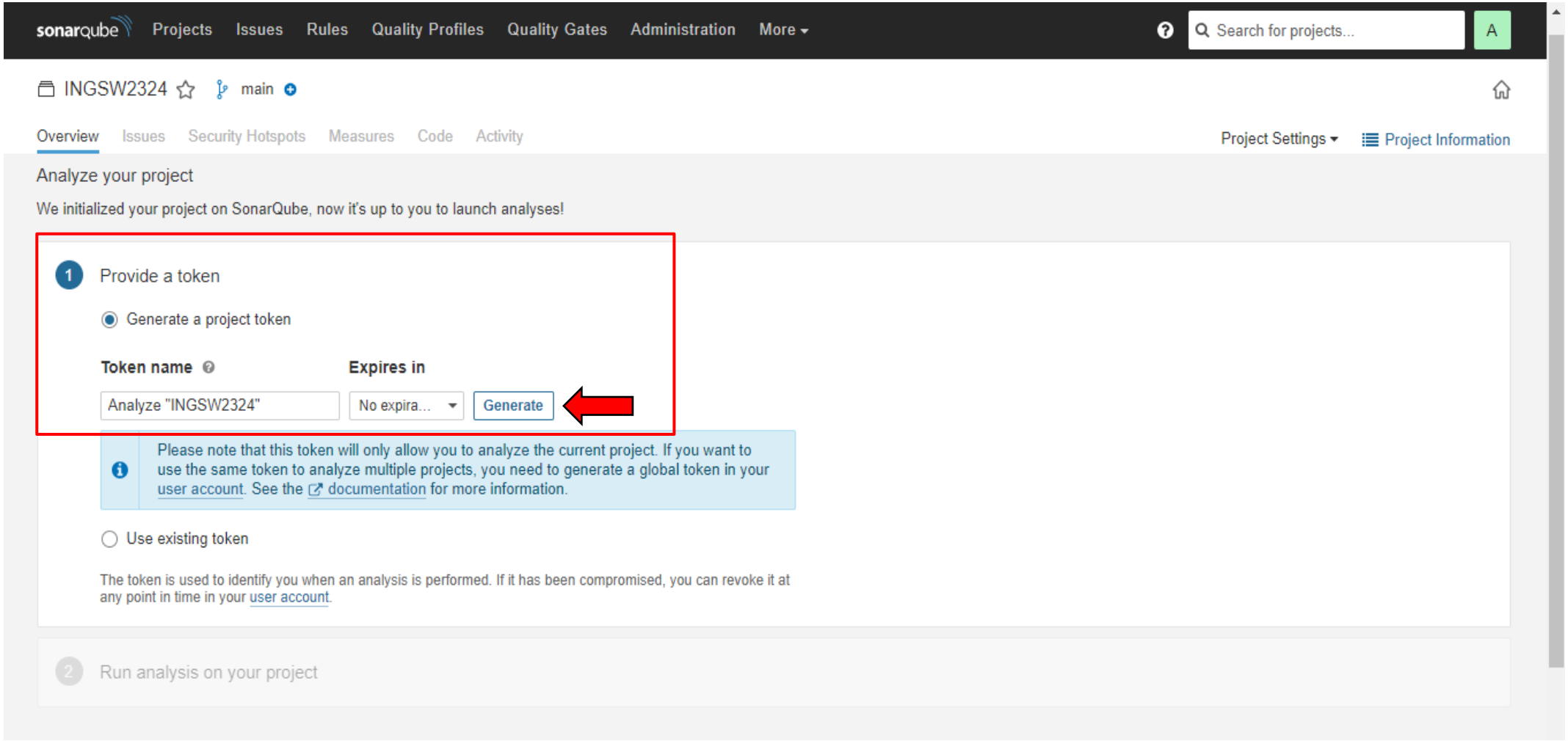
Create a New SonarProject (2)



