UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SOFTWARE ENGINEERING – LECTURE 35

Continuous Code Inspection with SonarQube

Prof. Luigi Libero Lucio Starace

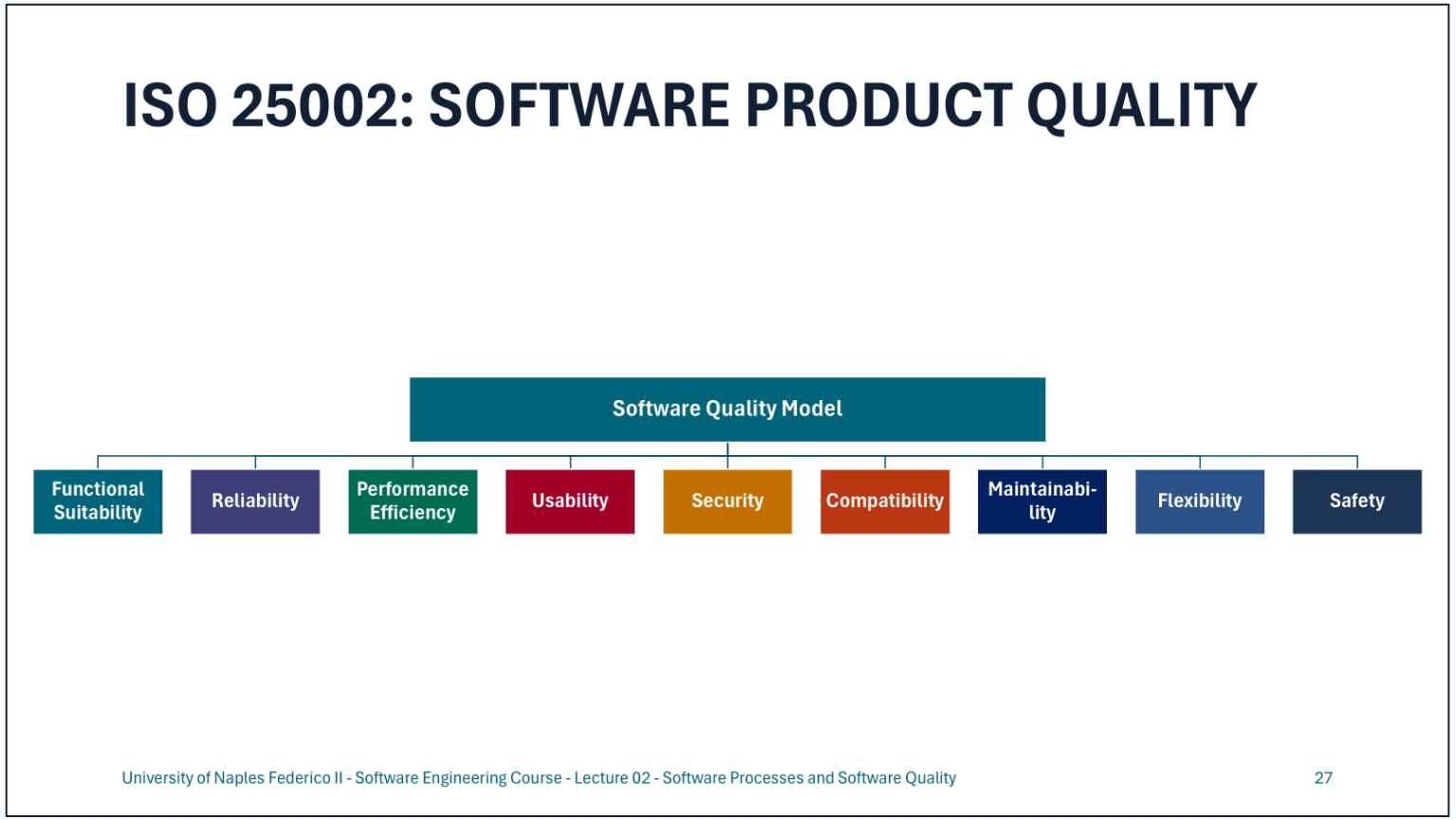
luigiliberolucio.starace@unina.it

https://luistar.github.io

https://www.docenti.unina.it/luigiliberolucio.starace

(A look back on) Software Quality

- Back in Lecture 2, we discussed about **Software Quality**

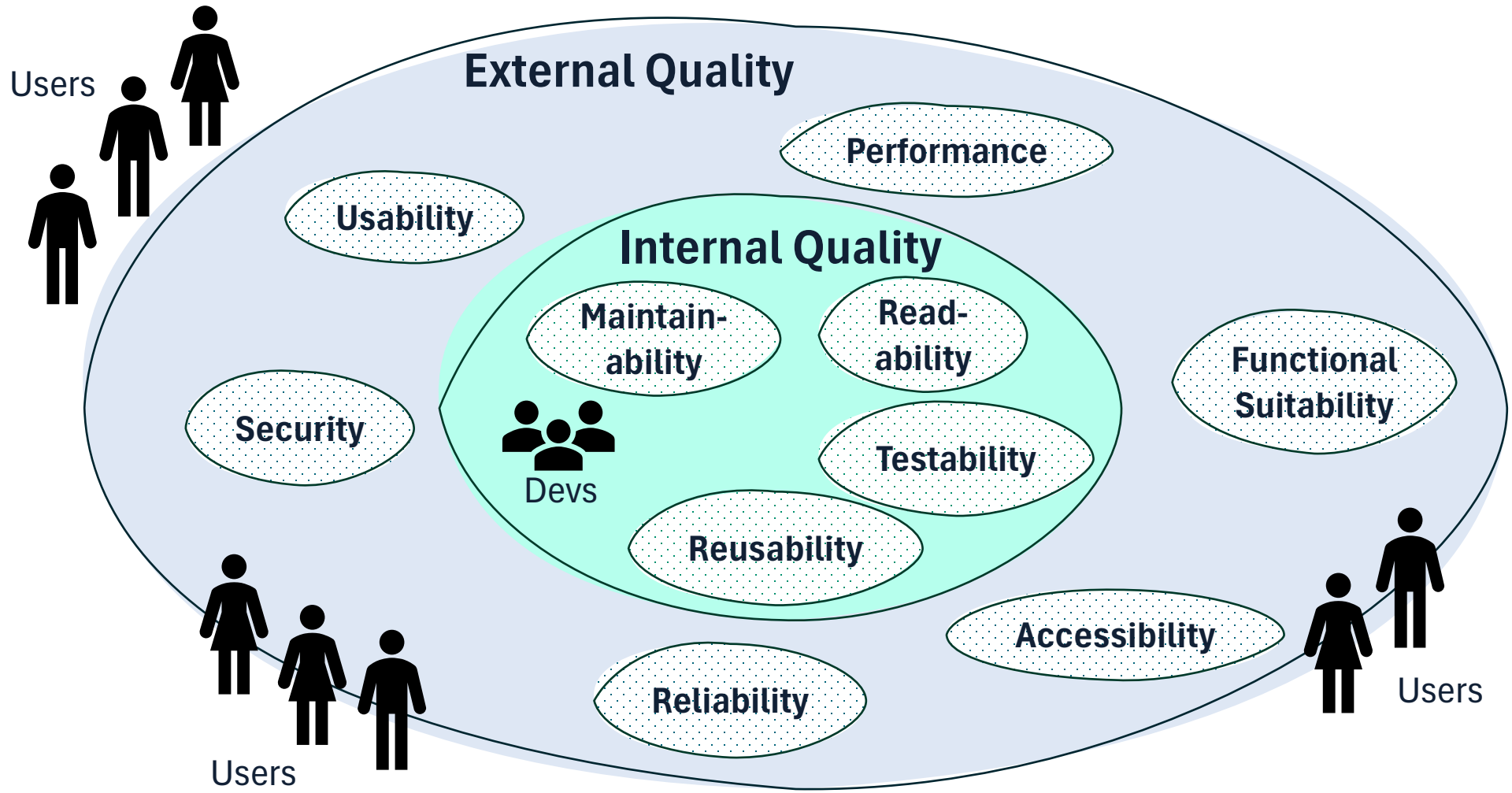


Internal and External Software Quality

Quality characteristics can also be conceptually organized in:

- **Internal Software Quality** characteristics
  - Quality as perceived by developers
  - A software with high internal quality has a codebase which is easy to read, understand and evolve
- **External Software Quality** characteristics
  - Quality as perceived by users
  - E.g.: usability, functional suitability, efficiency

Internal and External Software Quality



Internal and External Software Quality

- External quality is perceived **directly** by users and customers
- Internal quality is **not directly** perceived by users and customers
  - But it has a direct impact on the costs required to evolve a software system
  - It also has a direct impact on some external qualities. For example, **testability** has an impact on guaranteeing **functional correctness**

House with a beautiful exterior (external q.), but shaky foundation (internal q.)

Origins of poor internal quality

- **Pressure to meet fast-paced deadlines**
- **Inadequate knowledge**: Developers need more experience or training.
- **Manual issue remediation**: Without automated tools, identifying and fixing issues can be inconsistent and error-prone.
- **Inconsistent coding styles**: Varied coding practices within a team can result in a codebase that’s difficult to update efficiently.
- **AI coding assistants**: While promising efficiency, these tools can introduce buggy and insecure code if not properly managed

Impact of poor internal quality

- **Reduced maintainability and scalability**: Bad code is hard to understand and modify, making it difficult to evolve.
- **Increased bug count and technical debt**: Poorly written code is prone to bugs, contributing to technical debt that accumulates over time.
- **Decreased productivity and efficiency**: Developers spend more and more time deciphering and fixing bad code, diverting focus from innovation and new functionality.
- **Increased costs and risks**: The cumulative impact of bad code results in higher maintenance costs, frequent bug fixes, rework, and increased technical debt. Additionally, it poses risks to software reliability, security, and stability, leading to reputational damage and compliance issues.

