UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SOFTWARE ENGINEERING – LECTURE 35

Unit Testing in Isolation + 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

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SOFTWARE ENGINEERING – LECTURE 35 (PART 1)

Unit Testing in Isolation

Prof. Luigi Libero Lucio Starace

luigiliberolucio.starace@unina.it

https://luistar.github.io

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

Over-specification and Under-specification

Let’s try our hand with an assertion

Suppose you need to test the `getDivisors()` method:

```
class MathUtils {

    // Returns a list of divisors of number
    static List<Integer> getDivisors(int number){
        List<Integer> list = new ArrayList<Integer>();
        for(int i = 1; i <= number; i++) {
            if(number % i == 0) {
                list.add(i);
            }
        }
        return list;
    }
}
```

Let’s try our hand with an assertion (v. 1)

```
@Test
void testDivisorsOfEight() {
    List<Integer> divisors = MathUtils.getDivisors(8);
    List<Integer> expected = Arrays.asList(1,2,4,8);
    assertEquals(expected, divisors);
}
```

Quite brittle! We’re over-specifying

- The test depends on the implementation, not on the public interface
- We don’t really need the divisors to be in ascending order
- A small change to the implementation could break the test
  - E.g. adding 1 and `number` to the list before the loop, since both are sure to be divisors!

Let’s try our hand with an assertion (v. 1)

```
@Test
void testDivisorsOfEight() {
    List<Integer> divisors = MathUtils.getDivisors(8);
    List<Integer> expected = Arrays.asList(1,2,4,8);
    assertEquals(expected, divisors);
}
```

Error message might not be very informative!

- With two `List<Integer>` you should get something like:
  `expected: <[1,2,4,8]> but was: <[2,4,8]>`
  That’s not too bad, but what if there were 1000 elements?
- With two `List<Object>` you would have gotten something on the lines of `expected: <java.lang.ArrayList@6adbdf5> but was <java.lang.ArrayList@5ajrd98>`
  which is not very helpful!

Let’s try our hand with an assertion (v. 2)

```
@Test
void testDivisorsOfEight() {
    List<Integer> divisors = MathUtils.getDivisors(8);
    List<Integer> expected = Arrays.asList(1,2,4,8);
    assertTrue(divisors.containsAll(expected));
}
```

Too loose! We’re under-specifying

- The test would pass when `divisors` contains more numbers!

Let’s try our hand with an assertion (v. 3)

```
@Test
void testDivisorsOfEight() {
    List<Integer> divisors = MathUtils.getDivisors(8);
    List<Integer> expected = Arrays.asList(1,2,4,8);
    assertAll(
        () -> assertTrue(expected.containsAll(divisors)),
        () -> assertTrue(divisors.containsAll(expected))
    );
}
```

Solved the over/under-specification issues, but..

- Made it a little harder to read (think if we couldn’t use `containsAll()`)
- The error message we get is something like: `Multiple Failures (1 failure)`
  `expected: <true> but was: <false>`
  which is, again, not very helpful!

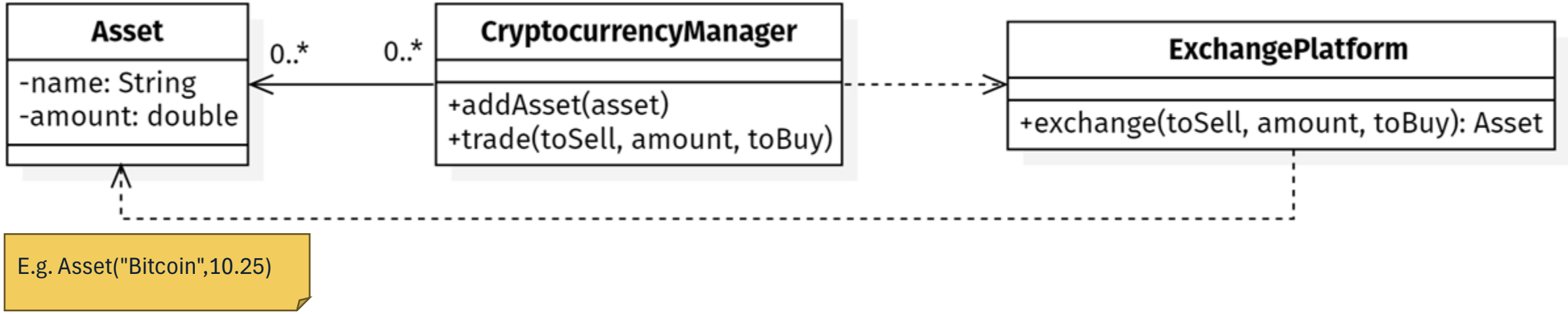
What's wrong with JUnit Assertions?

- Can lead to hard-to-read tests
- Can lead to over/under-specification
  - Resist the temptation of `assertEquals()` when it's not what we *really* need
  - Do not settle with loose specifications for the sake of convenience
- Error messages not always immediate to understand
  - There exist ways to achieve better error messages in standard JUnit assertions (e.g. the `Supplier<String>` parameter), but why should we re-invent the wheel?
- **Assertion frameworks** can help with those pain points!
  - You can learn more about them at the end of this slide deck

Testing in Isolation

Dependencies and Testing

A class may depend on other helper classes (e.g. DAOs, APIs)



Suppose we need to test the `trade()` method.

Dependencies and Testing

```java
@Test
void testTrading() {
    // arrange
    CryptocurrencyManager c = new CryptocurrencyManager();
    c.addAsset(new Asset("BTC",10.0));
    c.addAsset(new Asset("EUR",10000.0));
    // act (1 BTC = 7000.0 EUR)
    c.trade("EUR", 7000.0, "BTC");
    // assert
    assertTrue(c.getAsset("EUR").get().getAmount() == 3000.0);
    assertTrue(c.getAsset("BTC").get().getAmount() == 11.0);
}
```

We're using the real ExchangeAPI, trading **real** Euros!

The exchange rate might change tomorrow!

See anything wrong with this?

Dependencies and Testing

Using real production objects as helpers in our unit tests has a few drawbacks and may not always be possible:

- When a test fails, we don't know if the bug is in the unit we're testing or in its dependencies;
- The introduction of a bug in a highly-used helper object can cause a ripple of failing tests all across the system;
- Sometimes, we cannot afford to use real production objects! Think of a `BankTransactionsDAO`, or a `MissileLauncher`!

How can we alleviate the above issues and still test our units?

Test Doubles

Test Doubles are replacements for production objects that are typically used in tests.

Test Doubles can be classified as follows:

- **Fakes:** objects that have working implementations, but are not suitable for production because of limitations;
- **Stubs:** replace a real component and return pre-defined answers;
- **Mocks:** more advanced, configurable test stubs that can also record the indirect outputs of the unit under test.

(Lack of) Inversion of Control

Suppose our CryptocurrencyManager was like this:

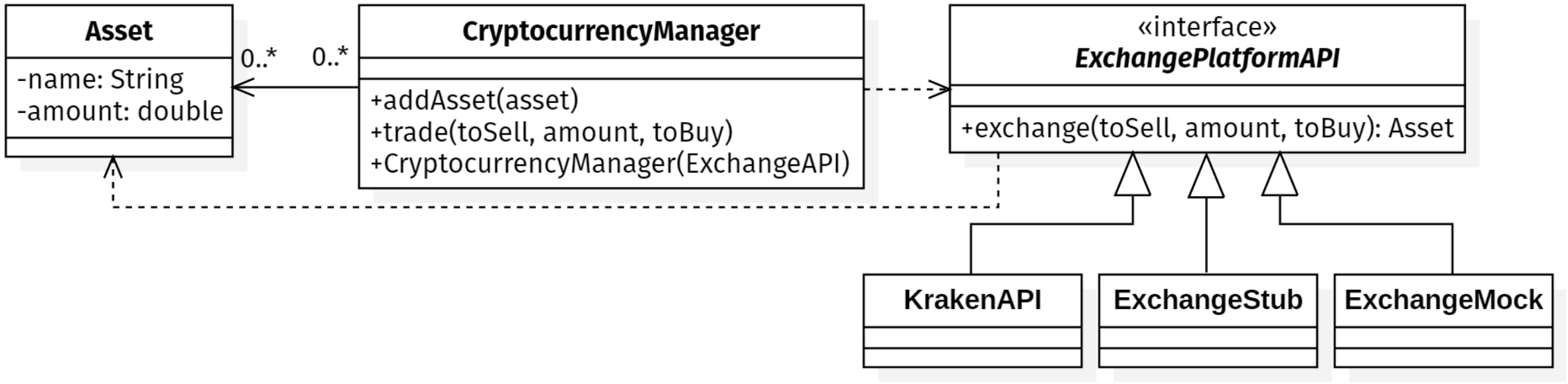
```
public class CryptocurrencyManager {
  private ExchangePlatform exchangeAPI = new ExchangePlatform();
  CryptocurrencyManager(){/*...*/}
  public void trade(String toSell, double amountToSell, String toBuy) {/*...*/}
}
```

That's really BAD design!

- What if we want to support more than one exchange platform?
- There's no way we can use a test double in place of the real deal!

Inversion of Control

- Client classes should be able to provide dependencies
  - Through constructors, public setters, builders, method parameters...
- Inversion of control is fundamental to make units testable



Dependencies and Testing

```
@Test
void testTrading() {
    // arrange
    CryptocurrencyManager c = new CryptocurrencyManager(new ExchangeStub());
    c.addAsset(new Asset("BTC",10.0));
    c.addAsset(new Asset("EUR",10000.0));
    // act (1 BTC = 7000.0 EUR)
    c.trade("EUR", 7000.0, "BTC");
    // assert
    assertTrue(c.getAsset("EUR").get().getAmount() == 3000.0);
    assertTrue(c.getAsset("BTC").get().getAmount() == 11.0);
}
```

We're not trading **real** Euros anymore!

The exchange rate won't change if the stub doesn't!

Our very simple Exchange Platform Stub

```
class ExchangeStub implements ExchangeAPI {
    @Override
    public Asset exchange(String toSell, double amount, String toBuy) {
        return new Asset("BTC",1.0);
    }
}
```

Don’t be fooled by this tiny example! Writing mocks and stubs can be very tedious!

Think of multiple methods, with many conditions each!

- E.g.: when amount < 1000, then return X, when >2000 …
- And you might not be able to re-use it in the next test!

Mocking frameworks

Writing stubs and mocks is boring and takes time.

Mocking frameworks allow us to easily create mocks and stubs to use in our tests.

Popular frameworks include: Mockito, EasyMock, JMockit.

Introducing Mockito

Mockito is a well-known mocking library.

Top 10 Java library across all libraries[1] .

[1] https://blog.overops.com/githubs-10000-most-popular-java-projects-here-are-the-top-libraries-they-use/

Mockito – Creating Mocks

Mocks can be created with the static method

`<T> T mock(Class<T> classToMock)`

By default, each method of the mock will return the default value for the corresponding type.

```
List mockedList = mock(ArrayList.class); // with classes
Map mockedMap   = mock(Map.class);       // and with interfaces too

mockedMap.get("Foo"); // returns null
mockedList.get(2);    // returns null
mockedList.isEmpty(); // returns false
mockedList.size();    // returns 0
```

Mockito – Configuring Mocks

We can configure the behaviour of our mock as follows:

```
List mockedList = mock(ArrayList.class); // with classes
Map mockedMap   = mock(Map.class);       // and with interfaces too

// configuration
when(mockedMap.get("Foo")).thenReturn("Bar");
doReturn("Bar").when(mockedList).get(2); // alternative way
when(mockedList.size()).thenReturn(99);

mockedMap.get("Foo"); // returns "Bar"
mockedList.get(2);    // returns "Bar"
mockedList.isEmpty(); // returns false as before
mockedList.size();    // returns 99
```

Mockito – Argument matchers

Often, we won't need to be so specific about input arguments.

