TDD Workshop

Nuno Bettencourt (nmb@isep.ipp.pt)
@Vienna (Online), April-May, 2021
https://www.linkedin.com/in/nunobett/

Questions

When in doubt,
feel free to raise your hand,
anytime

Unit Testing

- **Unit Testing means testing individual units of behaviour**
- Should be something that is written instead of manually performed
- **Are pieces of code, comprised by input, conditions and outputs**
- Reduces refactoring bugs
- Reduces testing efforts

Unit Testing Tips

- Before writing a unit test, think about the following questions:
  - What are you testing?
  - What are the domain invariants?
  - What should it do?
  - How can the test be reproduced?
  - What is the actual output?
  - What is the expected output?

What Unit Tests should have?

Usually

- **Business Logic (domain) rules**
- Functions with Pre and Post Conditions
  - With Valid inputs
  - With Invalid Inputs
  - By Identifying all the exceptions
    - E.g.: factorial(-1)
  - For extreme value Limits
    - E.g. factorial(0)

Hardly

- User-Interface code
- Database Schemas
- Complex code that requires mocking

Terminology

- Test fixture
  - Define data (classes, initializations arguments) for tests
- **Unit-test**
  - A test to a single piece of code
- **Test case**
  - Several unit-tests for different input values
- Test suite
  - Test Case collection
- Test Runner
  - Unit-Testing software to run Unit-test and report the results

Unit Testing: factorial example

Code

```
public class Calculator {
   public int factorial(int v) throws IllegalArgumentException{
     if (v < 0) {
        throw new IllegalArgumentException();
     }
     if (v == 0) {
        return 1;
     }
     if (v == 1) {
        return 1;
     }
     return v * fatorial(v - 1);
  }
```

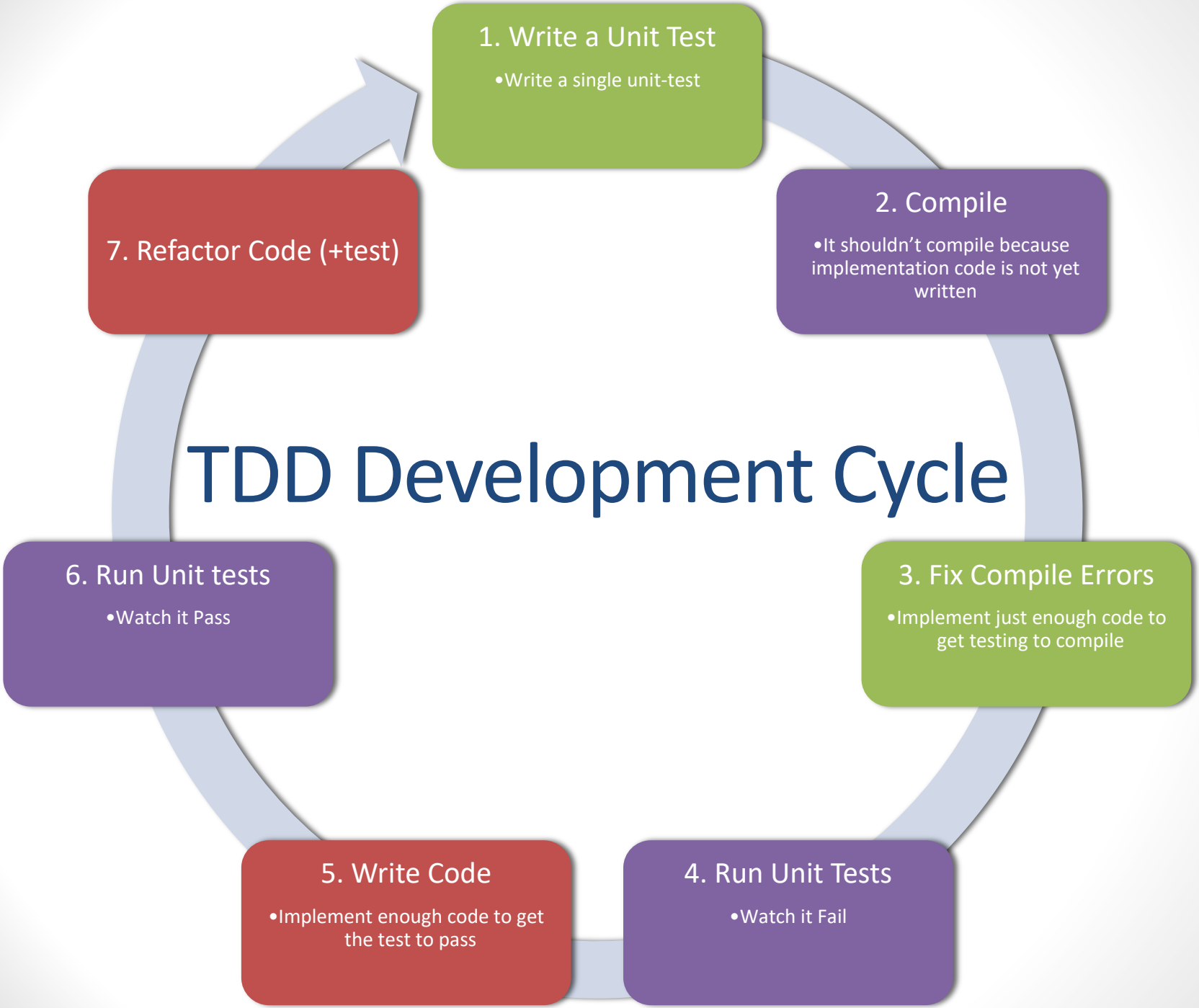
Test-Case*

```
public class CalculatorTest {
   @Test
   public void ensureFactorialOfMinusOneFails() {
      Assertions.assertThrows(IllegalArgumentException.class,
      () -> { new Calculator().factorial(-1); });
   }
   @Test
   public void ensureFactorialOfZeroIsOne() {
      assertEquals(new Calculator().factorial(0), 1);
   }
   @Test
   public void ensureFactorialOfOneIsOne() {
      assertEquals(new Calculator().factorial(1), 1);
   }
   @Test
   public void ensureFactorialOfTwoIsTwo() {
      assertEquals(new Calculator().factorial(2), 2);
   }
   @Test
   public void ensureFactorialOfFourIsTwentyFour() {
      assertEquals(new Calculator().factorial(4), 24);
   }
```