Costs of Poor Internal Software Quality

- The Consortium for IT Software Quality reported that, in 2022 [1]:
  - Unsuccessful development projects costed up to **$260 billion**
  - The cost of poor software quality in the U.S. grew to at least **$2.41 trillion**
  - The accumulated technical debt has grown to **$1.52 trillion**
- According to the Standish Group's CHAOS Report [2], **only 31% of software projects are completed on time and within budget**, with bad code being a significant factor.

[1] https://www.it-cisq.org/wp-content/uploads/sites/6/2022/11/CPSQ-Report-Nov-22-2.pdf
[2] https://www.csus.edu/indiv/v/velianitis/161/chaosreport.pdf

Measuring Software Quality

- Remember:
  - *You can’t manage what you can’t measure*
  - *You can’t improve what you can’t measure*
- We already discussed ways to measure external quality attributes
- **How do we manage and deal with internal quality?**

Code Inspections

- One way would be to perform **code inspections**
- A dedicated team goes over the source code, reads it, and ensures it adheres to principles detailed in a given **checklist**
- For example, for each Java source file
  - ❑ There is no duplicated code
  - ❑ All method names use the camel case naming convention
  - ❑ No method has more than 5 arguments
  - ❑ All variable names use the camel case naming convention
  - ❑ All opened resources are properly closed
  - ❑ No method is longer than 50 LoCs
  - ❑ …

Code Inspections

Performing manual code inspection does not scale well

- What if our codebase is **large (e.g.: 100k+ LoCs)**?
- What if changes are made multiple (e.g.: 1k+) times a day?

- Performing inspections manually is **unfeasible**!
  - Luckily, Software Engineers developed tools to assist with that
  - You'll see that there's way more to development than a good old IDE!

Automated Code Inspections Solutions

- Today, we'll take a look at a family of widely-used automated code inspection solutions by SonarSource

Issues and Rules

- Code is automatically processed to detect **issues**
- **Issues** are basically violations of rules
- The set of rules to apply is customizable
  - The default rules for Java code are are available here
- There are three types of issues:

 **Bug**: Mistakes that can lead to errors or unexpected behavior at runtime.

 **Vulnerability**: A point in your code that's open to attack.

 **Code Smell**: A maintainability issue that makes your code confusing and/or difficult to maintain.

Issue Severity

- **BLOCKER**: Bug with a high probability to impact the application in production. E.g.: memory leaks, or unclosed JDBC connections.
- **CRITICAL**: Either bugs with low probability to impact the application in production or an issue that represents a security flaw. E.g.: empty catch blocks or SQL injection.
- **MAJOR**: Quality flaws that can highly impact developer productivity. E.g.: Untested code, duplicated code, or unused parameters.
- **MINOR**: Quality flaws that can slightly impact developer productivity. E.g.: lines that are too long, "switch" statements with less than 3 cases.
- **INFO**: Neither a bug nor a quality flaw, just a finding.

A few words before we start

- You now know what a code smell is
- How many code smells would you say there were in the average object orientation project?

Scan the QR or use link to join

https://forms.office.com/e/hFz9mX8NZB

Copy link

0 response submitted

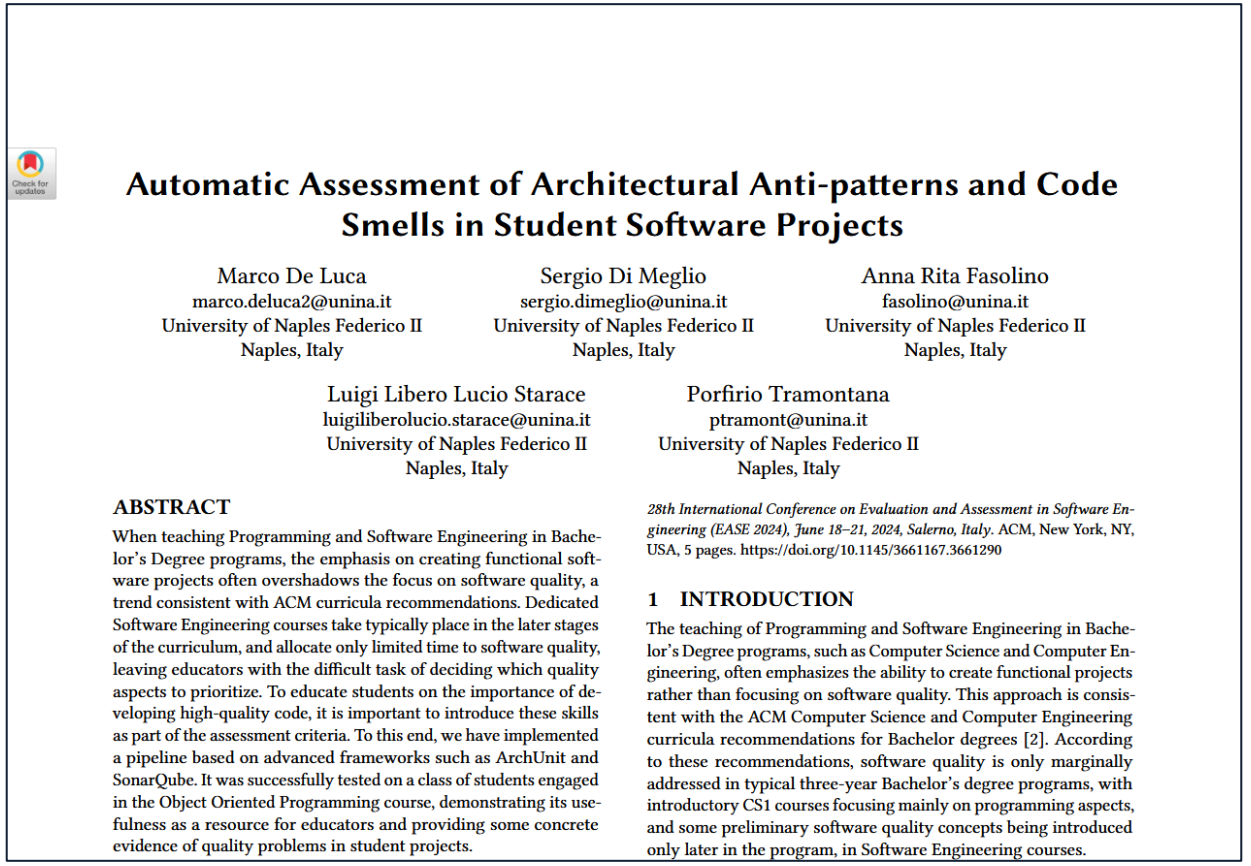
Quanti code smell sono presenti nel progetto medio di Object Orientation?

10-100	100-500	500-1500
1500-5k	5k-10k	10k+

Treemap | Bar

1 of 1

Wandering how do I know that?



Automatic Assessment of Architectural Anti-patterns and Code Smells in Student Software Projects

Marco De Luca
marco.deluca2@unina.it
University of Naples Federico II
Naples, Italy

Sergio Di Meglio
sergio.dimeglio@unina.it
University of Naples Federico II
Naples, Italy

Anna Rita Fasolino
fasolino@unina.it
University of Naples Federico II
Naples, Italy

Luigi Libero Lucio Starace
luigiliberolucio.starace@unina.it
University of Naples Federico II
Naples, Italy

Porfirio Tramontana
ptramont@unina.it
University of Naples Federico II
Naples, Italy

ABSTRACT

When teaching Programming and Software Engineering in Bachelor's Degree programs, the emphasis on creating functional software projects often overshadows the focus on software quality, a trend consistent with ACM curricula recommendations. Dedicated Software Engineering courses take typically place in the later stages of the curriculum, and allocate only limited time to software quality, leaving educators with the difficult task of deciding which quality aspects to prioritize. To educate students on the importance of developing high-quality code, it is important to introduce these skills as part of the assessment criteria. To this end, we have implemented a pipeline based on advanced frameworks such as ArchUnit and SonarQube. It was successfully tested on a class of students engaged in the Object Oriented Programming course, demonstrating its usefulness as a resource for educators and providing some concrete evidence of quality problems in student projects.

