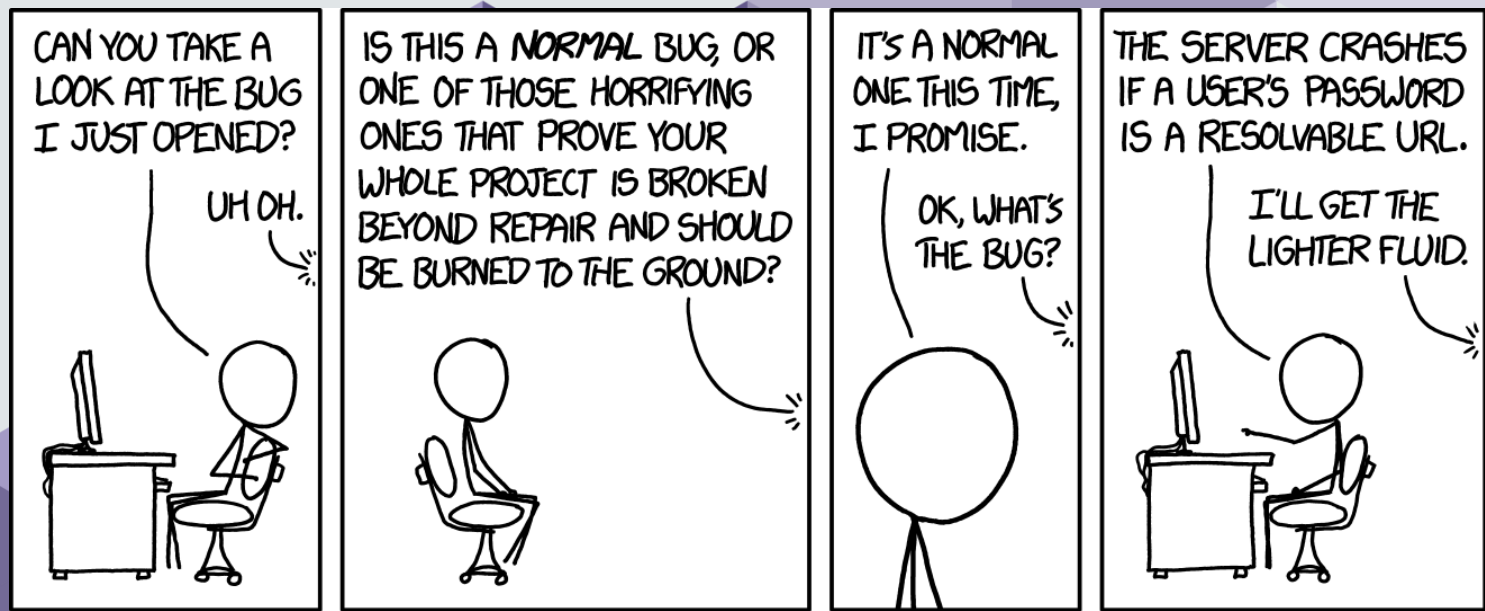


TESTEN

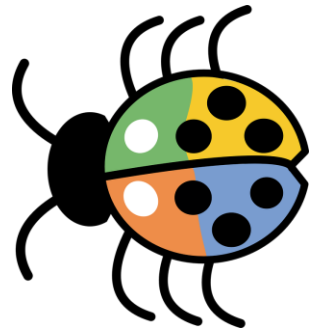
Bernhard Fuchs

basierend auf Folien von Christian Hofer



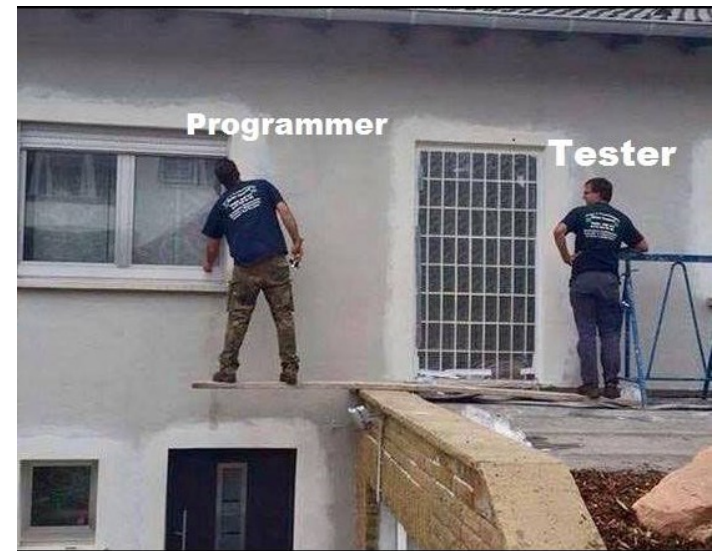
INHALTE & ZIELE

- Motivation
- Begriffe
- Debuggen und Testen
- Unit Testing
- JUnit Framework
- Unsere ersten Schritte
- Was gibt es noch



TESTEN SIE IHREN CODE!