In fact, being more generic could help us write less configuration code and more flexible tests.

```
// configuration: order matters!
when(mockedList.get(anyInt())).thenReturn("Mockito!");
when(mockedList.get(lt(0))).thenThrow(IllegalArgumentException.class);
when(mockedList.get(-25)).thenReturn("MOCKITO!");

mockedList.get(1988);  // returns "Mockito!"
mockedList.get(-25);   // returns "MOCKITO!"
mockedList.get(-2);    // throws IllegalArgumentException
```

Mockito – Configuring Mocks

When we invoke a method on mock with a given list of parameters, Mockito tries to find the latest configuration rule that «matches», and applies that.

If no configuration rule is found, it defaults to default values.

Therefore, configuration order matters, and more general rules should be declared before the more specific ones.

Mockito – Argument matchers

Mockito features a lot of built-in argument matchers, that can often also be combined in a Hamcrest-like fashion.

E.g.: `any()`, `anyString()`, `anyCollection()`, `leq()`, `isNull()`, …

We won't examine them in detail today, but if you're interested you can check out the ArgumentMatchers and the AdditionalMatchers classes from the docs.

It's also possible to use Hamcrest matchers as Mockito argument matchers with the MockitoHamcrest class!

Mockito – Arg. matchers with Hamcrest

```
import static org.mockito.Mockito.*;
import static org.mockito.hamcrest.MockitoHamcrest.argThat;
import static org.hamcrest.Matchers.*;
import static it.unina.computerscience.softeng.junitdemo.examples.IsPalindrome.*;

// configuration - argThat is a static method from the MockitoHamcrest class
when(mockedList.add(argThat(
        is(palindrome()) // Hamcrest Matcher
))).thenReturn(true);

mockedList.add("Abe");  // returns false
mockedList.add("Bob");  // returns true
mockedList.add("Otto"); // returns true
```

Mockito – Configuring Answers

When mocking with Mockito, you're not limited to pre-determined return values.

- With the generic interface Answer, you can customize the behaviour of your mocks by specifying an action to be executed to compute the desired output.

Mockito – Configuring Answers

```
List<Integer> myMock = mock(ArrayList.class);

when(myMock.get(anyInt())).thenAnswer(
  new Answer() {
    public Object answer(InvocationOnMock invocation) {
      return ((Integer)invocation.getArgument(0)) + 1;
    }
  });

//same as the above
when(myMock.get(anyInt())).thenAnswer(
  invocation -> ((Integer)invocation.getArgument(0)) + 1
);

myMock.get(42); // returns 43
myMock.get(99); // returns 100
```

Mockito – Verifying behaviour

Usually, in the assert phase of our tests, we focus on checking that the final state of the Class Under Test is the one we expect.

This is testing by side-effects, and often it is not enough!

```
// assert
assertTrue(c.getAsset("EUR").get().getAmount() == 3000.0);
assertTrue(c.getAsset("BTC").get().getAmount() == 11.0);
// hopefully, a call was made to ExchangeAPI.exchange() with the right params!
```

Mockito – Verifying behaviour

After you're done with your mock, you can further inspect it to check if and how it's been used!

Out-of-the-box, with no configuration required, a Mockito mock keeps track of every method call it receives.

You can verify a mock's past behaviour with the `verify()` method and its variants.

Mockito – Verifying behaviour: examples

```
// check that myMock.size() was called exactly once
verify(myMock).size();
// check that myMock.get(19) was called exactly once
verify(myMock).get(19);
// check that myMock.get was called exactly three times with any int arg.
verify(myMock, times(3)).get(anyInt()); // with Mockito matcher
// check that myMock.get was called no more than three times with any even arg.
verify(myMock, atMost(3)).get(argThat(is(even()))); // with Hamcrest matcher
// check that myMock.get(0) was called at least two time
verify(myMock, atLeast(2)).get(0);
// check that no interaction occurred with mockedStack
verifyNoInteractions(mockedStack);
// check that myMock.addAll was never called
verify(myMock, times(0)).addAll(anyCollection());
```

Mockito – Verifying behaviour: examples

```
// check that method calls happened in a precise order
InOrder inOrder = inOrder(myMock);
inOrder.verify(myMock).get(19);
inOrder.verify(myMock).get(42);
inOrder.verify(myMock).get(99);

// check that no interaction happened after myMock.get(99)
verifyNoMoreInteractions(myMock);
```

Mockito – Verifying behaviour: errors

Here's what a `verify()` error message looks like:

```
> Error message:

org.mockito.exceptions.verification.NoInteractionsWanted:
No interactions wanted here:
-> at CryptocurrencyManagerTest.java:153
But found this interaction on mock 'arrayList':
-> CryptocurrencyManagerTest.java:138
```

Getting back at our example

```
@Test
void testTrading() {
    // create and configure mock
    ExchangePlatformAPI api = mock(ExchangePlatformAPI.class);
    when(api.exchange("EUR", 7000.0, "BTC")).thenReturn(new Asset("BTC",1.0));
    // arrange
    CryptocurrencyManager c = new CryptocurrencyManager(api);
    c.addAsset(new Asset("BTC",10.0)); c.addAsset(new Asset("EUR",10000.0));
    // act
    c.trade("EUR", 7000.0, "BTC");
    //assert
    assertTrue(c.getAsset("EUR").get().getAmount() == 3000.0);
    assertTrue(c.getAsset("BTC").get().getAmount() == 11.0);
    verify(api).exchange("EUR",7000.0,"BTC"); // API were actually called
}
```

Mockito and JUnit 5

There’s an official mockito-junit-jupiter extension for JUnit 5.

```
@ExtendWith(MockitoExtension.class) // register the extension
class CryptocurrencyManagerTest {

    @Mock ExchangePlatformAPI api;  // parameter initialization

    @Test
    void testTrading() {/*...*/}

    @Test
    void testWithMockParam(@Mock List<Integer> list) {/*...*/} //param. resolution

}
```

Mockito – Limitations

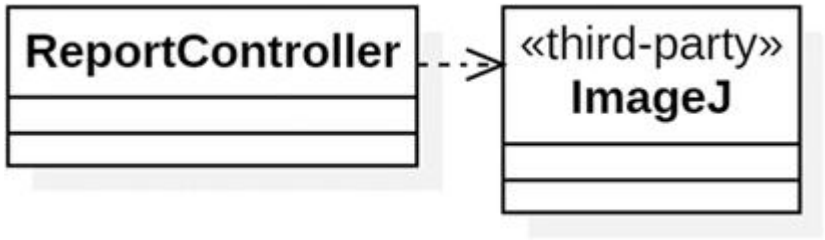
With Mockito you cannot mock:

- constructors;
- static methods;
- equals(), hashCode();

If you need to, you can either consider refactoring, or checkout more "invasive" tools like PowerMock, which uses custom classloaders and bytecode manipulation.

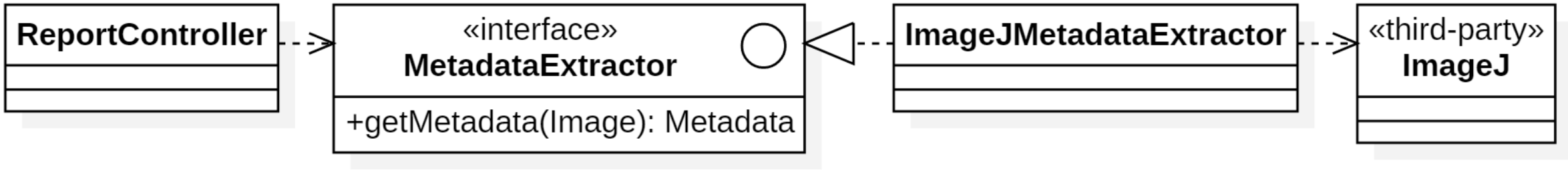
Rules to mock by

1. Mock behaviour, not data!
   - Just instantiate your POJOs, use builders if you need to.
2. Do not mock types you don’t own (e.g. third-party libraries)
   - It might cover up bugs!
   - You should put up an adapter layer between your units and 3rd-party libraries, and mock the adapter



Rules to mock by

1. Mock behaviour, not data!
   - Just instantiate your POJOs, use builders if you need to.
2. Do not mock types you don't own (e.g. third-party libraries)
   - It might cover up bugs!
   - You should put up an adapter layer between your units and 3rd-party libraries, and mock the adapter



Readings and Referencesl

- https://martinfowler.com/bliki/UnitTest.html
- https://martinfowler.com/bliki/TestDouble.html
- http://xunitpatterns.com/Test%20Double.html
- https://martinfowler.com/articles/mocksArentStubs.html
- https://junit.org/junit5/docs/current/user-guide/
- https://site.mockito.org/

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SOFTWARE ENGINEERING – LECTURE 35 (PART 2)