28th International Conference on Evaluation and Assessment in Software Engineering (EASE 2024), June 18–21, 2024, Salerno, Italy. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3661167.3661290

1 INTRODUCTION

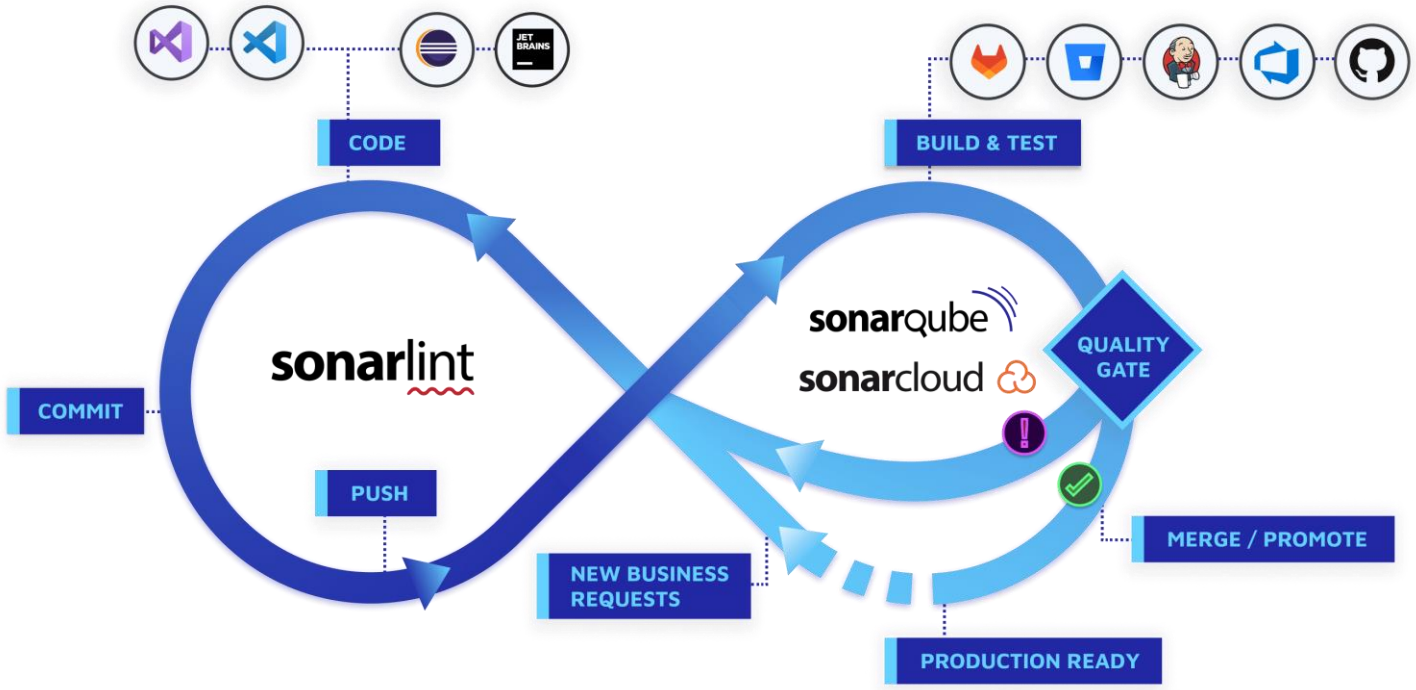
The teaching of Programming and Software Engineering in Bachelor's Degree programs, such as Computer Science and Computer Engineering, often emphasizes the ability to create functional projects rather than focusing on software quality. This approach is consistent with the ACM Computer Science and Computer Engineering curricula recommendations for Bachelor degrees [2]. According to these recommendations, software quality is only marginally addressed in typical three-year Bachelor's degree programs, with introductory CS1 courses focusing mainly on programming aspects, and some preliminary software quality concepts being introduced only later in the program, in Software Engineering courses.

https://dl.acm.org/doi/pdf/10.1145/3661167.3661290

Automated Code Inspection approaches

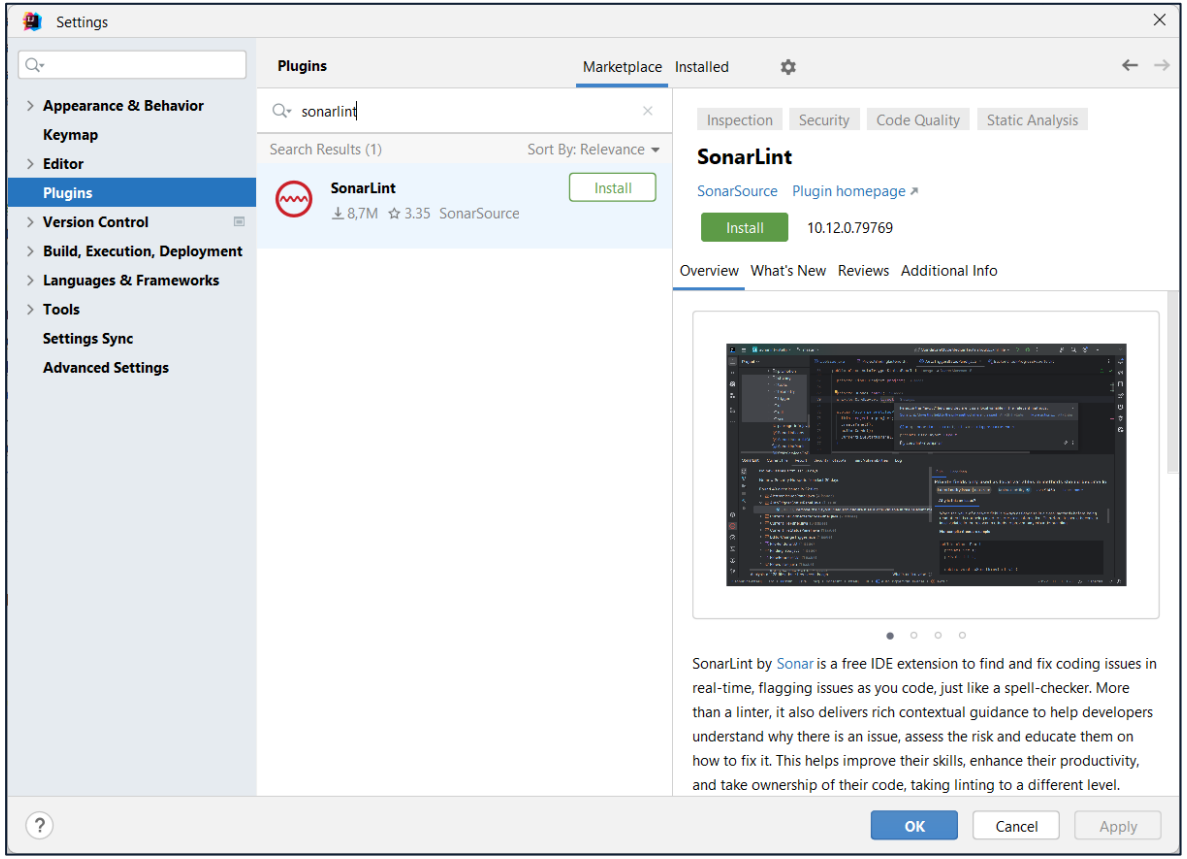
Two (not mutually exclusive) approaches are possible:

- We run the automated code inspections in our IDE, **while** we write code
- Automated code inspections are executed **after** we write code

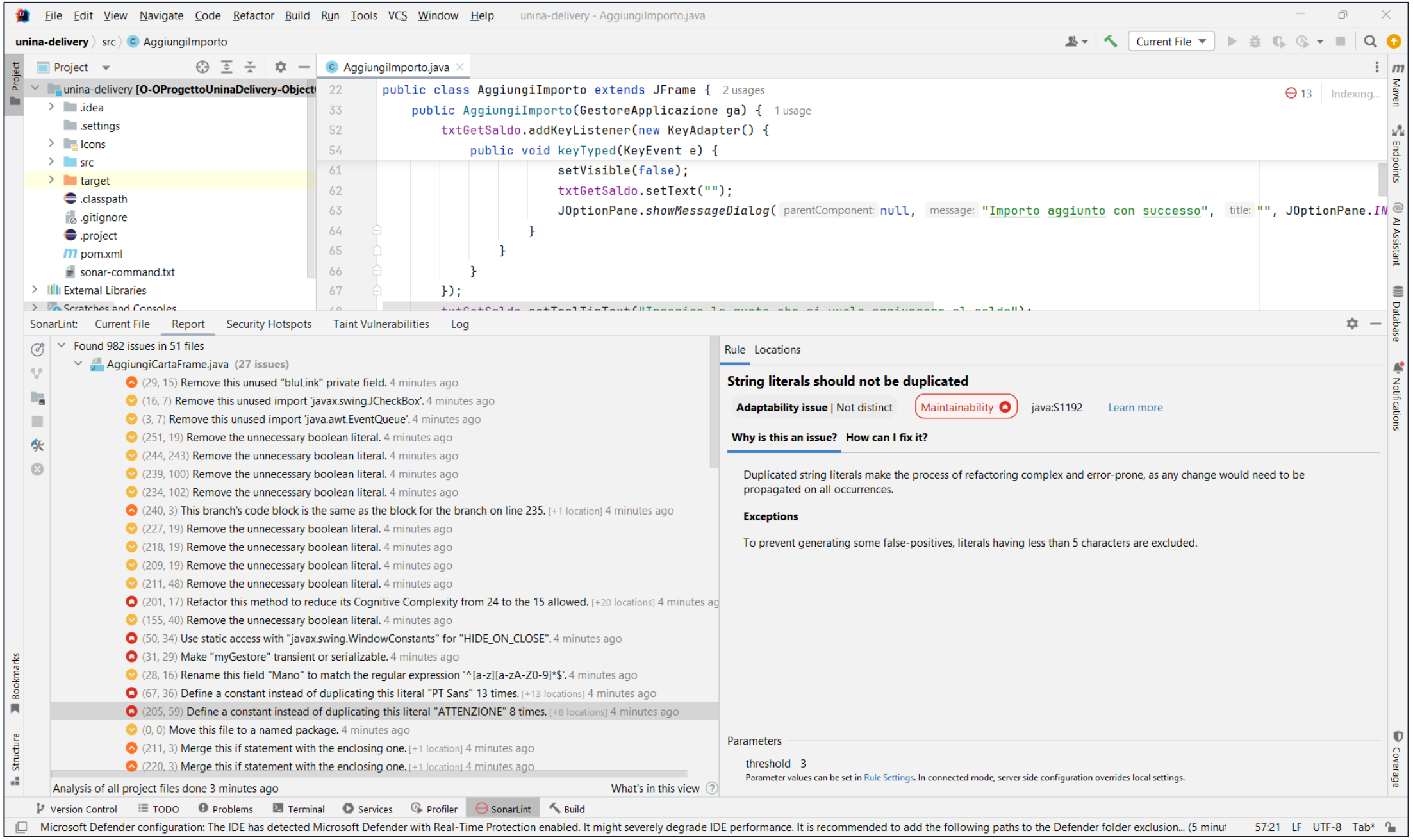


SonarQube IDE (previously SonarLint)