- Sonst wird jemand anders sehr schnell Ihren Fehler finden!
- Gleich den Fehler zu finden ist immer besser als auf Tester*in (oder Kund*in) zu warten.



Ein Test ist der erste User Ihres Codes!

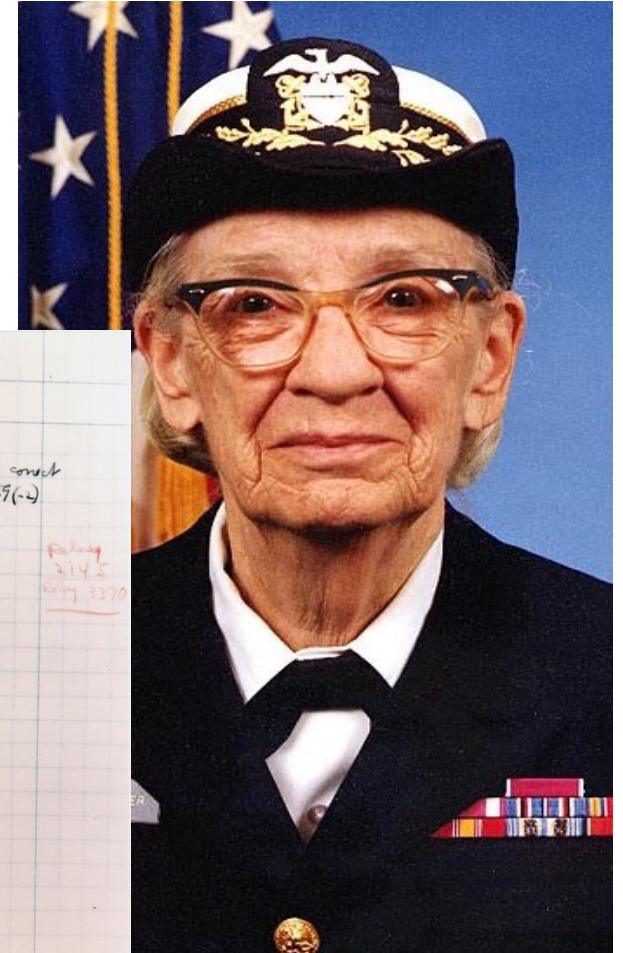
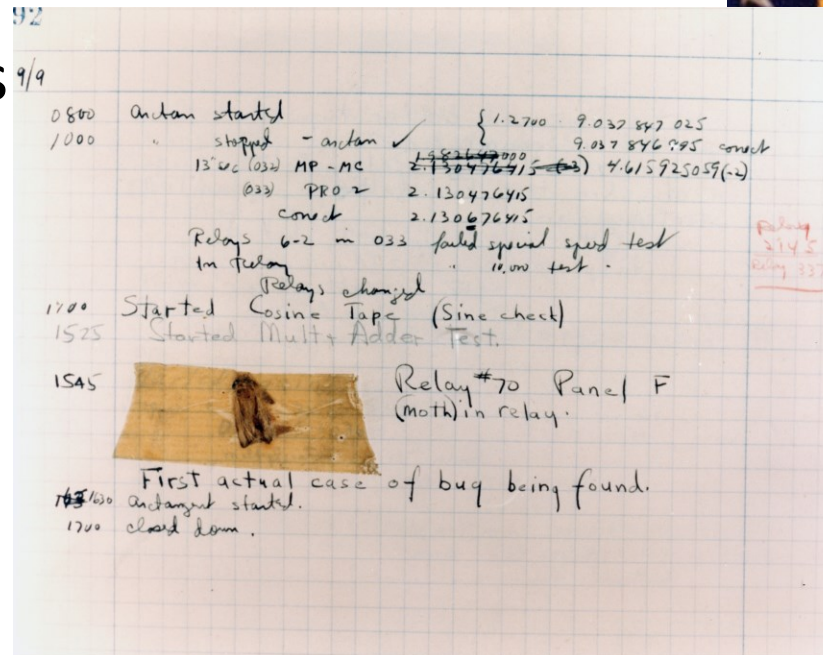
WARUM WIR AUTOMATISIERUNG BENÖTIGEN