Continuous Code Inspection with SonarQube

Prof. Luigi Libero Lucio Starace

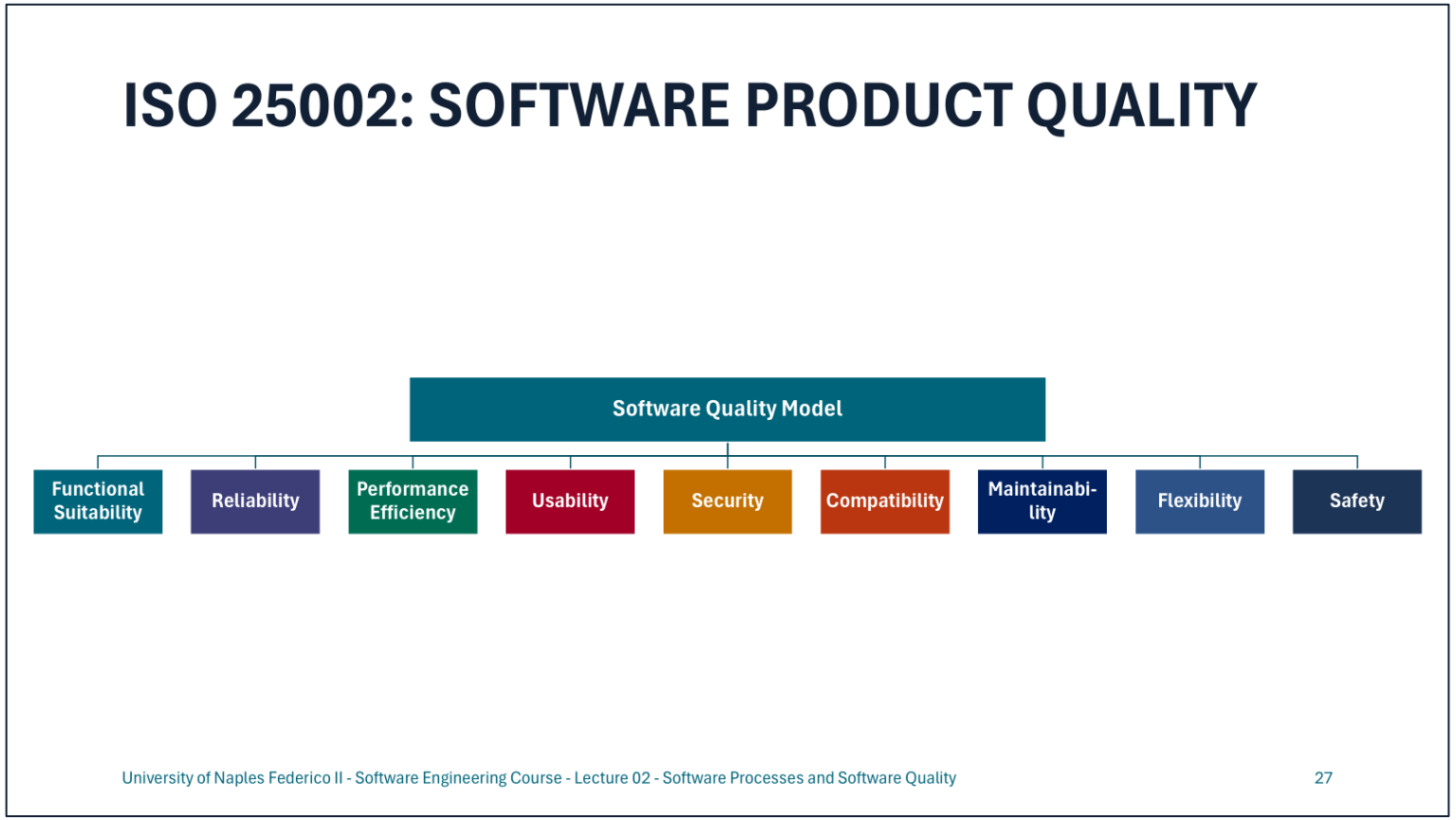
luigiliberolucio.starace@unina.it

https://luistar.github.io

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

(A look back on) Software Quality

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

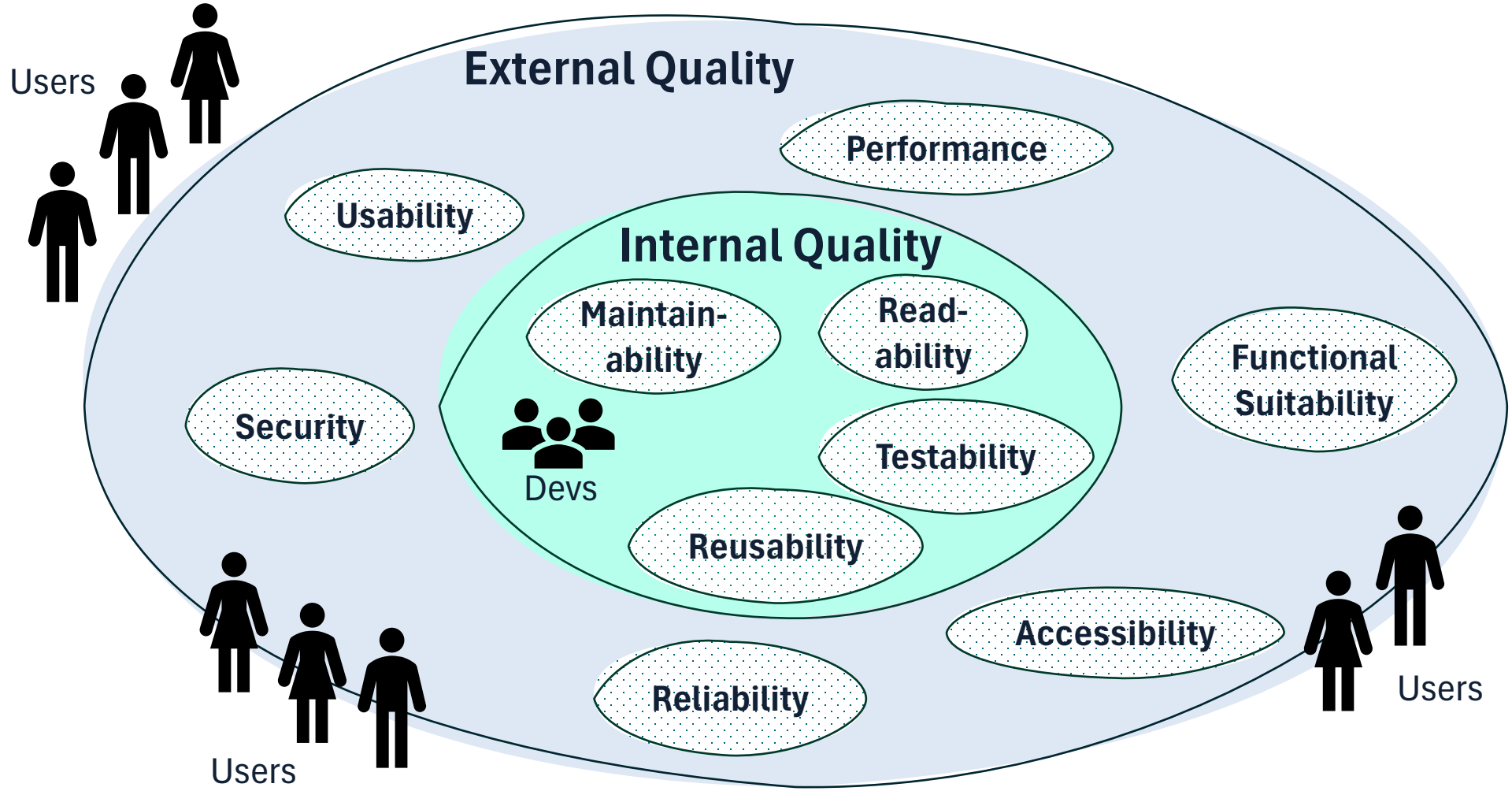


Internal and External Software Quality

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

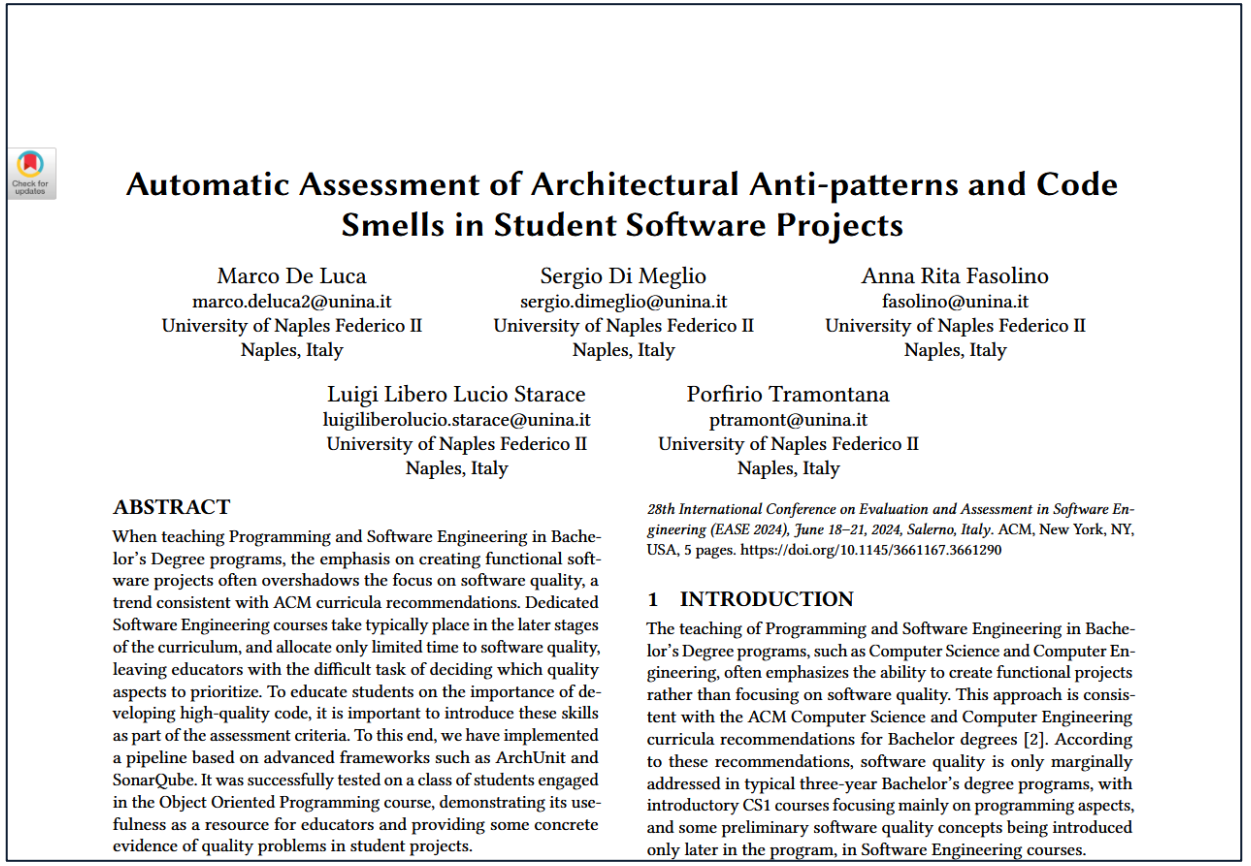
66 responses submitted

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

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

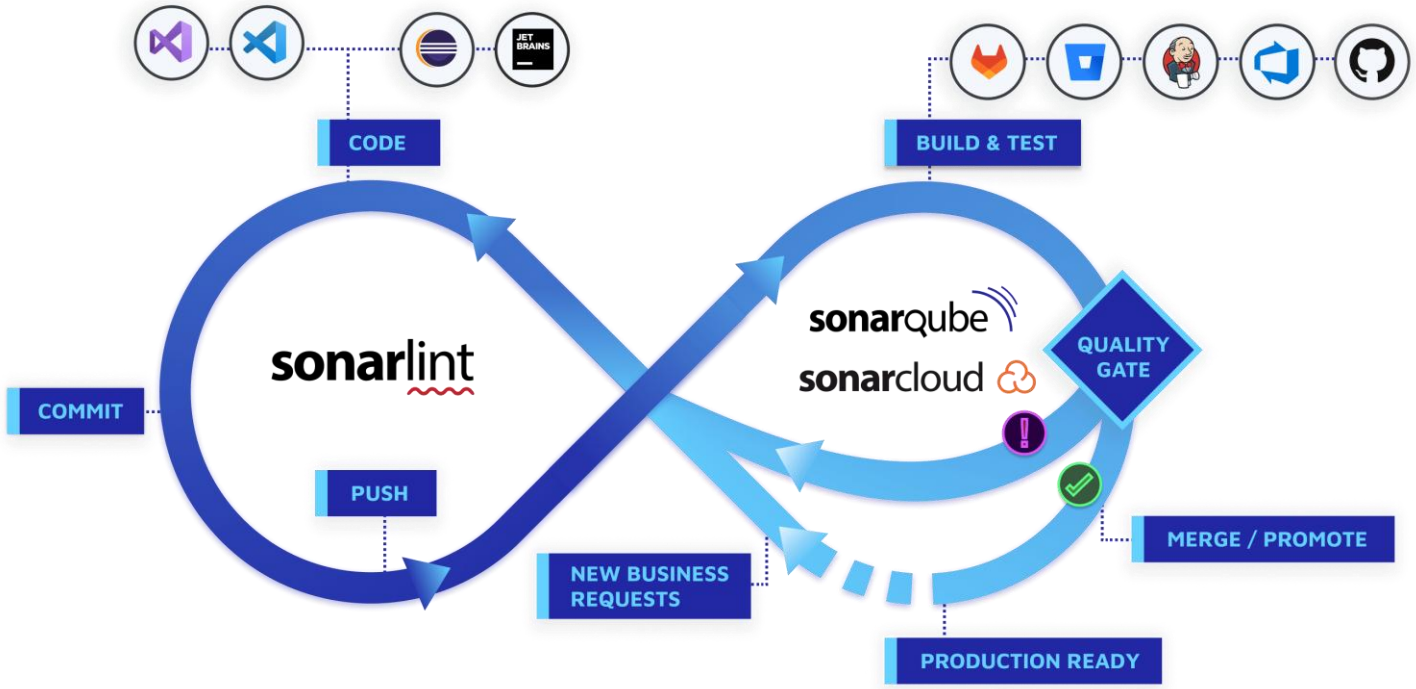
The teaching of Programming and Software Engineering in Bachelor's Degree programs, such as Computer Science and Computer Engineering, often emphasizes the ability to create functional projects rather than focusing on software quality. This approach is consistent with the ACM Computer Science and Computer 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

