

A very very very very short intro to

Testing

Andreas Naderlinger

Testen ist Teil von V&V

- V&V
 - Validierung und Verifikation
- Validierung
 - Erstellen wir das richtige Produkt?
 - Soll gewährleisten, dass das Softwaresystem die Kundenerwartungen erfüllt.
- Verifikation
 - Erstellen wir das Produkt richtig?
 - Überprüft, ob die Software die vorgegebenen funktionalen und nichtfunktionalen Anforderungen erfüllt.

Unterscheidungen

- 3 Testphasen
 - Entwicklertests (development testing)
 - Freigabetests (release testing)
 - Benutzertests (user testing)
- Manual
- Automatic
- Blackbox
 - mit Spezifikation
- Whitebox
 - mit Code

Entwicklertests

- Modultests (unit testing)
 - Programmeinheiten oder Objektklassen
 - → Funktionalität
- Komponententests (component testing)
 - Mehrere einzelne Einheiten
 - → Komponentenschnittstellen
- Systemtests (system testing)
 - Einige oder alle Komponenten
 - → Interaktion zwischen Komponenten

Modultests (unit testing)

- Alle zum Objekt definierten Operationen testen
- Alle Attributwerte des Objekts setzen und abfragen
- Das Objekt in alle möglichen Zustände versetzen

- Normale Eingaben
- Ungewöhnliche Eingaben

- Und trotzdem

Tests können nur die Anwesenheit von Fehlern aufzeigen, nicht ihre Abwesenheit.

(Dijkstra et al., 1972)

Äquivalenzklassen (equivalence partition)

- Man kann nicht alle Eingaben testen
- Systematische Vorgehensweise
 - Identifizieren aller Ein- und Ausgabeklassen eines Systems oder einer Komponente
- Ein Programm sollte alle Elemente einer Klasse auf die gleiche Art verarbeiten.
- Testfälle
 - an Klassengrenzen
 - in der Mitte von Klassen

Test-Driven Development (TDD)

- 1. Add a test**
Derived from specification & requirements
 - 2. Run all tests and see if the new one fails**
To test the test itself & the test environment
 - 3. Write some code**
So that tests pass, but no additional functionality, as simple as possible: KISS, YAGNI
 - 4. Run the automated tests and see them succeed**
 - 5. Refactor code**
Clean up, remove duplication, use tests for confirmation
- **Repeat**
Add functionality with another new test

TDD Discussion

- Unit tests are „executable documentation“
 - Tests show others how to use the code
 - Unlike written documentation, tests can never be out of sync
- Problem: It can be difficult to write tests for some functionality (e.g. user interfaces)
- Test-driven development needs automated unit tests
 - Faster & more reliable than manual tests
 - Eases continuous integration – all tests are run before code is added to a common code base

JUnit

- Unit testing framework for Java
- Current stable version: ≥ 4.12
- A **software framework** is an abstraction in which common code providing generic functionality can be selectively overridden or specialized by user code providing specific functionality.
- Also available for other languages („xUnit“)
- Other development methodologies than TDD benefit too

Java Annotations

- Annotations (metadata) provide data about a program that is not part of the program itself
- Examples for compiler annotations:

```
@SuppressWarnings (value = "unchecked")  
void myMethod() { }
```

```
@Override  
void mySuperMethod() { }
```

- Since Java 5.0
- Can be customized

JUnit main annotations

Annotations for JUnit test methods:

- **@Test** – indicates a method containing a test
- **@Test(timeout = [time in ms])** - fails if timeout is reached
- **@Test(expected = SomeException.class)** - only passed if SomeException is thrown
- **@BeforeClass** - run before any test has been executed
- **@Before** - run before each test.
- **@After** - run after each test
- **@AfterClass** - run after all the tests have been

Running Tests

- Run `org.junit.runner.JUnitCore [class to test] ...`
- Runs all methods marked with `@Test`
- Displays summary and information on failed tests
- Assertion methods can be used to check results (not to be confused with the `assert` keyword)

Assertions

- `org.junit.Assert` contains assertion functions
- `assertEquals(String message, Object expected, Object actual)`
- `assertNull(String message, Object object)`
- `assertTrue(String message, boolean condition)`
- `fail(String message)`
- ...

- **See JUnit JavaDoc**

<http://kentbeck.github.com/junit/javadoc/latest/>

- **Example:**

```
@Test
public void test() {
    int x = 2 + 2;
    assertEquals("Test if 2 + 2 = 4", 4, x);
}
```

Example

```
package ex02a_junit;

public class Adder {

    private int result;

    public Adder() {
        result = 0;
    }

    public void add(int x) {
        result += x;
    }

    public int getResult() {
        return result;
    }
}
```

```
package ex02a_junit;

import org.junit.*;
import static org.junit.Assert.*;

public class AdderTest {

    Adder adder;

    @Before
    public void setup() {
        adder = new Adder();
    }

    @Test
    public void testAdd() {
        adder.add(1);
        adder.add(4);
        assertEquals("Test addition of 1 and 4", 5, adder.getResult());
    }

    public static void main(String[] args) {
        org.junit.runner.JUnit4Core.main("ex02a_junit.AdderTest");
    }
}
```

JUnit Installation

- Download from <http://junit.org/>
- Put junitXXX.jar in classpath
- Study & test examples