- Am Anfang schreiben wir nur einfache Methoden
 - Mit Ausgaben überprüfen wir ob sie funktionieren
- Durch kontinuierliche Weiterentwicklung an einem Projekt wird Software immer größer und komplexer
 - Wir brauchen mehr als nur Ausgaben die wir manuell interpretieren.
- Wir brauchen mächtige Werkzeuge die uns unterstützen unseren Source Code immer wieder und halb automatisch zu testen!



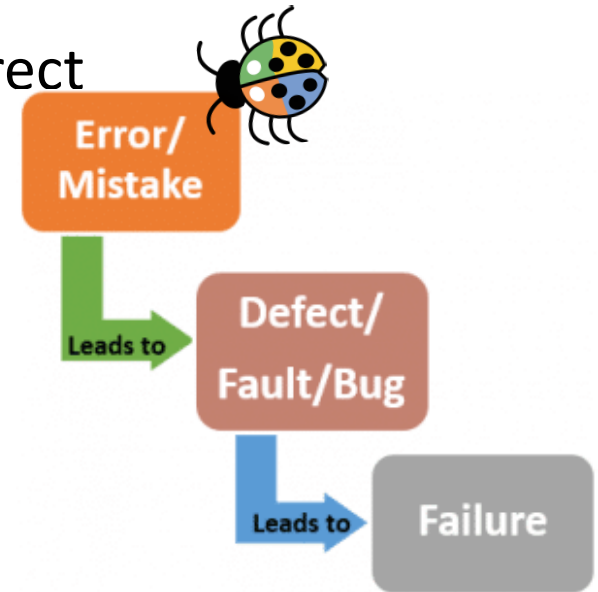
DER ERSTE BUG

- Seit 14 Jhd.
 - „object of terror“
- Motte in Relais
Grace Hopper
(1947)



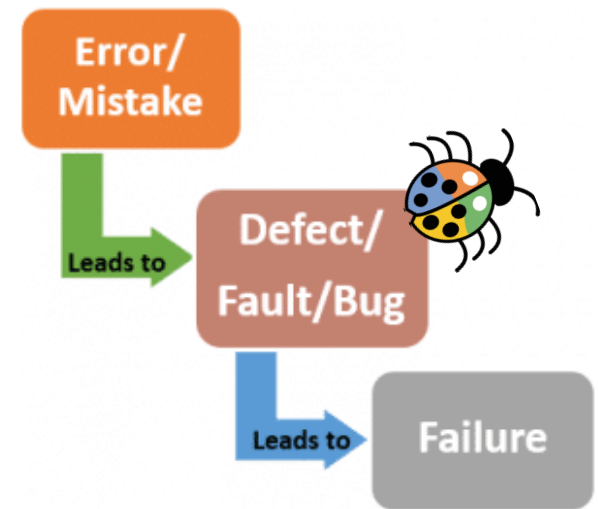
UNTERSCHIEDLICHE BEDEUTUNG BEGRIFF FEHLER

- Error (oder Mistake):
 - „A human action that produces an incorrect result“ (ISTQB)
- Ein Error kann zu einem Defect führen
- Beispiele
 - Syntax oder logisch
 - Requirementsanalyse
 - Testen
 - ...



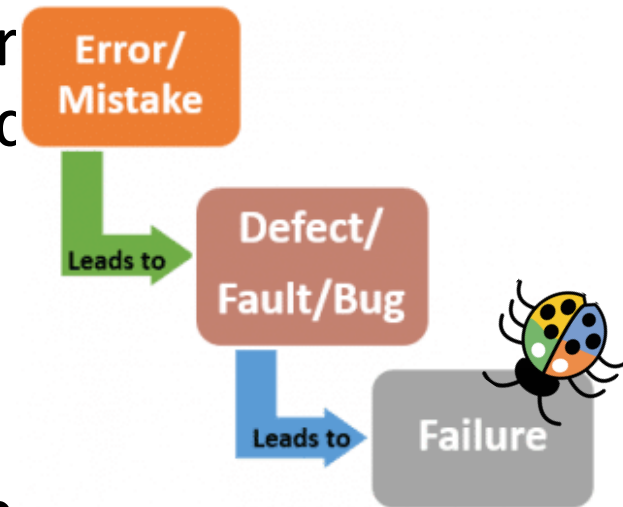
UNTERSCHIEDLICHE BEDEUTUNG BEGRIFF FEHLER