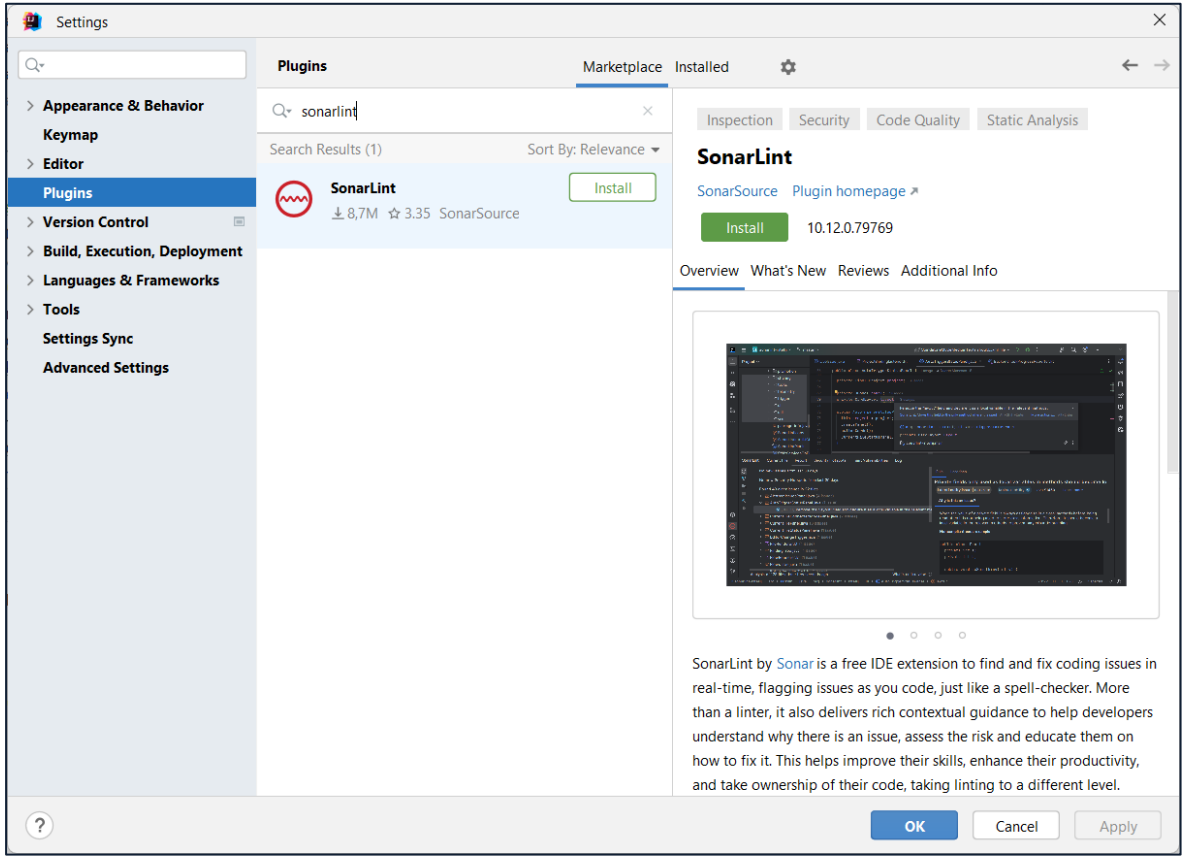
Two (not mutually exclusive) approaches are possible:

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

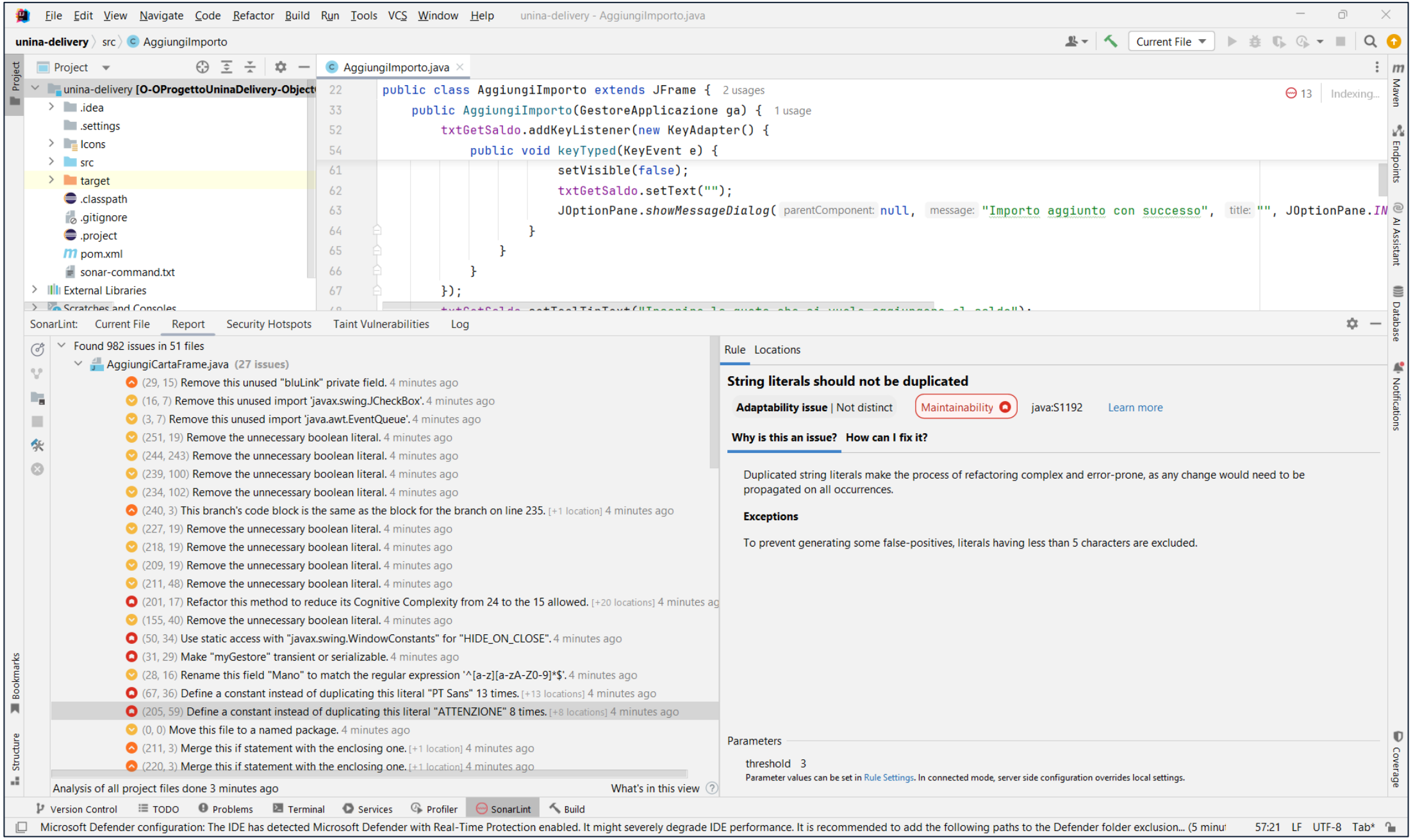


SonarQube IDE (previously SonarLint)