- **Clean as you code** approach
- Detect issues as soon as we are writing the code
- Free IDE plugin available for:
  - Jetbrains IDEs
  - Visual Studio
  - Visual Studio Code
  - Eclipse



Installing SonarQube IDE in Intellij IDEA


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
unina-delivery - AggiungiImporto.java
unina-delivery src AggiungiImporto
Current File
Project
unina-delivery [O-OProgettoUninaDelivery-Object
.idea
.settings
Icons
src
target
.classpath
.gitignore
.project
pom.xml
sonar-command.txt
External Libraries
Scratches and Consoles
AggiungiImporto.java
public class AggiungiImporto extends JFrame { 2 usages
public AggiungiImporto(GestoreApplicazione ga) { 1 usage
txtGetSaldo.addKeyListener(new KeyAdapter() {
public void keyTyped(KeyEvent e) {
setVisible(false);
txtGetSaldo.setText("");
JOptionPane.showMessageDialog(parentComponent: null, message: "Importo aggiunto con successo", title: "", JOptionPane.IN
}
}
}
});
13
Indexing...
SonarLint: Current File Report Security Hotspots Taint Vulnerabilities Log
Found 982 issues in 51 files
AggiungiCartaFrame.java (27 issues)
(29, 15) Remove this unused "bluLink" private field. 4 minutes ago
(16, 7) Remove this unused import 'javax.swing.JCheckBox'. 4 minutes ago
(3, 7) Remove this unused import 'java.awt.EventQueue'. 4 minutes ago
(251, 19) Remove the unnecessary boolean literal. 4 minutes ago
(244, 243) Remove the unnecessary boolean literal. 4 minutes ago
(239, 100) Remove the unnecessary boolean literal. 4 minutes ago
(234, 102) Remove the unnecessary boolean literal. 4 minutes ago
(240, 3) This branch's code block is the same as the block for the branch on line 235. [+1 location] 4 minutes ago
(227, 19) Remove the unnecessary boolean literal. 4 minutes ago
(218, 19) Remove the unnecessary boolean literal. 4 minutes ago
(209, 19) Remove the unnecessary boolean literal. 4 minutes ago
(211, 48) Remove the unnecessary boolean literal. 4 minutes ago
(201, 17) Refactor this method to reduce its Cognitive Complexity from 24 to the 15 allowed. [+20 locations] 4 minutes ag
(155, 40) Remove the unnecessary boolean literal. 4 minutes ago
(50, 34) Use static access with "javax.swing.WindowConstants" for "HIDE_ON_CLOSE". 4 minutes ago
(31, 29) Make "myGestore" transient or serializable. 4 minutes ago
(28, 16) Rename this field "Mano" to match the regular expression '^[a-z][a-zA-Z0-9]*$'. 4 minutes ago
(67, 36) Define a constant instead of duplicating this literal "PT Sans" 13 times. [+13 locations] 4 minutes ago
(205, 59) Define a constant instead of duplicating this literal "ATTENZIONE" 8 times. [+8 locations] 4 minutes ago
(0, 0) Move this file to a named package. 4 minutes ago
(211, 3) Merge this if statement with the enclosing one. [+1 location] 4 minutes ago
(220, 3) Merge this if statement with the enclosing one. [+1 location] 4 minutes ago
Analysis of all project files done 3 minutes ago
What's in this view
Rule Locations
String literals should not be duplicated
Adaptability issue | Not distinct
Maintainability
java:S1192
Learn more
Why is this an issue? How can I fix it?
Duplicated string literals make the process of refactoring complex and error-prone, as any change would need to be propagated on all occurrences.
Exceptions
To prevent generating some false-positives, literals having less than 5 characters are excluded.
Parameters
threshold 3
Parameter values can be set in Rule Settings. In connected mode, server side configuration overrides local settings.
Version Control TODO Problems Terminal Services Profiler SonarLint Build
Microsoft Defender configuration: The IDE has detected Microsoft Defender with Real-Time Protection enabled. It might severely degrade IDE performance. It is recommended to add the following paths to the Defender folder exclusion... (5 minut
57:21 LF UTF-8 Tab*
Maven
Endpoints
AI Assistant
Database
Notifications
Coverage
Bookmarks
Structure

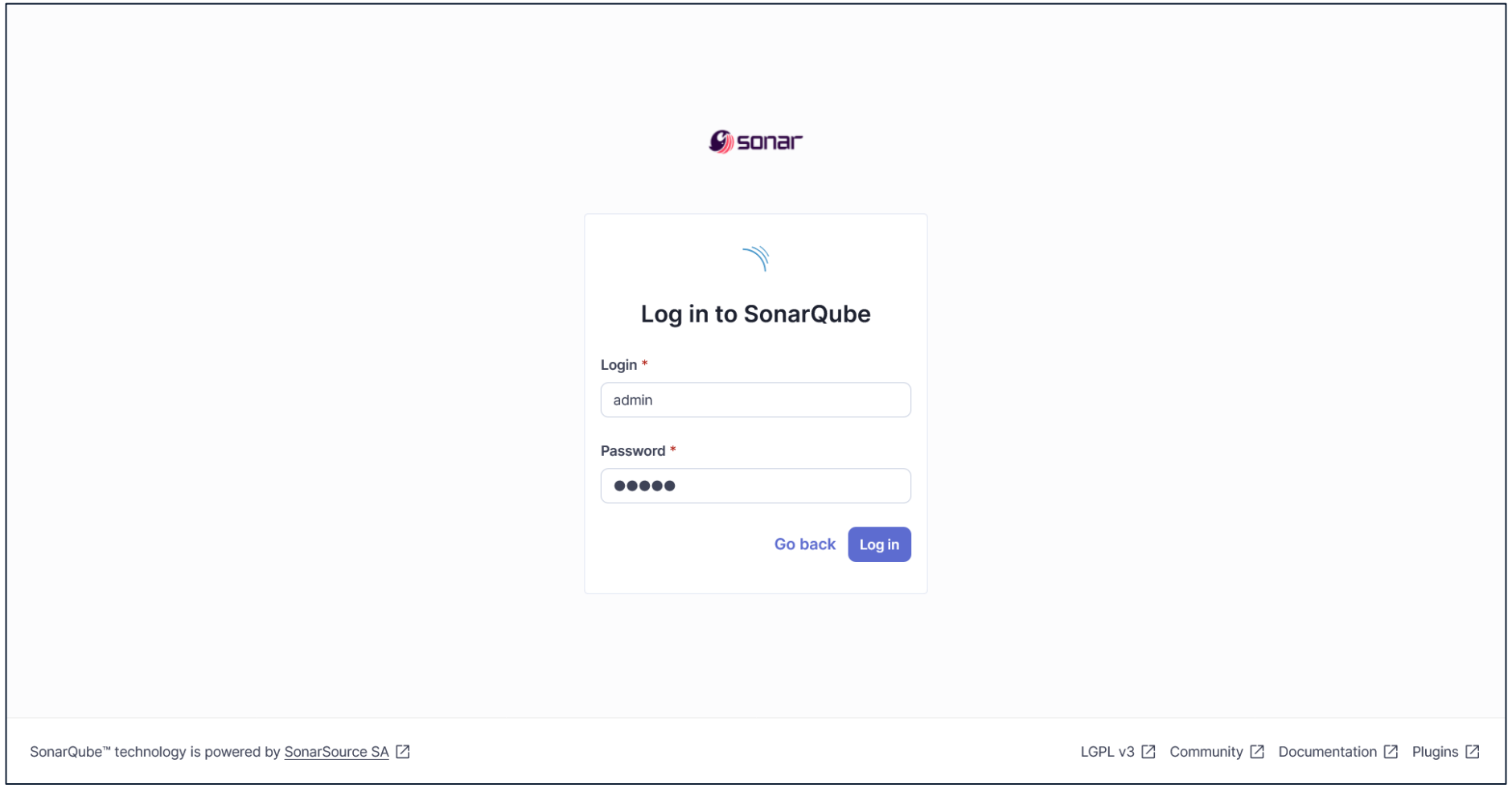
SonarQube Server

- The easiest way to set up a local instance of **SonarQube Server** is to use Docker and the official SonarQube images