- Defect (auch bekannt als Bug oder Fault)
 - „An imperfection or deficiency in a work product where it does not meet requirements or specifications.” (ISTQB)
- Ein Defect kann zu einem Failure führen
 - (unter gewissen Bedingungen)



UNTERSCHIEDLICHE BEDEUTUNG BEGRIFF FEHLER

- Failure
 - „An event in which a component or system does not perform a required function within specified limits.“ (ISTQB)
- (Kann zu unglücklichen Kund*innen bzw. Umsatzverlust führen)



VOM ERROR ZUM FAILURE (INFEKTIONSKETTE)

- **Error**



- Error made by developer
- Can cause a defect



Defect

- Defect in program state
- Can cause a failure



Failure

- Issue in delivered program

LÖSUNGEN?

Wie haben Sie bis jetzt getestet?

```
System.out.println("it work's");
```

```
//System.out.println("test 1");
```

```
if(something == true)
```


```
    System.out.println("it works now");
```



DEBUGGEN UND TESTEN

“Program testing can be a very effective way to show the **presence of bugs**, but is hopelessly **inadequate for showing their absence**”

Dijkstra

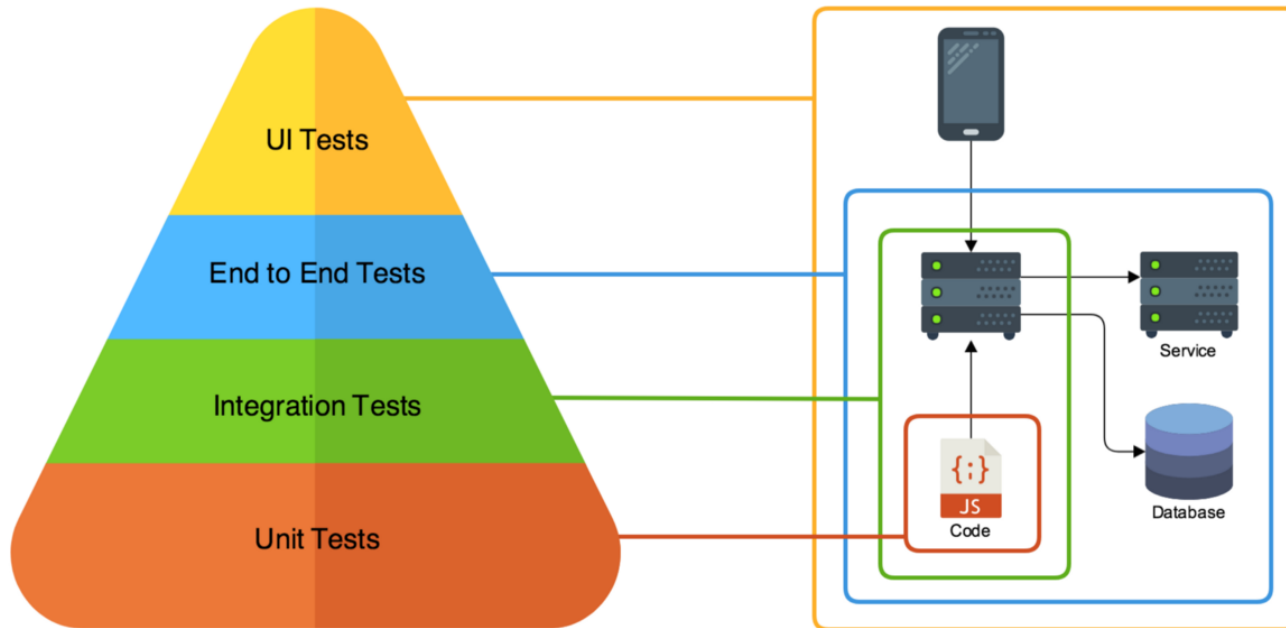
 Debuggen hilft uns die Infektionskette nachzuvollziehen.