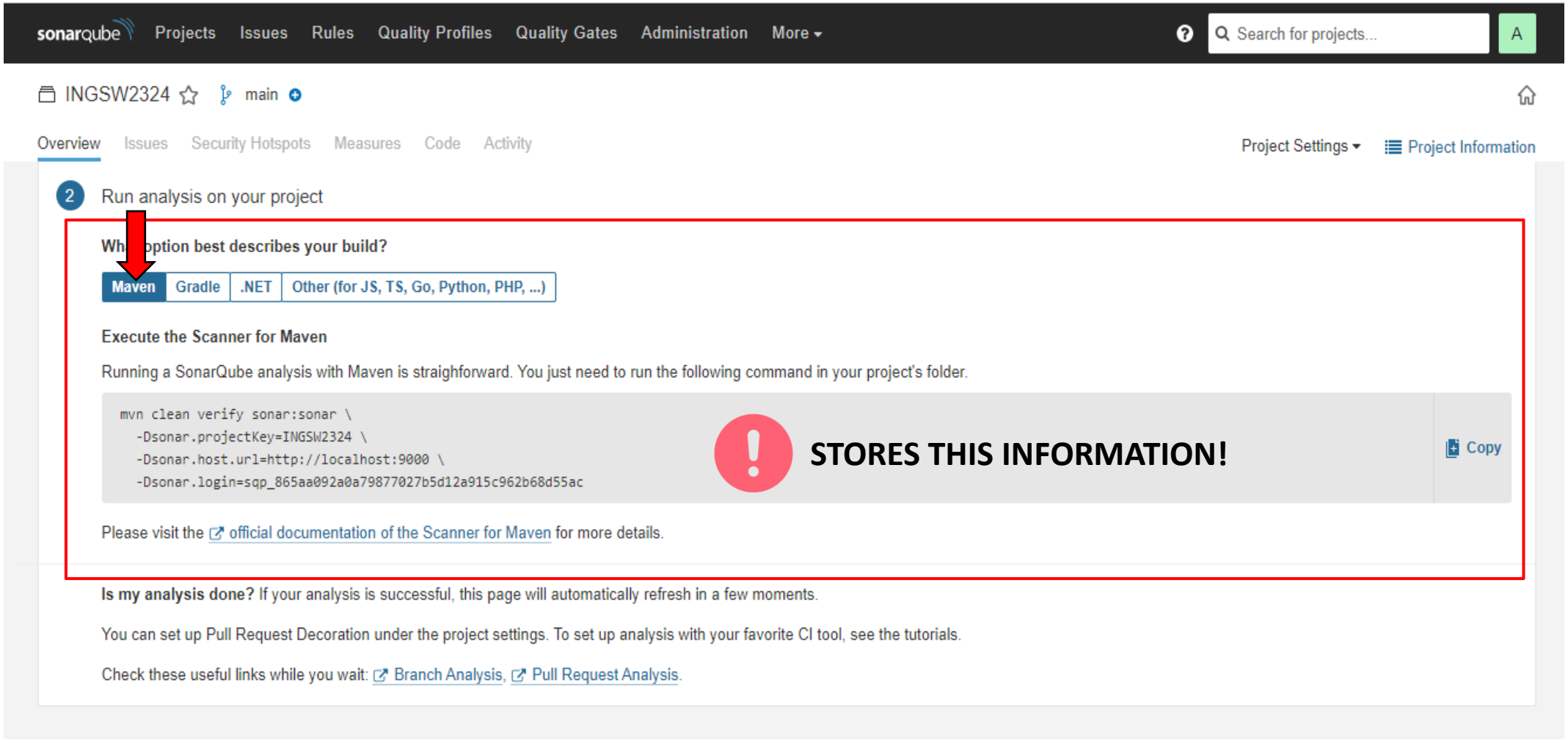
Create a New SonarProject (3)



Create a New SonarProject (4)



Create a New SonarProject (5)



Modify Maven POM (1)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>OOBD_2122_1</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>

  <name>OOBD_2122_01</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <log4j2-version>2.19.0</log4j2-version>
    <sonar.projectKey>INGSW_LECTURE_EXAMPLE</sonar.projectKey>
    <sonar.login>sqp_a7672d91524e81baec9f017654b3649e94cfb98c</sonar.login>
    <sonar.host>http://localhost:9000</sonar.host>
  </properties>
```

Set the previously generated information as a property element within the POM.

- https://docs.sonarsource.com/sonarqube/9.8/analyzing-source-code/analysis-parameters/

Modify Maven POM (2)

```
84  <build>
85    <defaultGoal>install</defaultGoal>
86    <plugins>
87      <plugin>
88        <groupId>org.sonarsource.scanner.maven</groupId>
89        <artifactId>sonar-maven-plugin</artifactId>
90        <version>3.9.1.2184</version>
91        <executions>
92          <execution>
93            <goals>
94              <goal>sonar</goal>
95            </goals>
96            <phase>verify</phase>
97          </execution>
98        </executions>
99      </plugin>
```

Run SonarQube Analyzer

```
> mvn verify
```