- **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

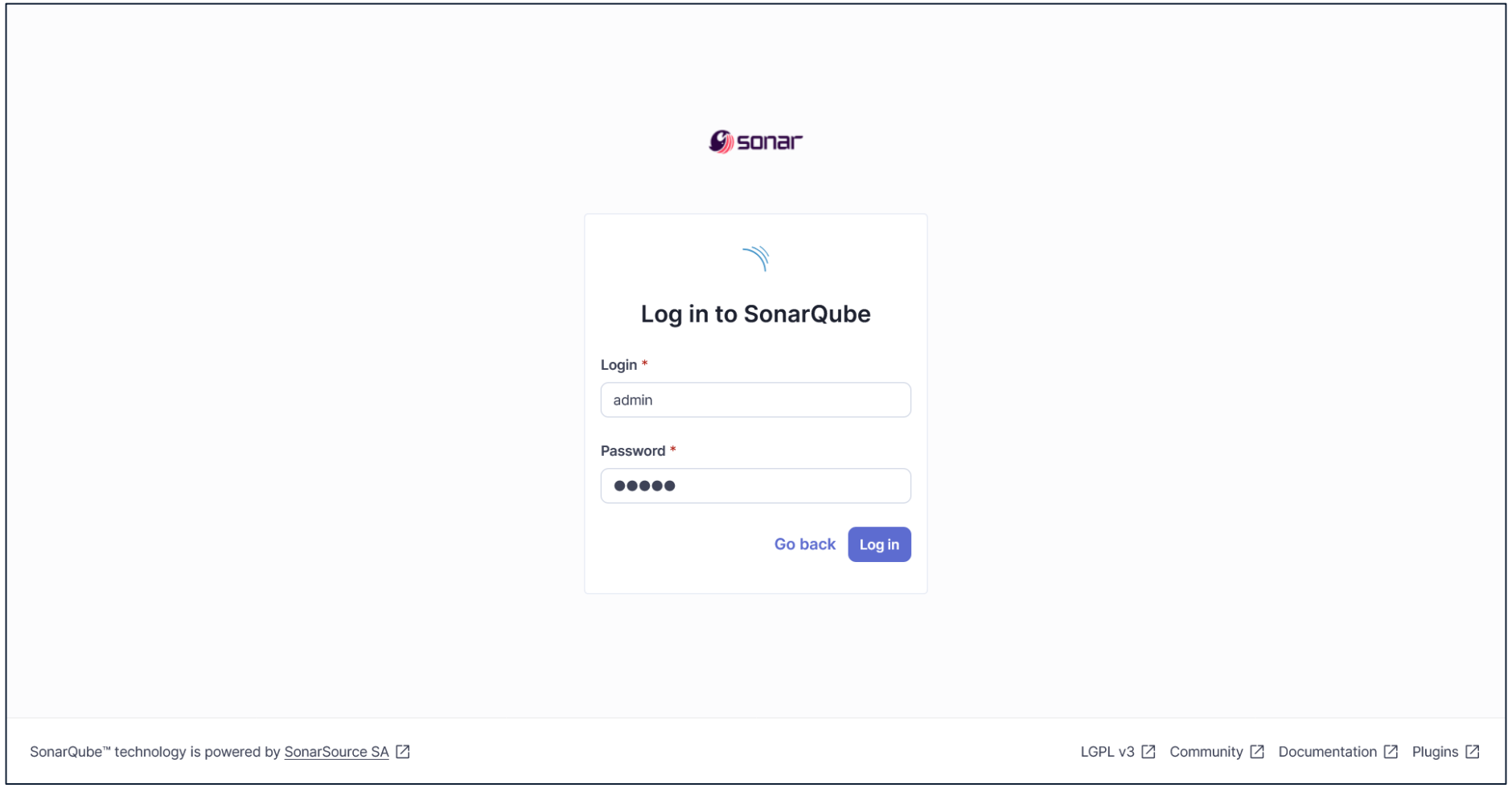
SonarQube Server

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

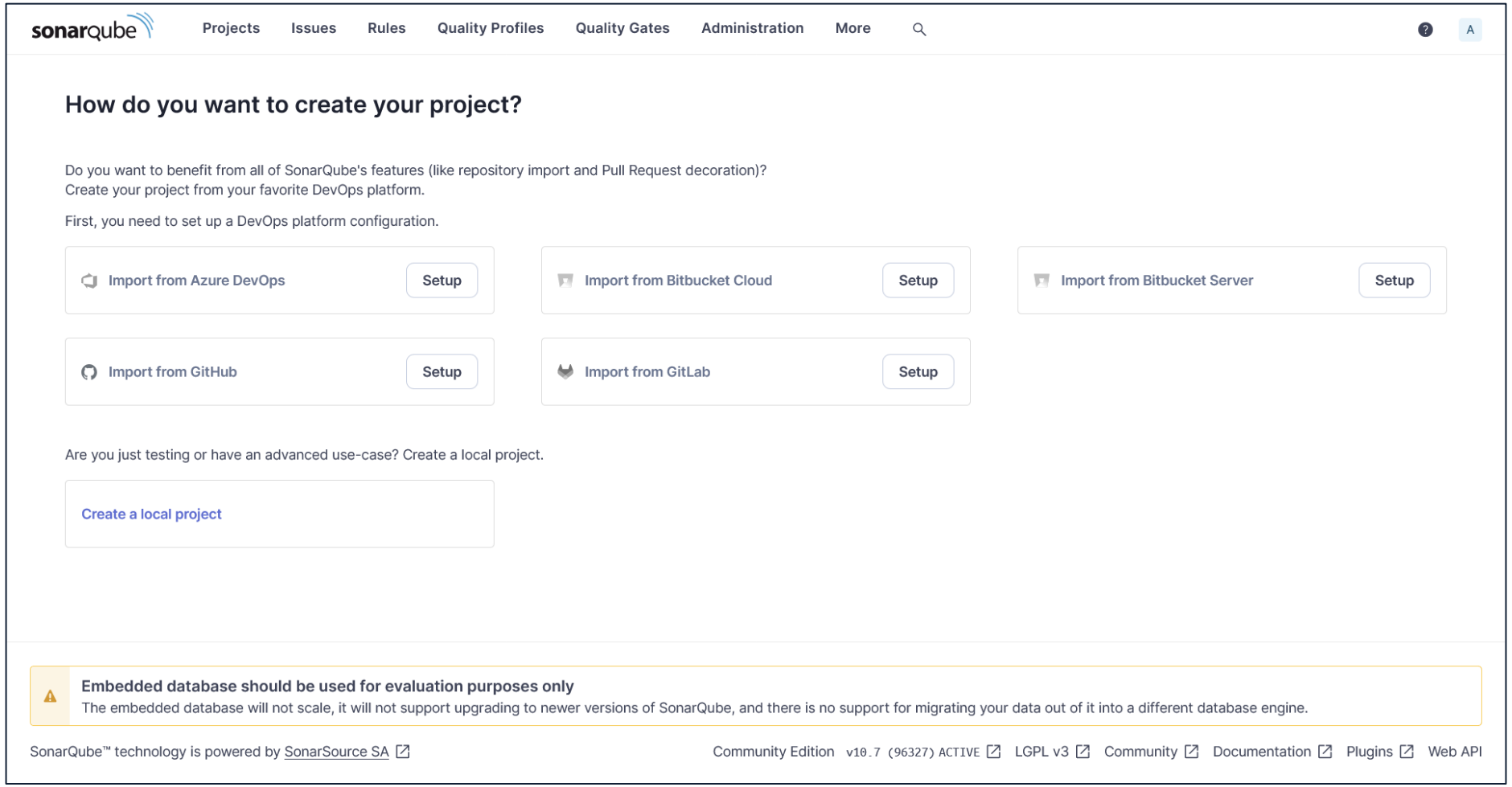
```
@luigi ➜ D/O/T/S/sonar $ docker run -d --name sonarqube
-e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:10.7-community
```