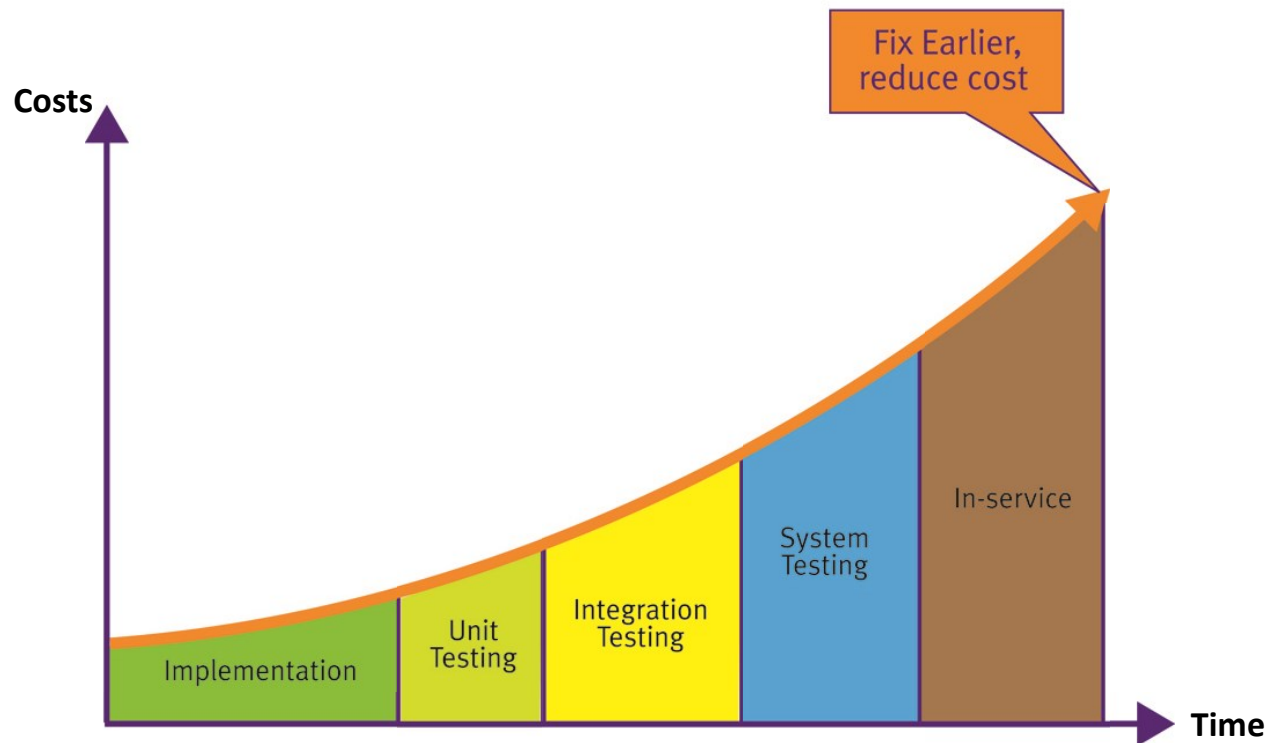
TESTING FOR DEBUGGING

- Was tun **nachdem Fehler bereinigt wurde?**
- **Testen** den Source Code **erneut** damit
 - **Reproduziere** Anwendungsfälle/Fehler
 - **Verifizieren** Fehlerlösung
 - **Ausführen** aller Tests vor nächster Veröffentlichung
 - **Automatisieren** Testfälle
- **Nur funktionsfähigen Source Code einchecken** in das remote repository

