Unit Testing with JUnit

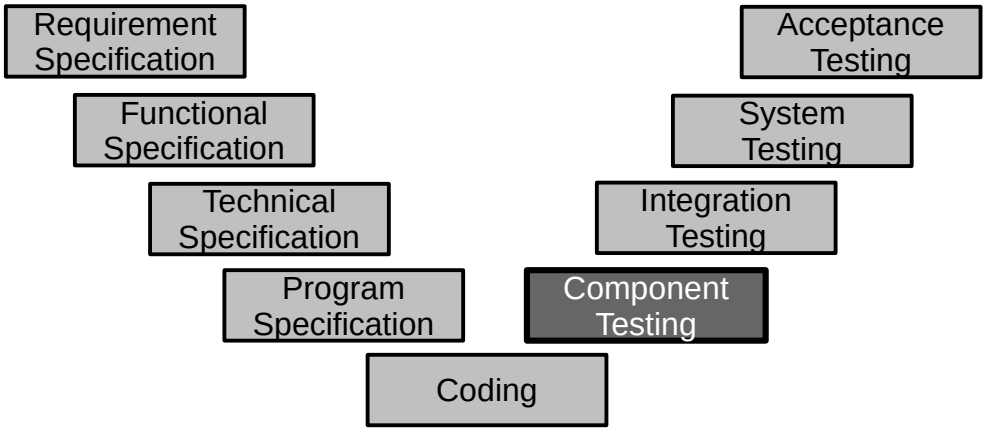
Professorship of Open Source Software

Friedrich-Alexander University Erlangen-Nürnberg

ADAP B04

Definition Unit Testing

- Recap
- Unit = Component (in our context)
- Often classes are seen as units



Example of Test Harness

- JUnit (Java Unit Testing Framework)
  - A test harness implemented as an object-oriented testing framework
  - Supports tests and test suites, set-ups, tear-downs, etc.
  - Small and simple, easy to learn
  - Well-supported by tools / integrated into IDEs like Eclipse

JUnit popularized unit testing: “Never in the field of software development have so many owed so much to so few lines of code.” [M07]

JUnit Information

- Available from http://junit.org
  - Comes as pre-installed plug-in with Eclipse and most other IDEs
  - See course literature for an introduction to JUnit
- Version history of JUnit
  - Prior to JUnit 4 conventions rather than annotations
  - Wahlzeit uses JUnit 4
- JUnit 5
  - Is the new major version of the testing framework
  - Is a complete rewrite of JUnit 4
  - Provides new foundation for developer-side testing on the JVM
  - Uses Java 8 features, for example, lambdas
  - Has a modular concept, imports only what is needed

JUnit Example: Component Under Test

- Scheduler for tasks that are triggered the first time on a certain point in time and then in a defined fix interval

JUnit Example: Simple Unit Tests (1/2)

- Annotate test method with **@Test**
- Conventions:
  - Containing class name ends with ‘Test’
  - Test methods start with ‘test’
  - File locations depend on build tool, e.g. Gradle
    - Sources: $project/src/main/java
    - Tests: $project/src/test/java
  - Test package hierarchy should mirror the main hierarchy

JUnit Example: Simple Unit Tests (2/2)

- **Assertions** for checking the results
- Explicit assertions / failures in code
  - assert(...)
  - fail(...)
- Annotations with expected results
  - @Test(expected = SomeException.class)
  - @Test(timeout = 500)

3A Pattern

1. Arrange
2. Act (execute)
3. Assert (check)

Test Results

1) **Pass**

2) **Fail**
   - a) Program is defect
   - b) Test is defect

3) Test execution error

JUnit Static Test Setup & Teardown

- Setting up / tearing down test environment **for every** test in a class
  - @Before
  - @After
- Setting up / tearing down test environment **once for all** tests in a class
  - @BeforeClass
  - @AfterClass

JUnit Dynamic Test Setup & Teardown

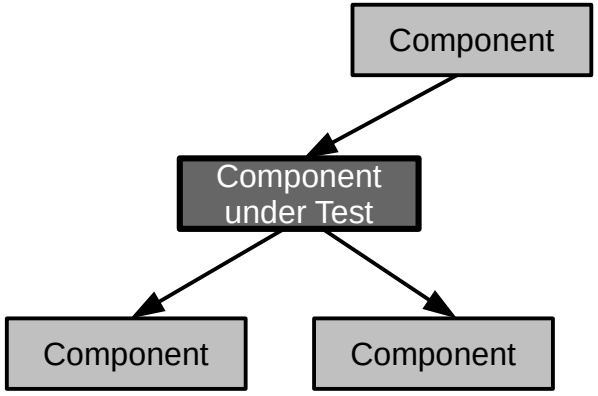
- Reusable Setup and Teardown
- TestRule
  - e.g. TmpDir Rule
    - Temporary directory that is cleared after each test case
  - Rule chain supports composition of test rules
    - Rule chain lines up test rules in sequence
    - Fluid programming style chains methods
  - Use @Rule and @ClassRule analogous to @Before and @BeforeClass
- ExternalResource
  - More complex set-ups to be run once or only a few times
  - Applies, for example, to heavyweight database set-up

Changes in JUnit Versions

- Tests are implemented in test classes
  - JUnit 3.8 or before
    - Start test method name with “test”
    - End test class name with “Test”
  - JUnit 4
    - Annotate test method with @Test
    - **Annotate set-up methods with @Before and @BeforeClass**
    - **Annotate tear-down methods with @After and @AfterClass**
    - End class name with Test (optional)
  - JUnit 5
    - Annotate test method with @Test
    - **Annotate set-up methods with @BeforeEach and @BeforeAll**
    - **Annotate tear-down methods with @AfterEach and @AfterAll**
    - End class name with Test (optional)

Test Drivers and Test Doubles

- Components are part of a dependency graph
- We need to isolate components in order to test them as a single unit
- **Using components** as inspiration for test drivers (calling the component under test)
- **Used components** need to be replaced by test doubles



Isolating Units with Test Doubles

- Test Doubles
  - Object or component that we install in place of the real component for a test
- Dummy Object
  - Placeholder object that is passed to the SUT as an argument (or an attribute of an argument) but is never actually used
- Test Stub
  - Replaces component that SUT depends on, configure indirect inputs to the SUT
- Mock Object
  - Test Stub + ability to verify inputs of the SUT by behaviour expectation
- Test Spy
  - Test Stub + ability to verify inputs of the SUT by recording calls to the spy that can be verified
- Fake Object
  - Replaces component with an alternative implementation of the same functionality

xUnit Test Patterns: Refactoring Test Code by Gerard Meszaros, Published by Addison-Wesley Professional, 2007

Mockito

- Available from https://site.mockito.org/
- Serves a variety of testing double functionality
  - Stubbing
  - Mocking
  - Spying
  - Etc.
- Easy syntax to create test doubles
- Easy syntax to verify test double behaviour
- Interacts very well with JUnit

Mockito Example: Component under Test

Inversion of Control

Mockito Example

- Mock creation with static *mock* method or with *@Mock* annotation
- Behaviour specification with static *when* method
- Behaviour verification with static *verify method*

{ **Inversion of Control**

Thank you! Questions?

dirk.riehle@fau.de – https://oss.cs.fau.de

dirk@riehle.org – https://dirkriehle.com – @dirkriehle

Credits and License

- Original version
  - © 2012-2021 Dirk Riehle, some rights reserved
  - © 2019-2021 Friedrich-Alexander University Erlangen-Nürnberg, some rights reserved
  - Licensed under Creative Commons Attribution 4.0 International License
- Contributions
  - Georg Schwarz (2019)