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

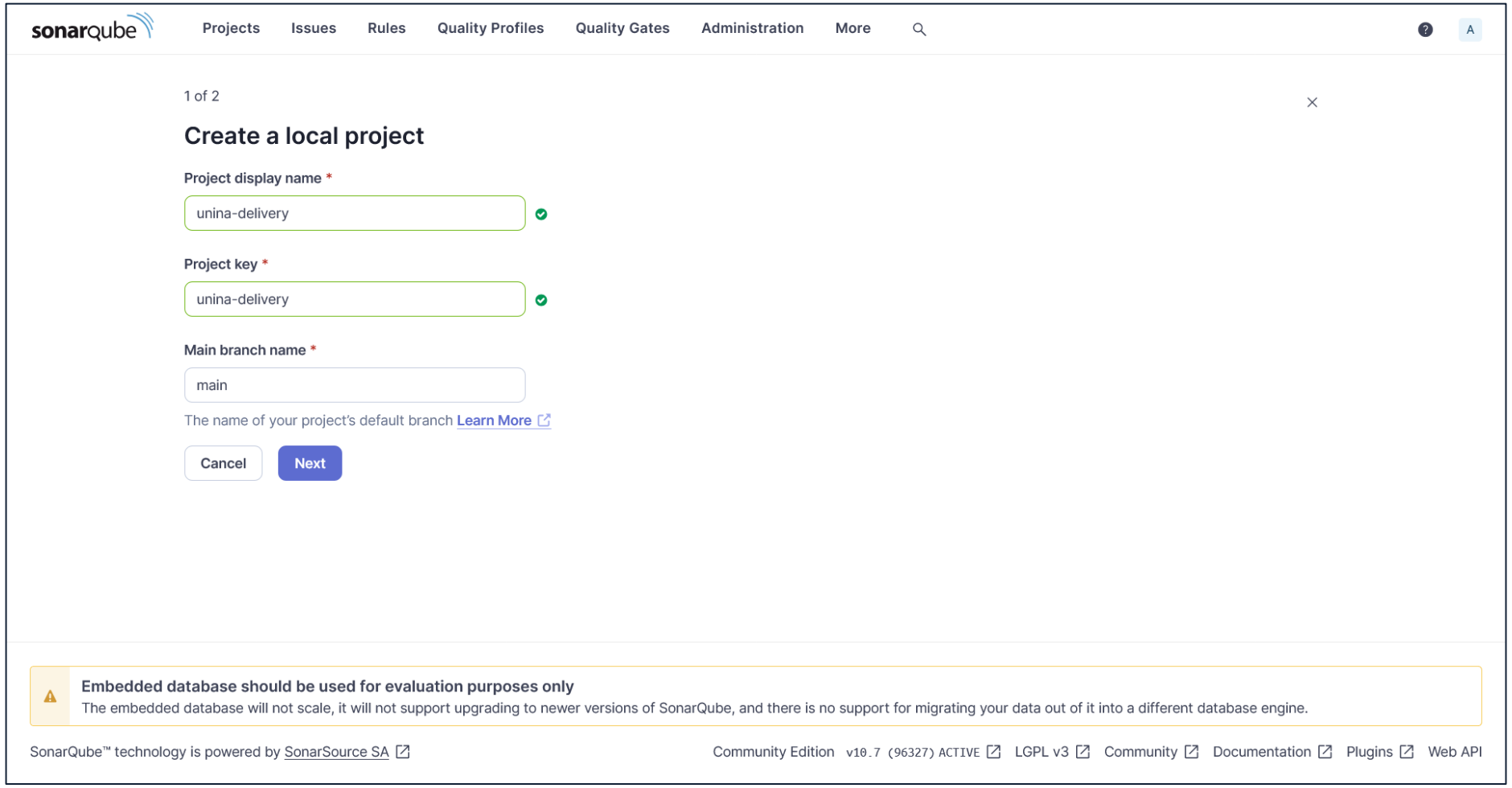
SonarQube Server: Login



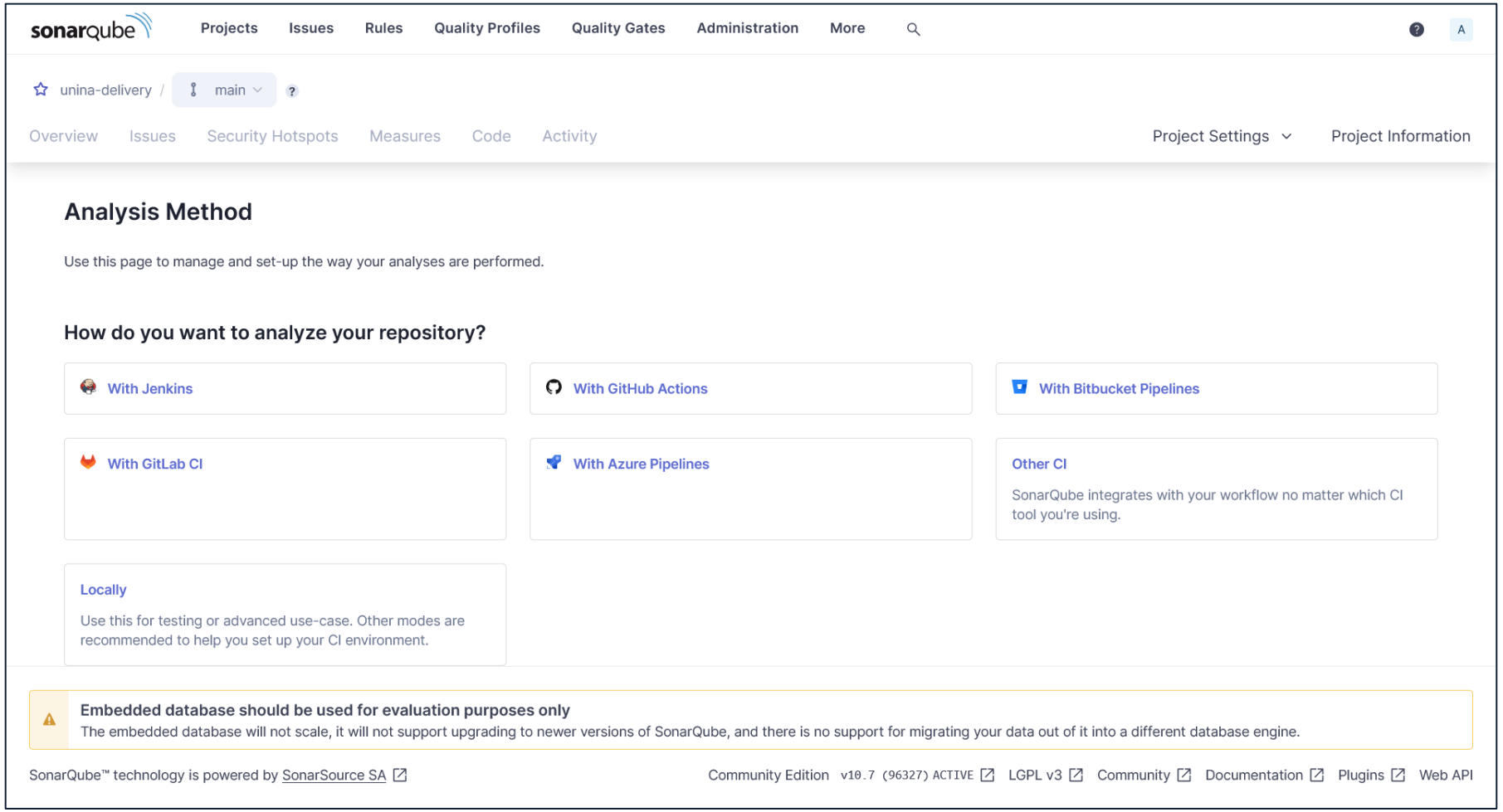
SonarQube Server: Create Project



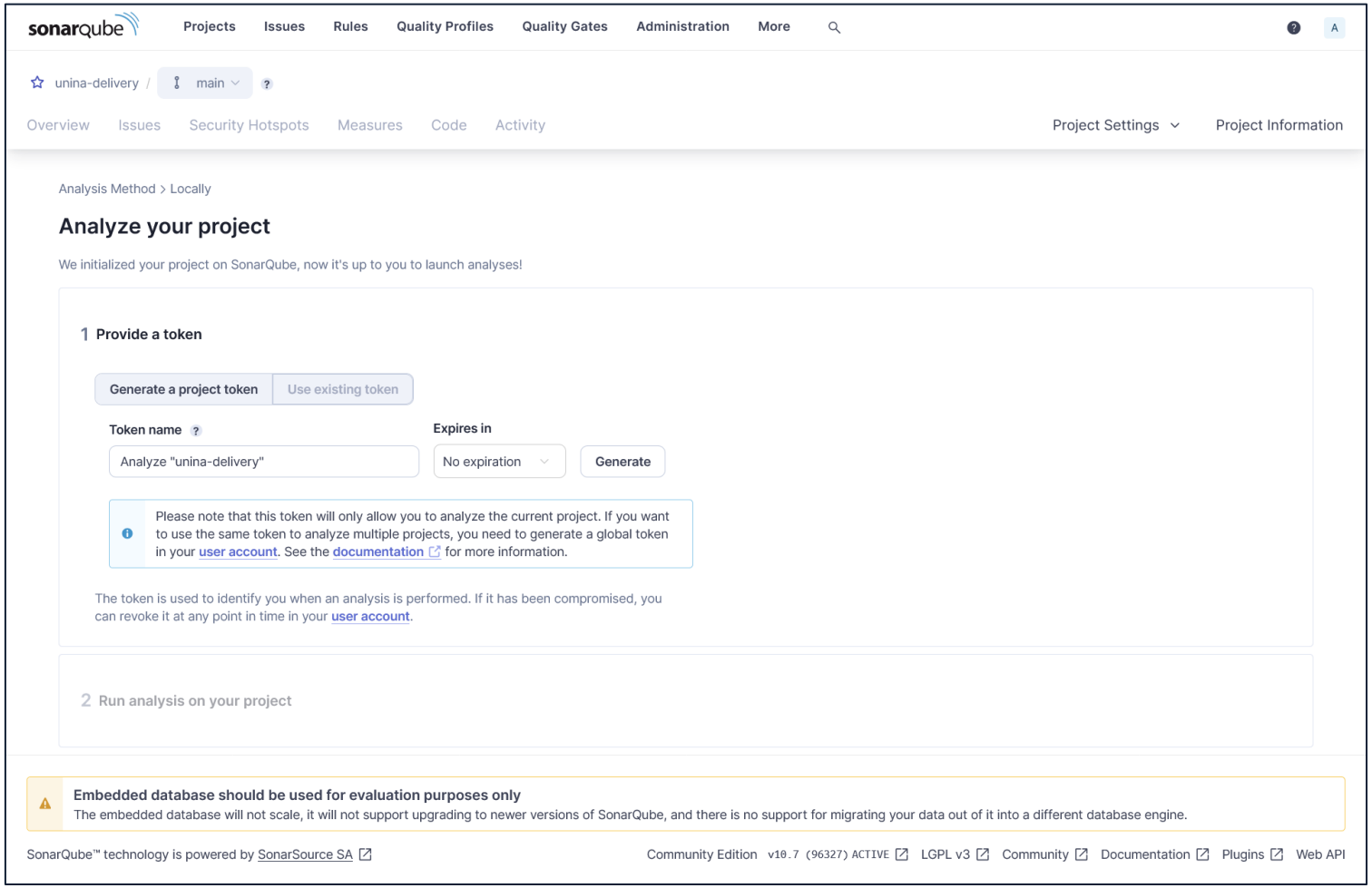
SonarQube Server: Create Project



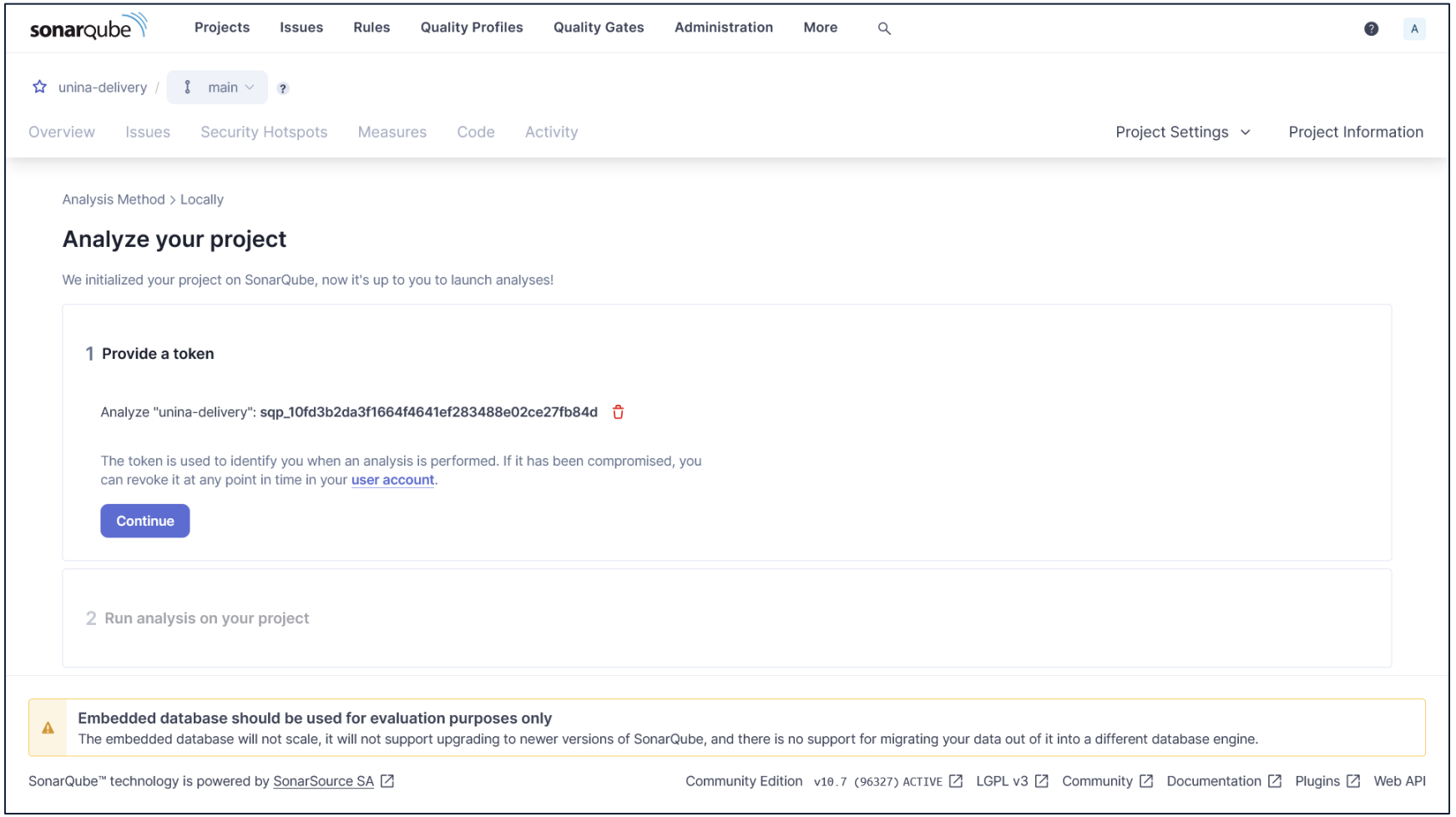
SonarQube Server: Analysis Methods



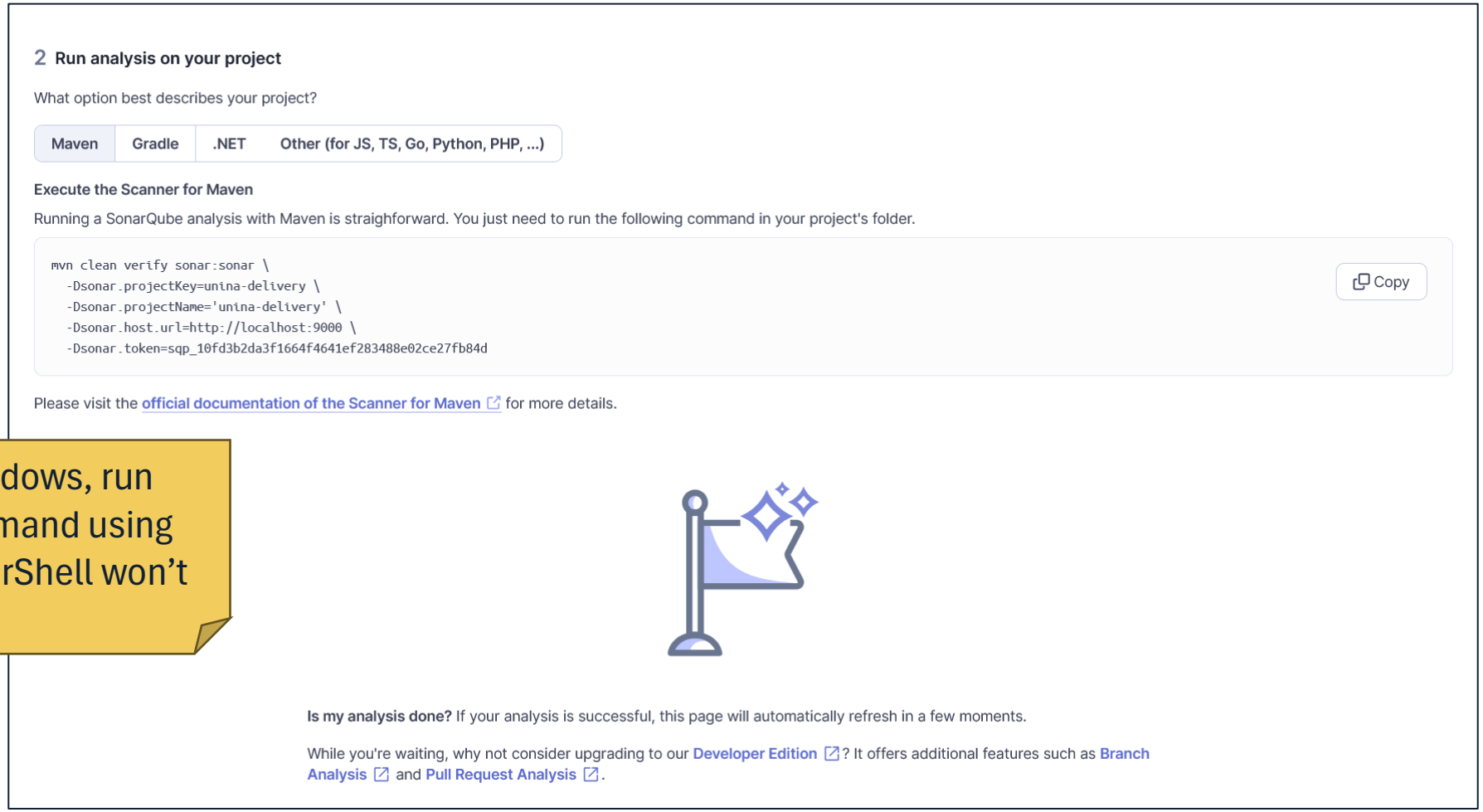
SonarQube Server: Local Analysis



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)