* for simplicity reasons, test-code has been reduced to the essential

TDD Definition

- Test-Driven Development (TDD) is a **software development process** that relies on the **repetition of a very short development cycle**
- Test-cases are a measure/specification of what code should do
- Methodology
  - TDD is the technique of writing test-cases ***before*** code
  - Testing ***leads*** to the development of source code
    - Only write source code when tests fail



TDD Development Cycle

Code Coverage

- Code Coverage **measures** the **degree** to which the source code of a program is executed when a particular test suite runs.

- Tools
  - Clover, Cobertura, Jacoco

- Result
  - The percentage of covered code

Code Coverage Example

Unit Test

```
@Test
public void
ensureSecondNegativeOperandWorks() {

 // Given
   int firstOperand = 10;
   int secondOperand = -5;
   int expected = 5;

   //When
   CalculatorExample calculator = new
CalculatorExample();
   int result = calculator.sum(firstOperand,
secondOperand);

   //Then
   assertEquals(expected, result);
}
```

Code

```
public int sum(int firstOperand, int
secondOperand)
{
   return firstOperand + secondOperand;
}
```

Code Coverage: Runner

- *<!-- Required for running unit tests -->*

```
<plugin>
   <artifactId>maven-surefire-plugin</artifactId>
   <version>2.22.2</version>
   <configuration>
      <includes>
         <include>**/Test*.java</include>
         <include>**/*Test.java</include>
         <include>**/*Tests.java</include>
         <include>**/*TestCase.java</include>
      </includes>
      <excludes>
         <exclude>**Main**</exclude>
      </excludes>
   </configuration>
</plugin>
```

Code Coverage: Reporting

- *<!-- Required for generating coverage report -->*

```
<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <version>0.8.6</version>
  <configuration>
    <excludes>
      <exclude>**/ui/**/*</exclude>
    </excludes>
  </configuration>
  <executions>
    <execution>
      <id>default-prepare-agent</id>
      <goals>
        <goal>prepare-agent</goal>
      </goals>
    </execution>
    <execution>
      <id>default-report</id>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Exercise: Bookmarking tool (i)

- As a user I want to bookmark a URL
  - URLs must be valid
- As a user I want to be able to Tag a URL with a keyword
- As a user when I add a duplicate bookmark, I want the system to increase the rating of that bookmark, because no exact duplicates should exist

Exercise: Bookmarking tool (ii)

- As a user I want to know how many of my bookmarks are secure URLs
- As a user I want a new bookmark to become associated with other bookmarks that are from the same domain
- As a user I want to be able to filter bookmarks by one keyword
- As a user I want to be able to filter bookmarks by one or more keywords