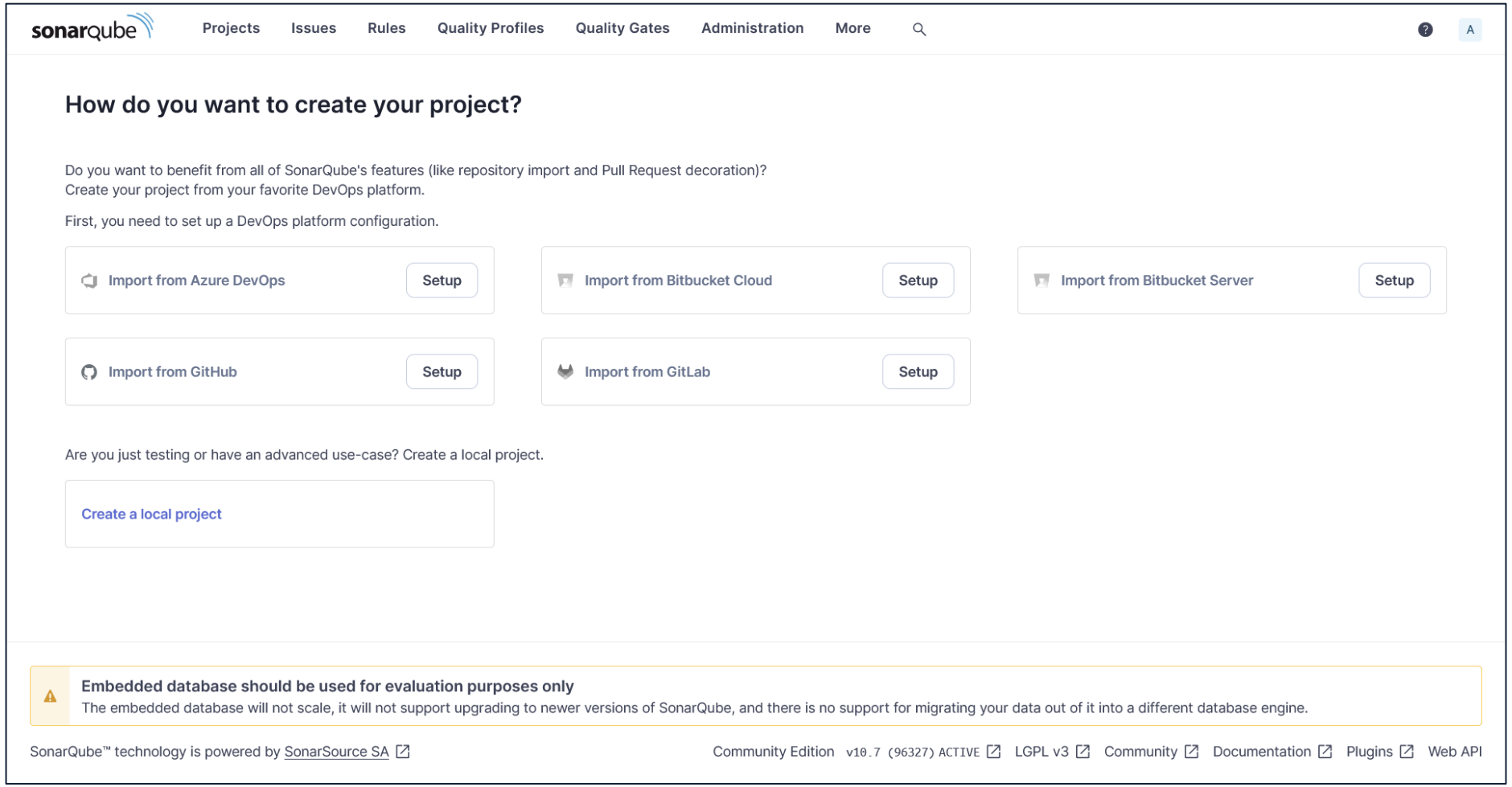
```
@luigi → D/O/T/S/sonar $ docker run -d --name sonarqube
-e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:10.7-community
```

- Once your instance is up and running, Log in to http://localhost:9000 using System Administrator credentials:
  - Username: `admin`
  - Password: `admin`