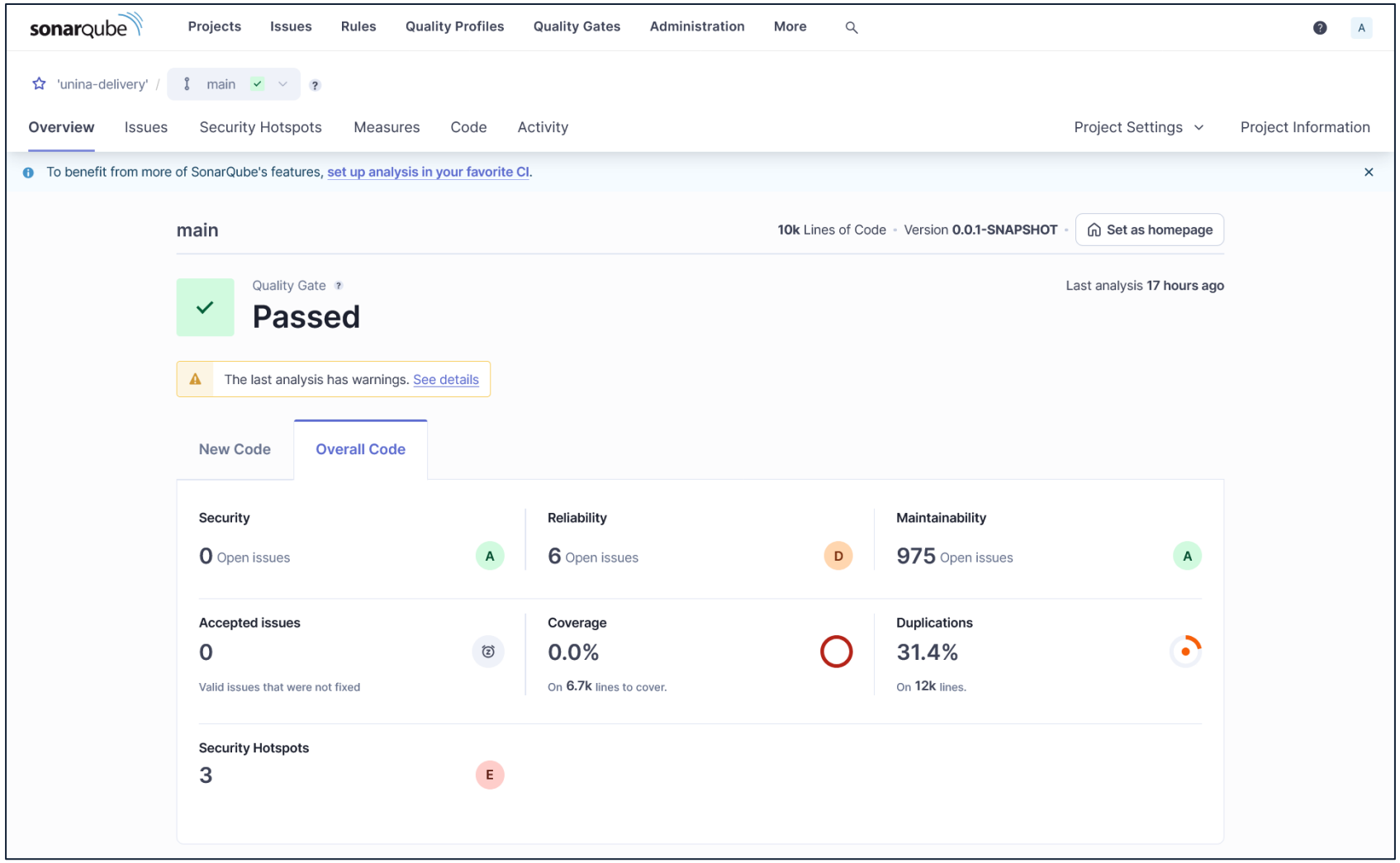
SonarQube Server: Running an Analysis

```
@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)

```
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>
```

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)

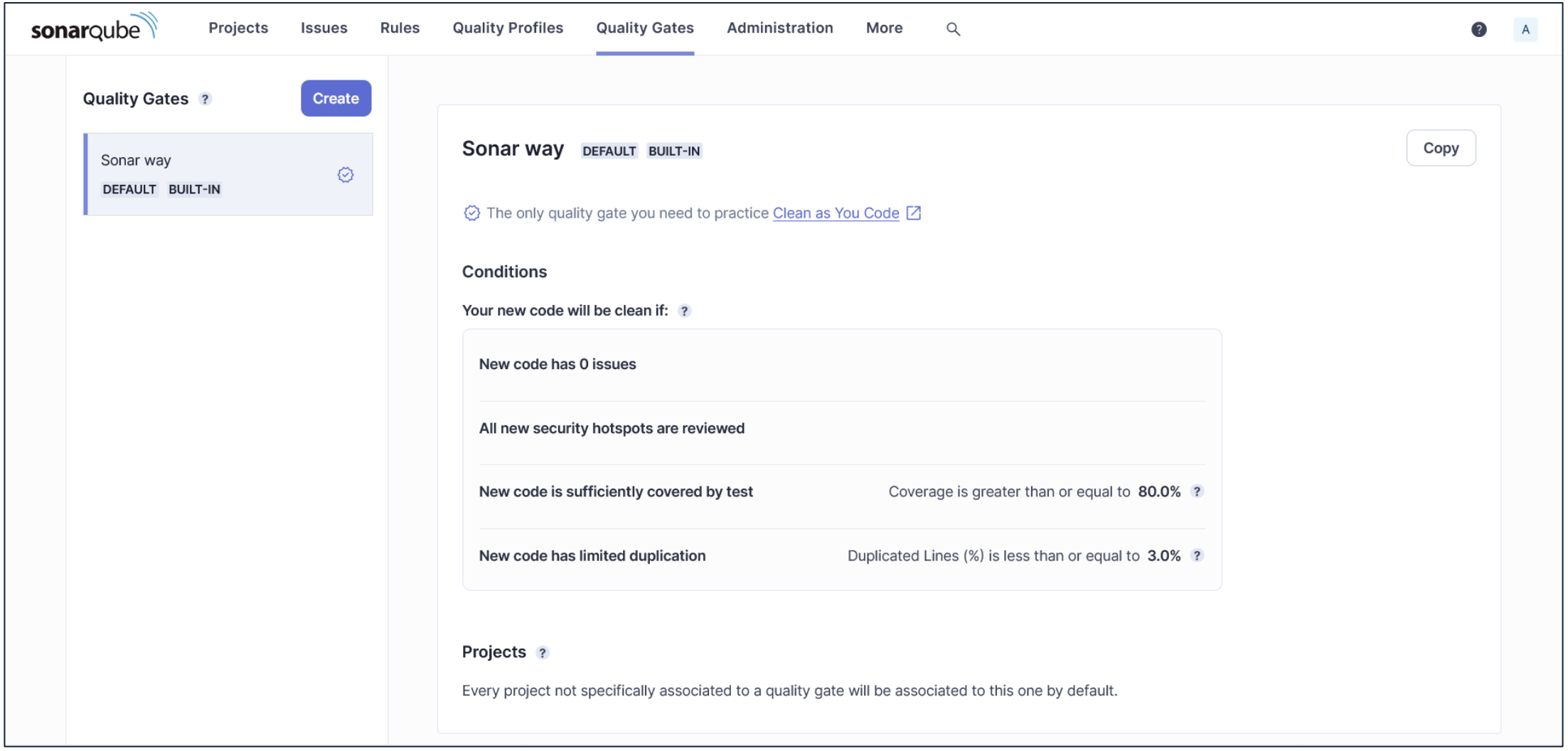
Continuous Delivery / Deployment (CD)

- 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

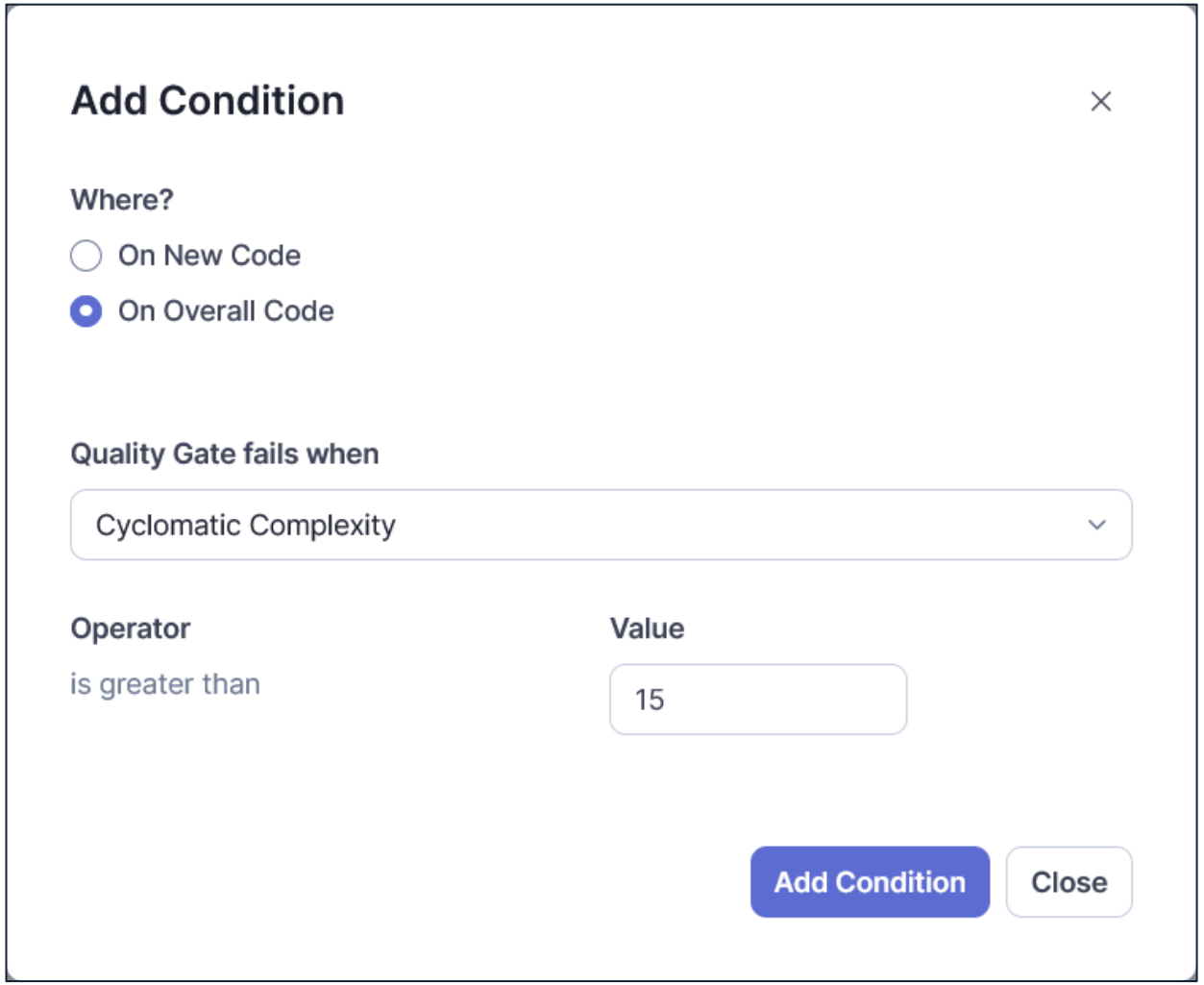
- In a CI/CD context, we may want to ensure that code adheres to established quality requirements
- With SonarQube, we can make a build fail based on quality requirements
- This is done with **Quality Gates**
  - A set of quality requirements that code (or new code) must satisfy
  - E.g.: no blocker 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

BEWARE: EXTRA MATERIALS AHEAD

- The remaining part of this slide deck is about **Assertion Frameworks**
- In particular, the slides describe **Hamcrest**, one of the most popular assertion frameworks, available for different programming languages
- Assertion frameworks are nice to know and an important tool to write effective tests...
- ... but they are **not required** for the **Software Engineering** course!

Assertion Frameworks

Assertion Frameworks

Assertion frameworks allow developers to write better assertions, and thus better tests, without re-inventing the wheel!

- More readable
- More flexible
- More informative error messages

Notable examples include: Hamcrest, Google Truth, AssertJ

What's wrong with standard JUnit assertions?

HAMCREST

MATCHERS

Getting to know Hamcrest

Hamcrest allows developers to declaratively define

Matchers that can be combined to create flexible expressions of intent.

- Not only for testing (UI Validation, data filtering, ...)
- Matcher objects represent conditions, and they can be combined declaratively to create natural-language like (fluent) assertions

Testing with Hamcrest

The entry point for Hamcrest in our tests is the method