TEST PYRAMIDE



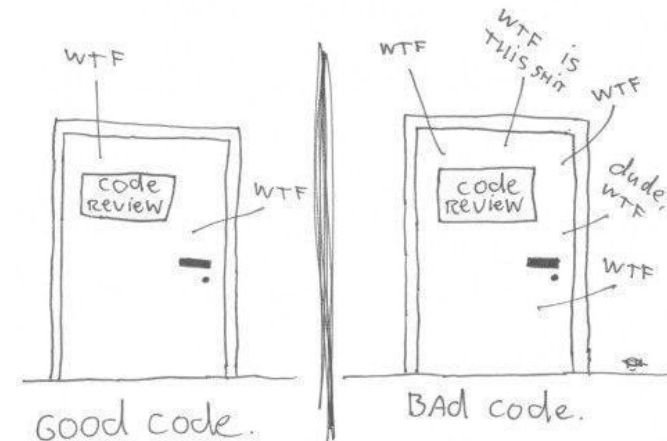
JE FRÜHER DESTO BESSER



TESTAUTOMATISIERUNG

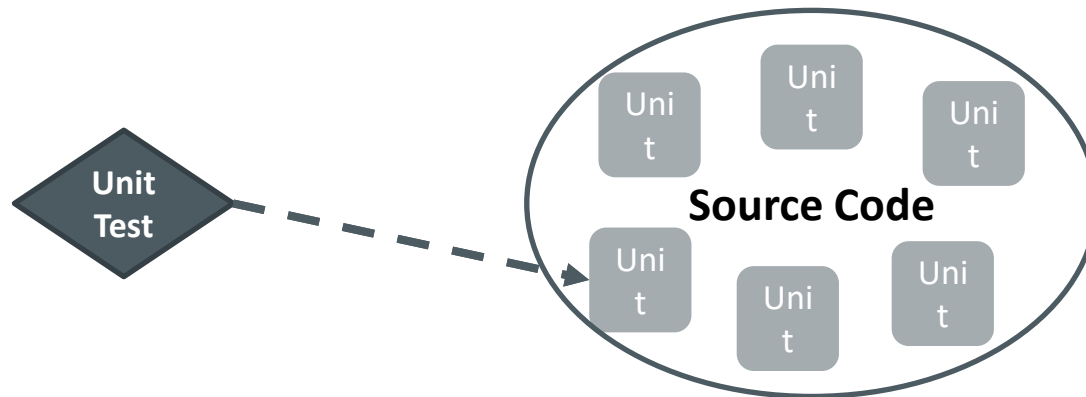


The only valid measurement
of code quality: WTFs/minute



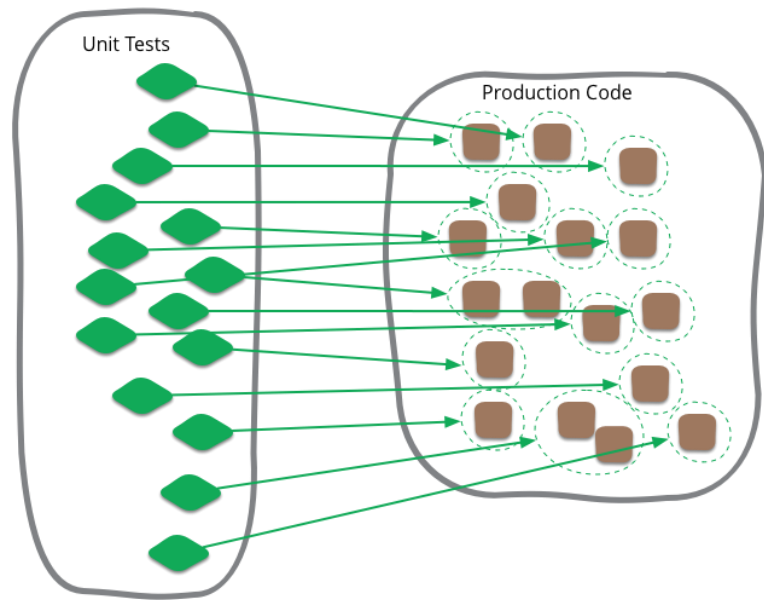
KONZEPT XUNIT TESTS

- Testen **kleine Teile** des Source Codes (sogenannte Einheiten units)
- **Verfügbar** in vielen Sprachen



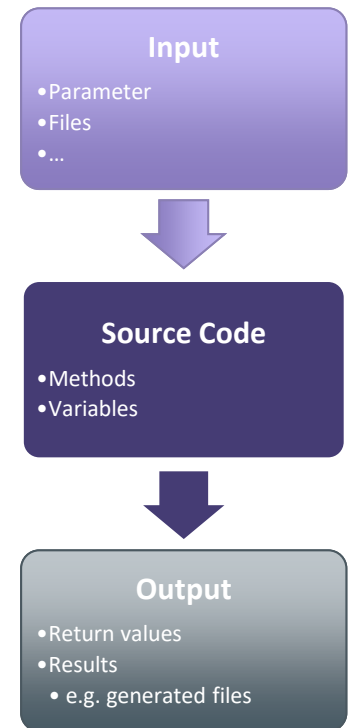
UNIT TESTS UND SOURCE CODE

- Der ganze Code **soll getestet** werden so möglich.
- Jeder Test erhöht die **test coverage**.
- Schreiben kurzen **Test Code** um kleinen Teil des **Source Codes** zu überprüfen