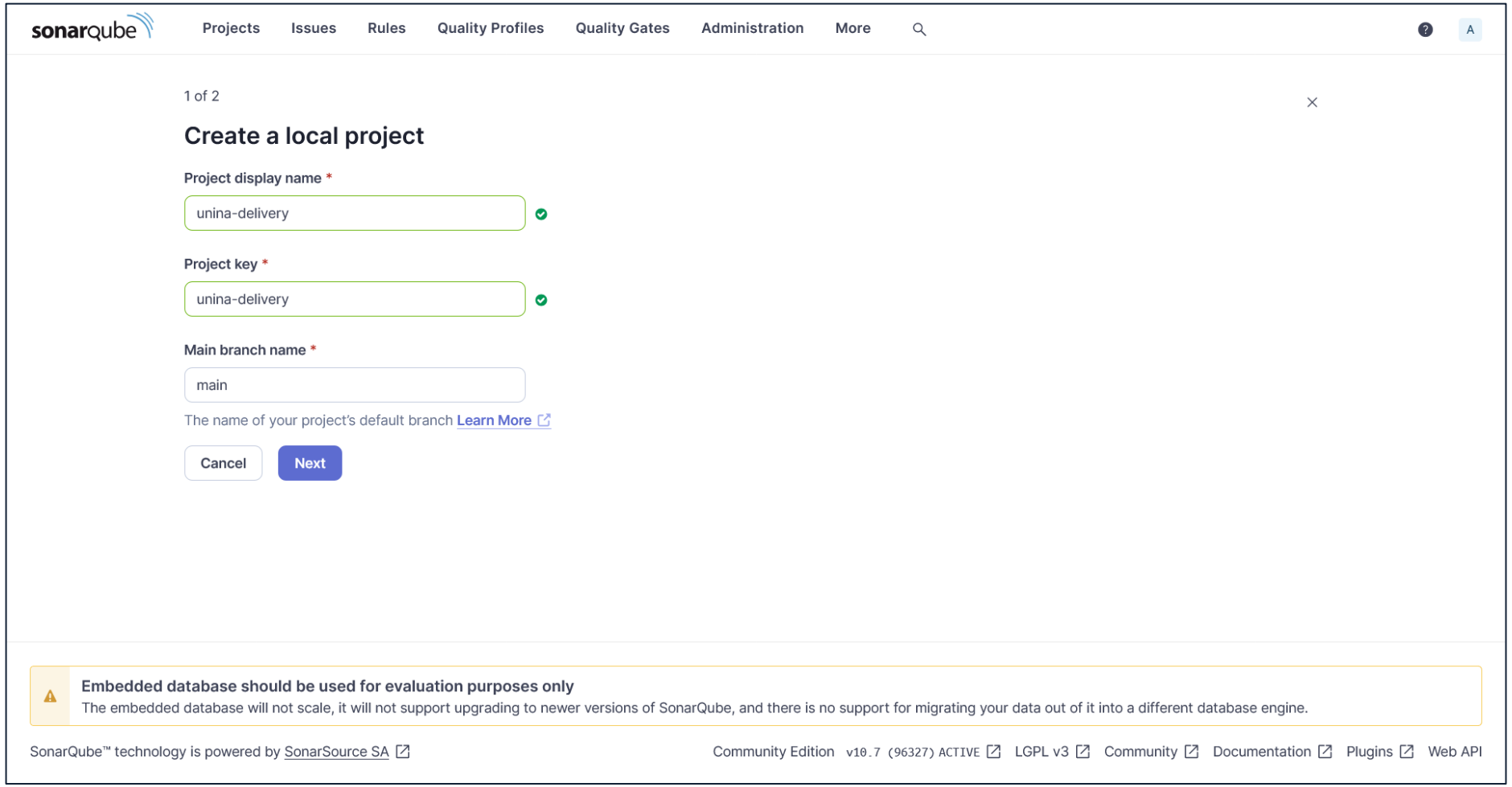
SonarQube Server: Login



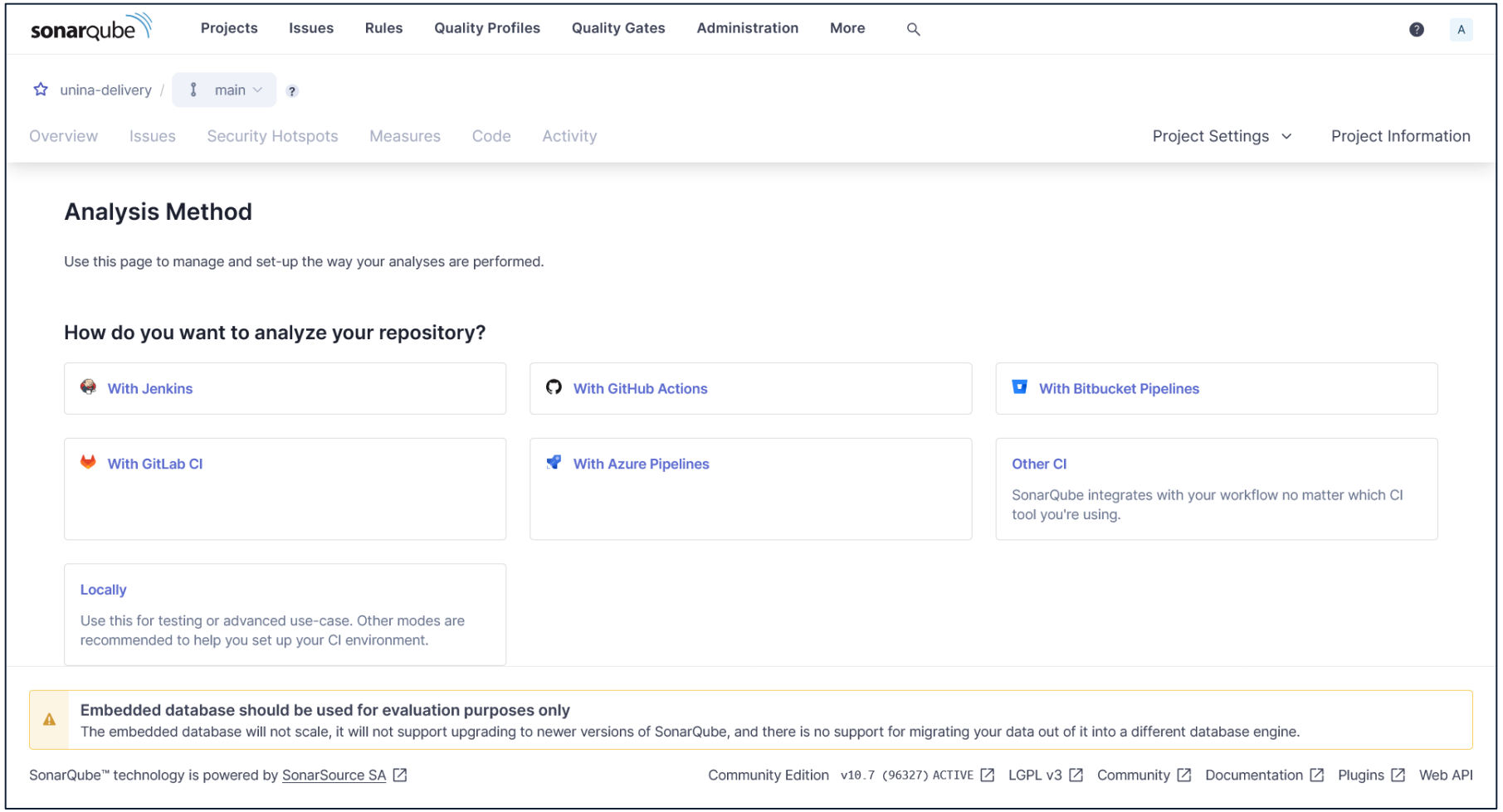
SonarQube Server: Create Project



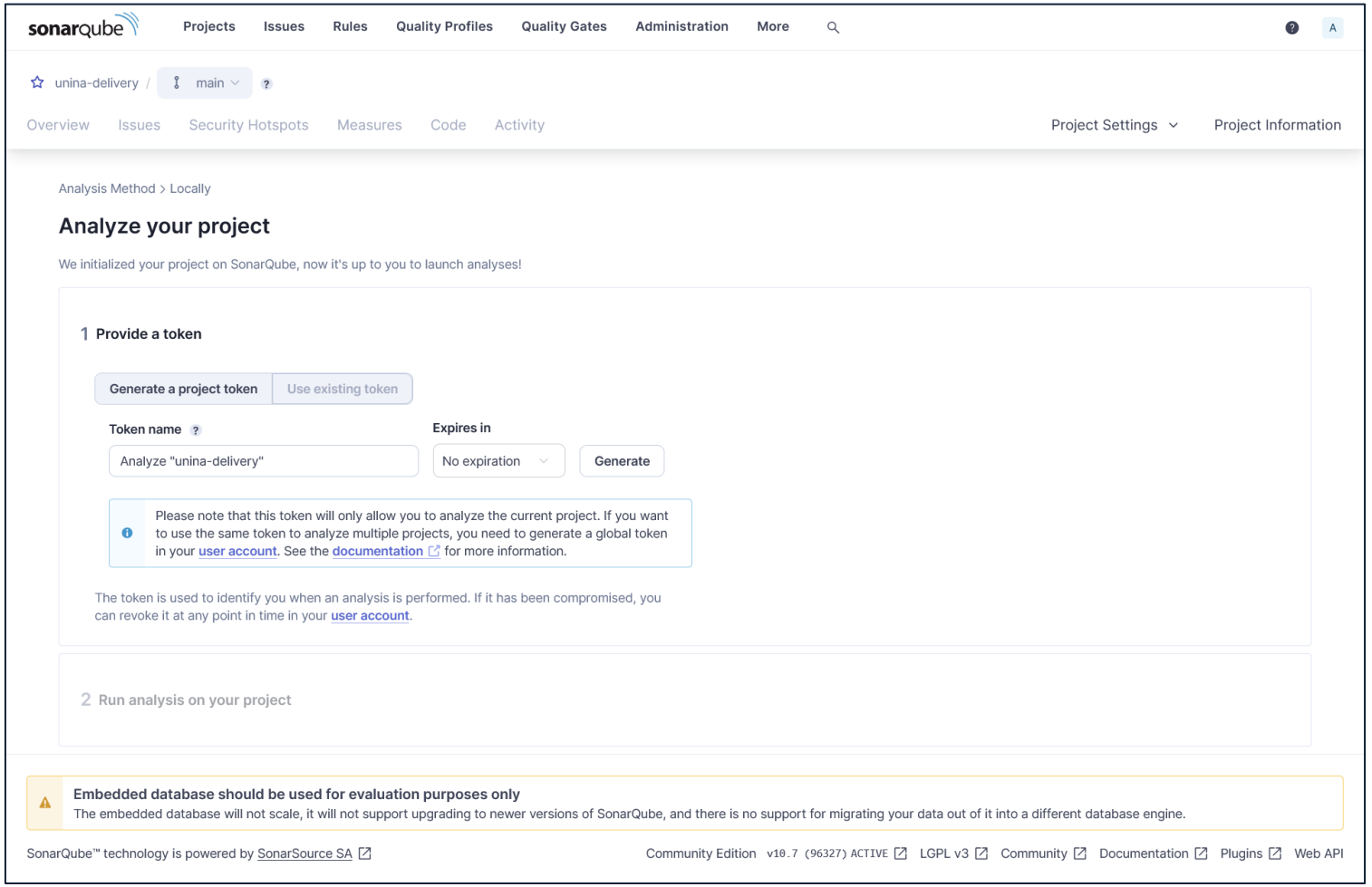
SonarQube Server: Create Project



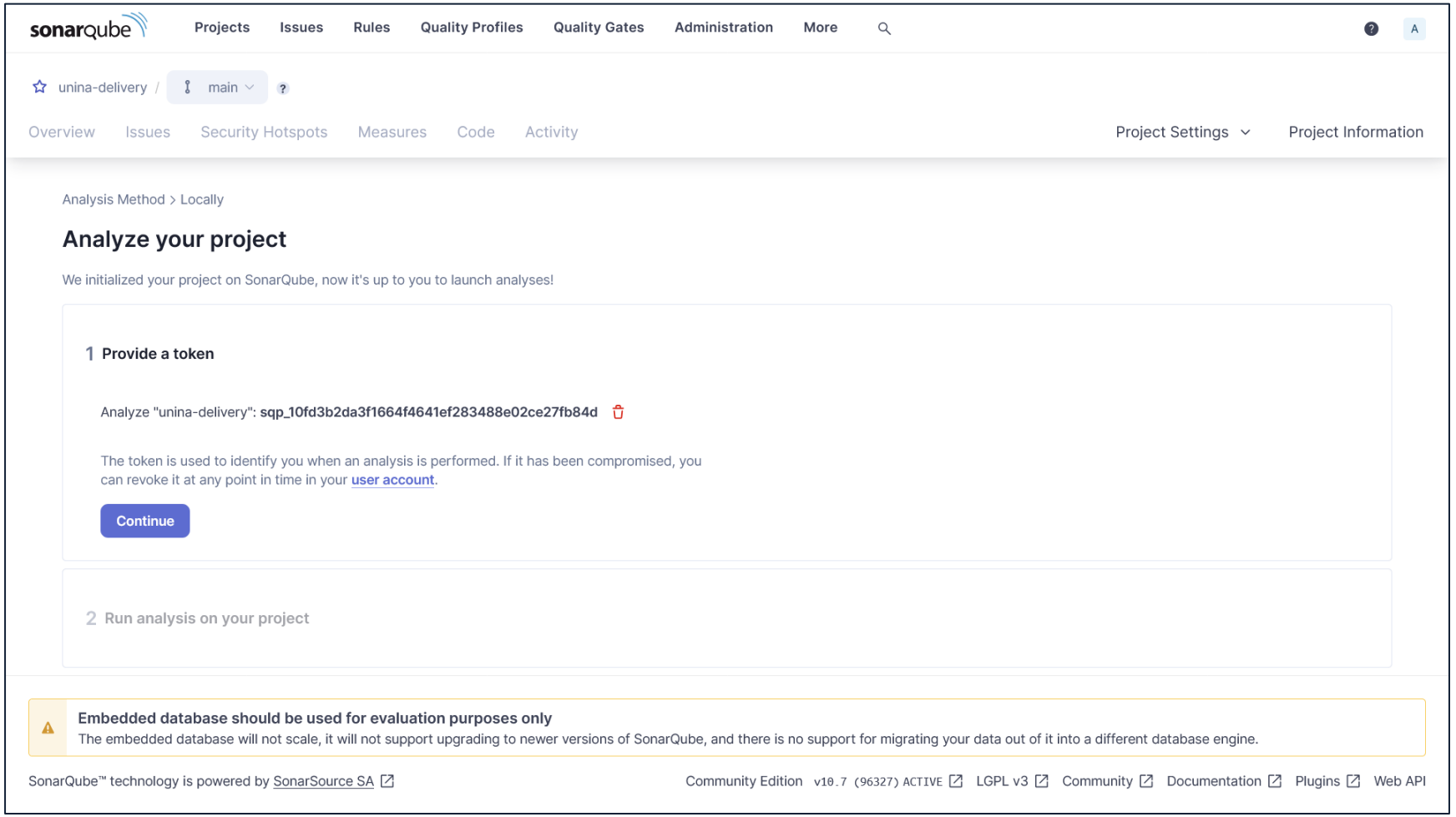
SonarQube Server: Analysis Methods



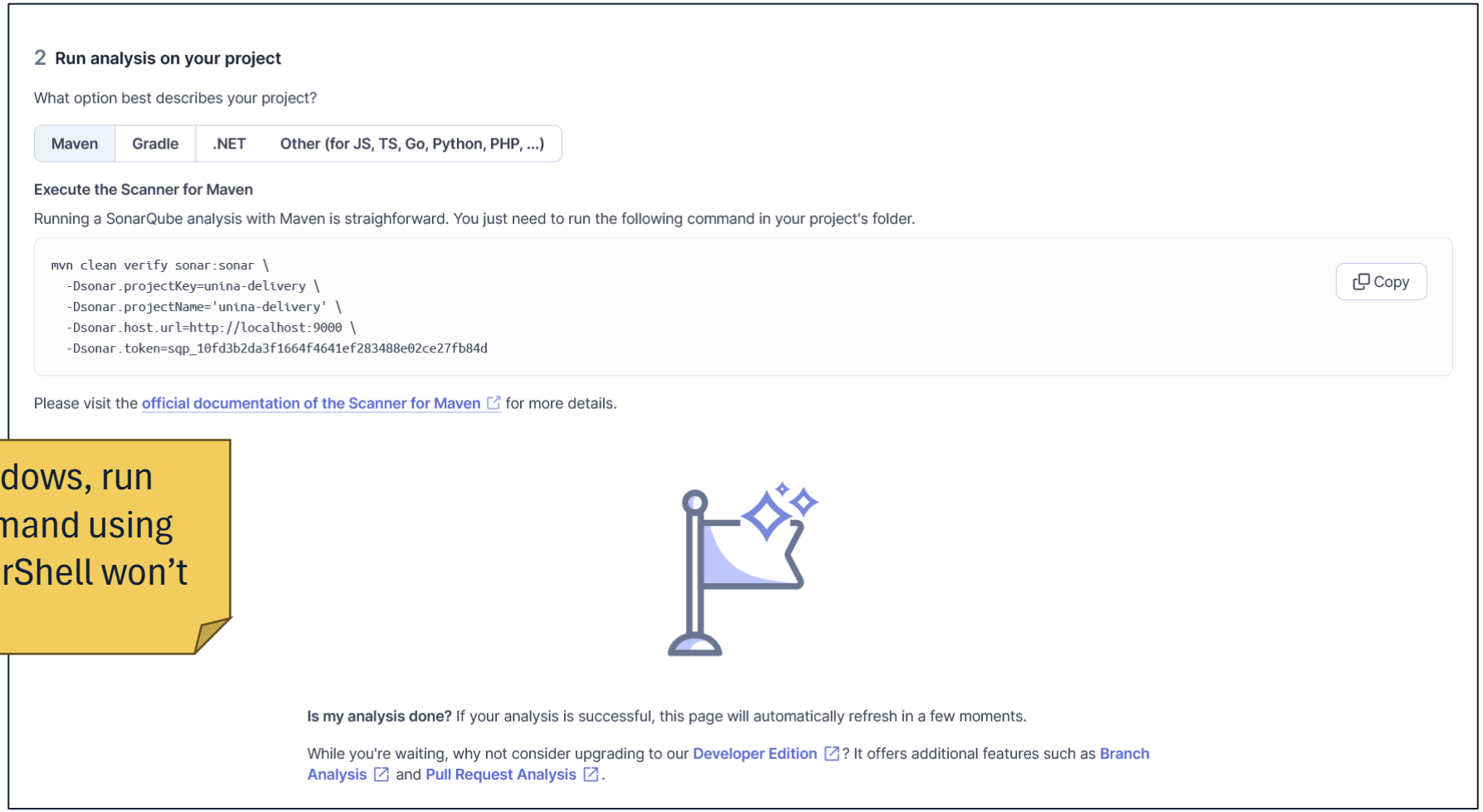
SonarQube Server: Local Analysis



SonarQube Server: Local Analysis



SonarQube Server: Running an analysis



If you're on Windows, run the above command using CMD.exe (PowerShell won't work)

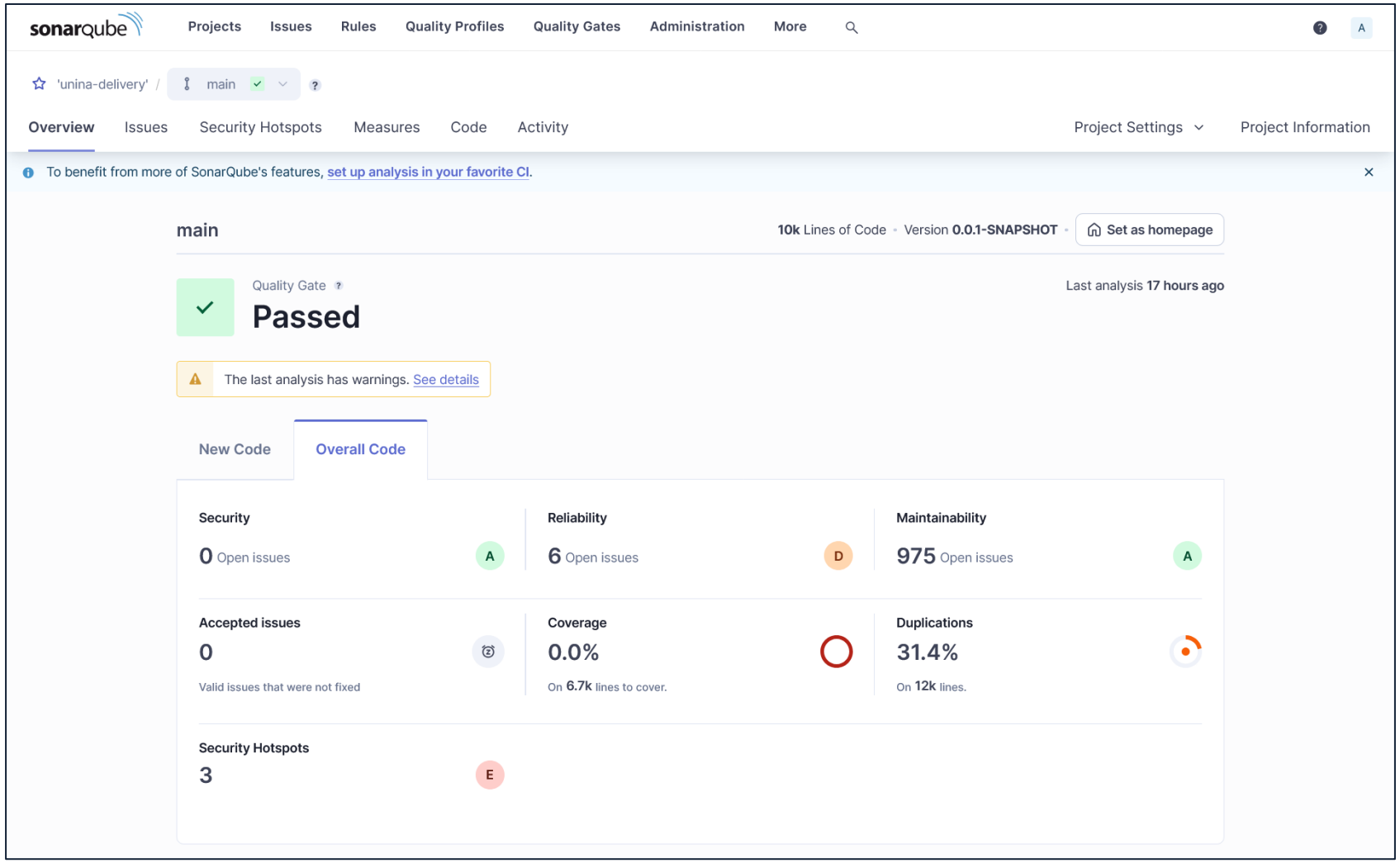
SonarQube Server: Running an Analysis

```
@luigi ➜ D/O/T/S/sonar $ docker run -d --name sonarqube
-e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:10.7-community

...

[INFO] 16:55:01.681 CPD Executor CPD calculation finished (done) | time=97ms
[INFO] 16:55:04.953 Analysis report generated in 229ms, dir size=1.3 MB
[INFO] 16:55:05.118 Analysis report compressed in 139ms, zip size=414.5 kB
[INFO] 16:55:05.169 Analysis report uploaded in 47ms
[INFO] 16:55:05.175 ANALYSIS SUCCESSFUL, you can find the results at:
       http://localhost:9000/dashboard?id=unina-delivery
[INFO] 16:55:05.238 Analysis total time: 23.735 s
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  30.596 s
[INFO] Finished at: 2024-11-16T16:55:05+01:00
[INFO] ------------------------------------------------------------------------
```

SonarQube Server: Project Dashboard



Maven Integration

- You don’t need to manually execute the above command everytime you need to run an analysis
- You can include the Sonar analysis in the Maven build pipeline
  - Add the Sonarsource Maven Plugin to your POM
  - Add the sonar goal within the pipeline (e.g.: in the verify phase)
  - Then you can perform Sonar analyses by just running «mvn verify»