```
<T> void assertThat(T actual, Matcher<? super T> matcher)
```

Testing with Hamcrest

The entry point for Hamcrest in our tests is the method

```
<T> void assertThat(T actual, Matcher<? super T> matcher)
```

```
// these express the same assertion
assertEquals(expected, actual);
assertThat(actual, equalTo(expected));
assertThat(actual, is(expected));
assertThat(actual, is(equalTo(expected)));

// and so do these
assertTrue(condition);
assertThat(condition, is(true));

// and these
assertNotNull(actual);
assertThat(actual, is(not(nullValue())));
```

Hamcrest – Core Matchers

- `equalTo(o)` given an `Object`, returns a Matcher that matches with any object which is equal to `o` (according to `o.equals()`);
- `not(m)` returns a Matcher that complements the given Matcher `m`;
- `is(k)` if given an object, returns `equalTo(k)`. If given a Matcher, returns the matcher without changing its behaviour;

```
assertThat("foo", is(not(equalTo("bar"))));
```

Hamcrest – String Matchers

Take a `String s` as input and return a suitable Matcher

- `containsString(s)` and `containsStringIgnoringCase(s)`
- `startsWith(s)` and `startsWithIgnoringCase(s)`
- `endsWith(s)` and `endsWithIgnoringCase(s)`

```
assertThat("Hamcrest", startsWith("Ham")); // passes
assertThat("Hamcrest", startsWith("ham")); // fails
assertThat("Hamcrest", containsStringIgnoringCase("CRE")); //passes
assertThat("Hamcrest", endsWithIgnoringCase("REST")); //passes
```

Hamcrest – Comparable Matchers

Take a `Comparable c` as input and return a suitable Matcher

- `greaterThan(c)` and `greaterThanOrEqualTo(c)`
- `lessThan(c)` and `lessThanOrEqualTo(c)`
- `closeTo(c,delta)` with `delta` being the admissible error

```
assertThat(10, is(greaterThanOrEqualTo(10)));
assertThat(10, is(lessThan(100)));
double a = 1.001, b = 1.002, delta = 0.002;
assertThat(a, is(closeTo(b, delta)));
```

Hamcrest – Logical Matchers

- `allOf(m1,…)` returns a matcher that matches the examined object only if it matches all the given matchers.
- `anyOf(m1,…)` returns a matcher that matches the examined object only if it matches one of the given matchers.
- `both(m1).and(m2)` and `either(m1).or(m2)` build matchers that match only if both (resp. either) `m1` and (resp. or) `m2` match.

```
assertThat("Hamcrest", allOf(startsWith("Ham"), endsWith("crest")));
assertThat(123, anyOf(is(lessThan(50)), is(greaterThan(120))));
assertThat(123, either(is(lessThan(50))).or(is(greaterThan(130))));
```

Hamcrest – Object Matchers

- hasProperty(s) matches with all objects with an «s» property
- hasProperty(s, m) matches with all objects whose «s» field matches the given matcher

```
Car car = new Car("Stelvio", 280, "Gasoline"); //model, hp, fuel
assertThat(car, hasProperty("fuel"));
assertThat(car, hasProperty("hp", is(greaterThan(200))));
```

Hamcrest – Collection Matchers: Iterables

- `hasItem(m)` matches with Iterables that contain an item matching with `m`
- `contains(k1,…)` matches only if each n-th element of the examined Iterable matches with the corresponding n-th matcher/object
- `containsInAnyOrder(k1,…)` matches only if each element of the examined Iterator matches with exactly one of the matchers/objects (same length required).

```
List<String> s = Arrays.asList("Anna","Bob","Carl");
assertThat(s, hasItem(both(startsWith("A")).and(endsWith("a"))));
assertThat(s, contains(startsWith("A"),startsWith("B"),startsWith("C")));
assertThat(s, containsInAnyOrder("Bob","Anna","Carl"));
```

Hamcrest – Collection Matchers: Maps

- `hasKey(m)` matches with Maps that contain a key matching with `m`
- `hasValue(m)` matches with Maps containing a value matching with m
- `hasEntry(m1,m2)` matches with Maps that contain a key matching with m1 whose value matches with m2.

```
Map<String,Integer> map = new HashMap<String,Integer>();
map.put("Anna", 1); map.put("Bob", 3); map.put("Carl", 3);
assertThat(map, hasKey(containsString("nn")));
assertThat(map, hasValue(is(greaterThan(2))));
assertThat(map, hasEntry(startsWith("A"), is(lessThan(2))));
```

Hamcrest – There's much more!

- Check out the docs if you want to know more!
- For a starter, you can take a look at the `Matchers` class.

Let’s get back at our example

```
@Test
void testDivisors() {
    List<Integer> divisors = MathUtils.getDivisors(8);
    List<Integer> expected = Arrays.asList(1,2,4,8);
    assertAll( // v3
        () -> assertTrue(expected.containsAll(divisors)),
        () -> assertTrue(divisors.containsAll(expected))
    );
    // with Hamcrest
    assertThat(divisors, containsInAnyOrder(1,2,4,8));
}
```

And here’s what an error message looks like with Hamcrest:

```
Expected: iterable with items [<1>, <2>, <4>, <8>] in any order
     but: no item matches: <1> in [<2>, <4>, <8>]
```

Another example

We need to check that in a `List<Car>` there is at least one `Car` such that its fuel is `"Electric"` or `"Hydrogen"`.

Another example – The classic way

We need to check that in a `List<Car>` there is at least one `Car` such that its fuel is `"Electric"` or `"Hydrogen"`.

```
boolean found = false;
for(Car c: Car.getCars()) {
    if(c.getFuel().equals("Electric") || c.getFuel().equals("Hydrogen")) {
        found = true;
        break;
    }
}
assertTrue(found);

> Error message:
Expected: <true> but was: <false>
```

Another example – The Hamcrest way

We need to check that in a `List<Car>` there is at least one `Car` such that its fuel is `"Electric"` or `"Hydrogen"`.

```
assertThat(Car.getCars(), hasItem(
    hasProperty("fuel", either(is("Electric")).or(is("Hydrogen")))
));


> Error message:
Expected: a collection containing
          hasProperty("fuel", (is "Electric" or is "Hydrogen"))
     but: mismatches were: [ property 'fuel' was "Hidrogen",
          property 'fuel' was "Gas", property 'fuel' was "Gas"]
```

Writing our own Matcher

Suppose we need to check that a `List<Student>` contains at least one `Student` whose name is not a palindrome.

Yes, we *really* do need that! 🙄

```
assertThat(list, hasItem(hasProperty("firstName", is(not(palindrome())))));
```

Hamcrest features a lot of built-in Matchers, but none to match palindrome strings, unfortunately.

Not too bad, we'll write it ourselves!

Writing our own Matcher

To create a new matcher, one needs to implement the Matcher interface:

```
public interface Matcher<T> {
    boolean matches(Object actual);
    void describeMismatch(Object actual, Description mismatchDescription);
}
```

Often, anyway, it is more practical to extend one of the abstract classes `BaseMatcher<T>` or `TypeSafeMatcher<T>`, which implement `Matcher<T>`.

Writing our own Matcher

```
public class IsPalindrome extends TypeSafeMatcher<String>{

    @Override
    public void describeTo(Description description) {
        description.appendText("a palindrome string");
    }

    @Override
    protected boolean matchesSafely(String item) {
        StringBuilder sb = new StringBuilder(item).reverse();
        return sb.toString().equalsIgnoreCase(item);
    }

    public static IsPalindrome palindrome() {
        return new IsPalindrome();
    }
}
```

Generates a description of the object we match

matchesSafely is called by TypeSafeMatcher. matches()

Static method to access our matcher, so we can use it as the built-in ones

Using our own matcher

```
import static org.hamcrest.Matchers.*;
import static it.unina.computerscience.softeng.junitdemo.examples.IsPalindrome.*;

// ...

assertThat(list, hasItem(hasProperty("firstName", is(not(palindrome())))));


> Error message:
Expected: a collection containing
          hasProperty("firstName", is not a palindrome string)
     but: mismatches were: [ property 'firstName' was "Anna",
          property 'firstName' was "Bob",  property 'firstName' was "Otto"]
```

Other Java Assertion frameworks

Other well-known Java assertion frameworks include:

- AssertJ (web)
- Google Truth (web)

They're less general than Hamcrest, and are based on method chaining.

```
// AssertJ
assertThat(gamma.getName()).startsWith("Er").endsWith("ich");
assertThat(theGangOfFour).contains(gamma,helm).doesNotContain(fowler);

// Truth
assertThat(theGangOfFour).containsExactly(gamma,helm,vliss,johnson).inOrder();
```

AssertJ / Truth vs Hamcrest

AssertJ / Truth

- Based on method chaining
  - No LISP-like parentheses hell
  - IDE auto-complete
- Generally can provide better error messages

Hamcrest

- More general
  - Applications outside of testing
  - Can be used to set expectations when mocking

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