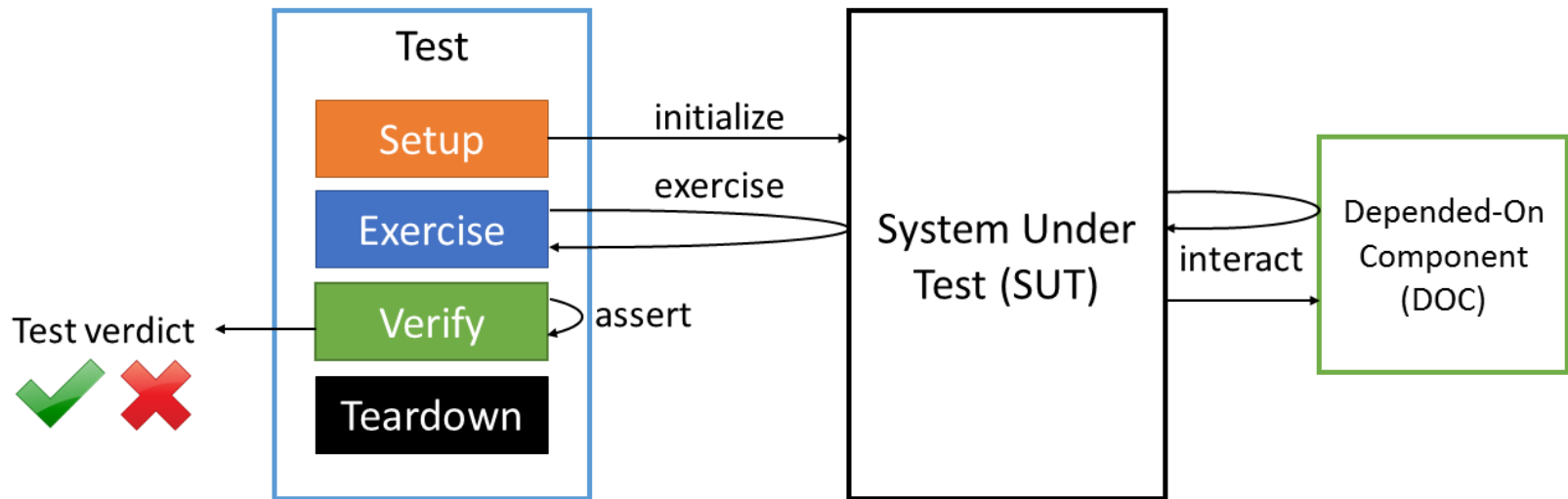


UNIT TEST IN A NUTSHELL

- **Testen** den Anwendungscode
- **Verifizieren** das Verhalten von Methoden
- Unterschiedliche **Inputs**
- Verhalten des **Source Code**
- Überprüfung **Outputs**



TEST STRUKTUR



JUNIT, EIN SIMPLES FRAMEWORK

- Simple Java Framework um **wiederholbare Tests zu schreiben**
- Tests **benötigen keinen Menschen zur Entscheidung ob etwas funktioniert oder nicht**
- **Leicht** viele hintereinander auszuführen



- Nur kleine Unterschiede zwischen **JUnit 4 und 5**

JUNIT 4 BEISPIEL (ALTE VERSION)

```
public class SimpleTest{  
    @Before  
    public void setUp() { /* ... */ }  
  
    @After  
    public void tearDown() { /* ... */ }  
  
    @Test  
    public void testMethod() {  
        /* ... */  
  
        Assert.assertTrue( /* ... */ );  
    }  
}
```

JUNIT 5 BEISPIEL

```
public class SimpleTest{  
    @BeforeEach  
    public void setUp() { /* ... */ }  
    @AfterEach  
    public void tearDown() { /* ... */ }  
    @Test  
    public void testMethod() {  
        /* ... */  
        Assert.assertTrue( /* ... */ );  
    }  
}
```