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>
```

Modify Maven POM (2)

SonarQube Cloud

- SonarQube Cloud is SonarQube Server, offered as a SaaS in the cloud
- You don’t need to bother running a local instance of SonarQube
- It’s free for open-source projects
- If your project is in a public GitHub repository, you can effortlessy include it in SonarCloud
    1. Login into SonarCloud using your GitHub credentials (select free plan)
    2. Select the repository your project is in
    3. Enjoy a completely managed SonarQube instance in the cloud

A little digression on CI/CD

Delivering a Software Product

- When working on a software product, we want to produce high-quality software, and to deliver it **rapidly**
- **The job isn’t done as soon as we finish writing the source code**
- The project needs to be built (e.g.: produce a JAR)
- We need to execute tests
- We need to run automated code inspections
- We need to **deploy** the new version (e.g.: install it whenever it needs to be installed)

Integration Challenges

- When large teams of developers are working on a project, integrating changes from multiple contributors becomes a challenge
- Even if each change worked in isolation, we do not know if all changes work when merged toghether!
- Integrating multiple weeks/months of work can be a cumbersome task
- Integration requires time and work, and additional issues might emerge
- Integrating every small changes with a higher frequency is a better approach and allows faster integrations

Continuous Integration (CI)

Continuous Integration is a software development practice where each member of a team merges their changes into a codebase together with their colleagues changes **at least daily**.

Key points:

- Use a shared repository (with a main branch)
- Have an automated build process in place (e.g.: Maven)
- Have automated tests in place
- Every push to the shared repository should trigger a build
  - Commits that break the build are not merged. The dev that pushed a breaking change gets notified (and can fix it work before pushing again)

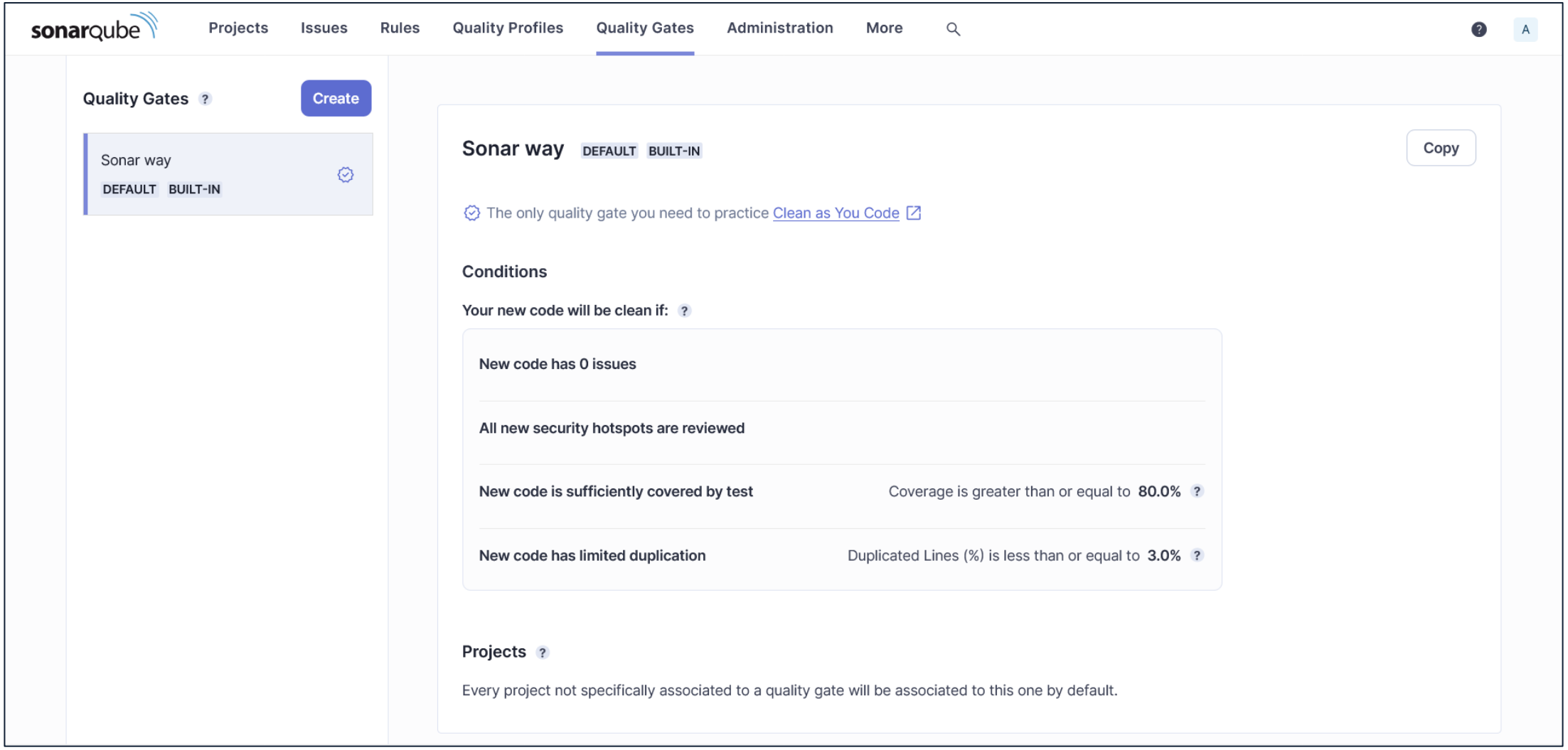
Continuous Delivery / Deployment (CD)

- The aim of Continuous Delivery is that the product should always be in a state where it is possible to release the latest build. This is essentially ensuring that the release to production is only a business decision.
  - To ensure this, testing should be conducted automatically on every build
- This might be take a step further, and include automatic deployment to production of the latest non-breaking build that passed all tests and quality checks

SonarQube: Quality Gates

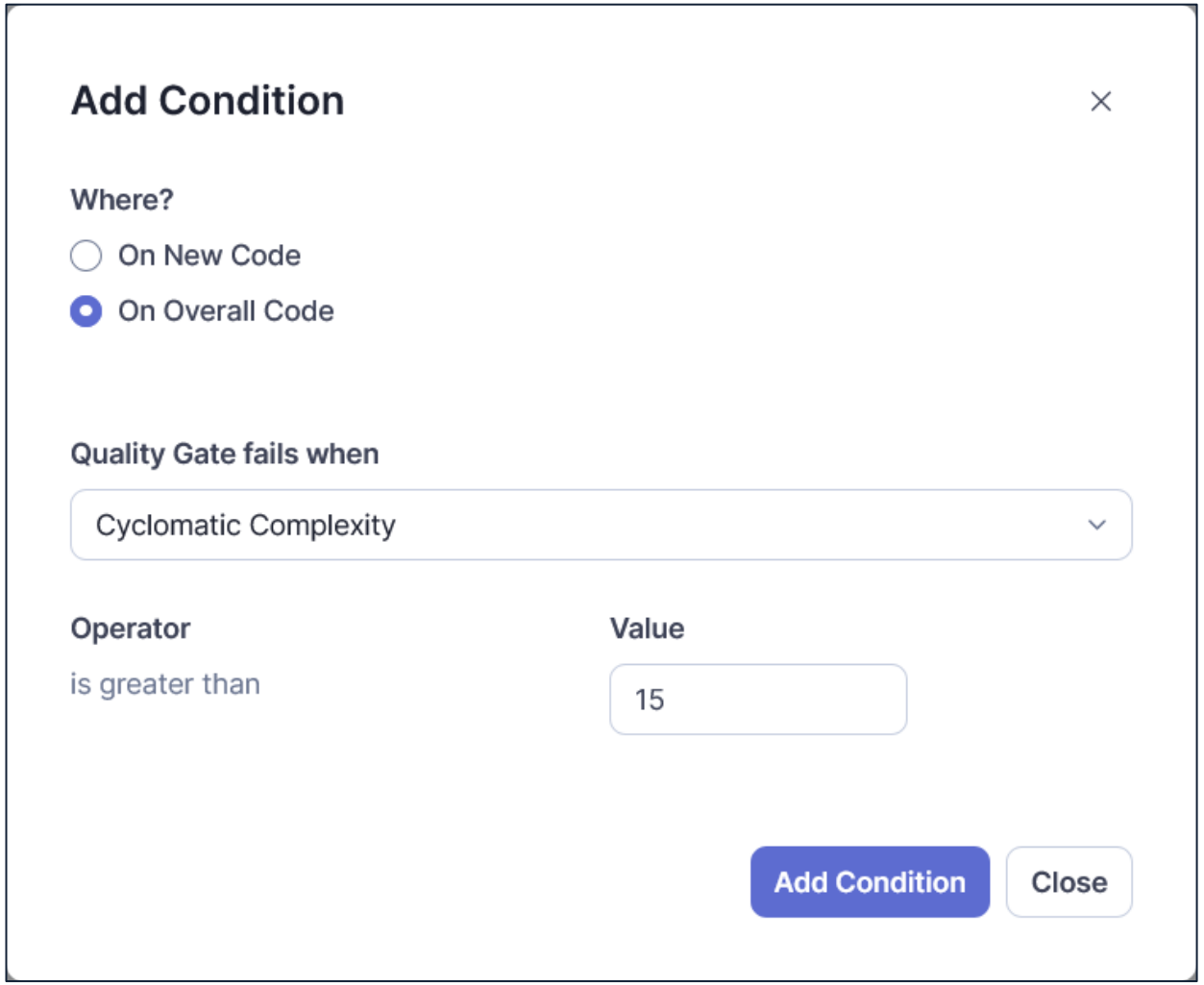
- In a CI/CD context, we may want to ensure that code adheres to established quality requirements
- With SonarQube, we can make a build fail based on quality requirements
- This is done with **Quality Gates**
  - A set of quality requirements that code (or new code) must satisfy
  - E.g.: no blocker or critical issues; at least 80% code coverage in tests, ...
  - A codebase either passes a quality gate or not

SonarQube: Quality Gates



SonarQube: Custom Quality Gates

- You can also define custom quality gates to enforce



SonarQube and CI/CD

- SonarQube can be integrated in CI/CD pipelines to ensure that code quality does not degrade
- E.g.: refuse merges which do not pass a quality gate

READINGS AND REFERENCES

- SonarSource website
  https://www.sonarsource.com/
- SonarQube docs
  https://docs.sonarsource.com/sonarqube/latest/
- Continuous Integration, Martin Fowler
  https://martinfowler.com/articles/continuousIntegration.html
- Automatic Assessment of Architectural Anti-patterns and Code Smells in Student Software Projects, L. L. L. Starace (among other authors)
  https://dl.acm.org/doi/pdf/10.1145/3661167.3661290