JUNIT 4 / 5 ANNOTATIONS

- **@Test** identifiziert eine Testmethode – muss keine Parameter haben bzw. void sein.
- **@Ignore** markiert eine Testmethode die ignoriert werden soll (**@Disabled**)
- **@Before**, **@After** zum Initialisieren und Freigeben vor bzw. nach jeder Testmethode
- **@BeforeEach**, **@AfterEach** zum Initialisieren und Freigeben vor bzw. nach jeder Testmethode (JUnit 5)

ANNOTATIONS:

JUNIT 4

- `@Test`
- `@Ignore`
- `@Before`, `@After`
- `@BeforeClass`, `@AfterClass`
- `@Test(expected=Exception.class)`
- `@Test(timeout=1000)`

JUNIT 5

- `@Test`
- `@Disabled`
- `@BeforeEach`, `@AfterEach`
- `@BeforeAll`, `@AfterAll`
- ~~`@Test(expected=Exception.class)`~~
 - In Testmethode mit **`assertThrows`**

JUNIT ASSERTIONS

- Fehlgeschlagene Assertions werden aufgezeichnet
- Direkt verwenden:
`Assert.assertEquals(...)`
 - Meistens zwei Parameter
 - Erwarteter Wert
 - Tatsächlicher Wert
 - Falls beide gleich dann Ergebnis positiv!!

JUNIT TEST VIER PHASEN

JUnit Test

Four Phases

```
public class SimpleTest{  
    @Before  
    public void setUp() { /* ... */ }  
  
    @After  
    public void tearDown() { /* ... */ }  
  
    @Test  
    public void testMethod() {  
        /* ... */  
        Assert.assertTrue( /* ... */ );  
    }  
}
```

Setup

Teardown

Exercise

Verify

SIMPLER JUNIT 4 / 5 TEST

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

import org.junit.Test;

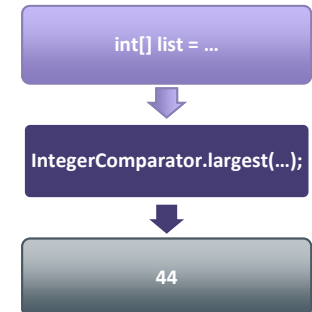
public class TestIntegerComparator {

    @Test
    public void testLargestNumber() {
        // create demo list
        int[] list = new int[]{ 12, 23, 9, 44, 2 };
        assertEquals(44, IntegerComparator.largest(list));
    }
}
```

Setup (red dashed box)

Verify (green dashed box)

Exercise (blue dashed box)



ERSTES BEISPIEL

- Wie können wir JUnit5 integrieren?



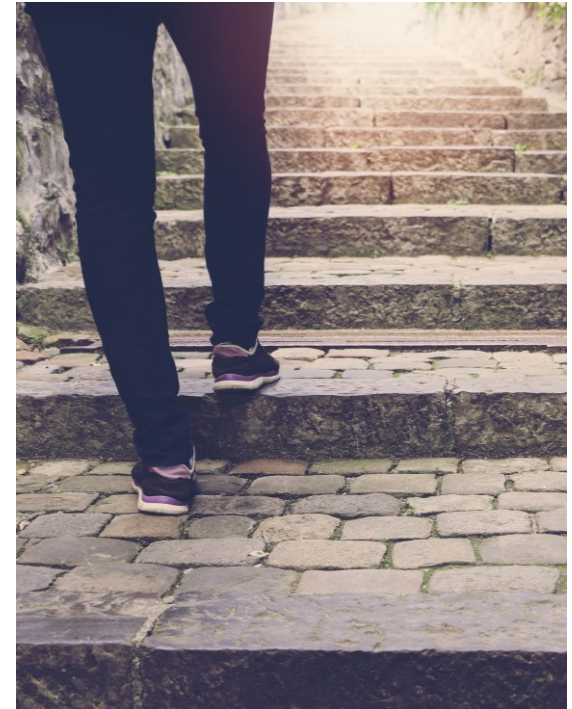
ERSTES BEISPIEL

- Spezifizieren Abhängigkeit in pom.xml (Maven)
 - Siehe Dokumentation

```
<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.8.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.8.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-params</artifactId>
    <version>5.8.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```


(MÖGLICHEE) NÄCHSTE SCHRITTE

- Weitere JUnit Features
- Mock Objekte
- Hamcrest
- Softwaretest Methoden
 - Statisches vs. dynamisches Testen
 - Black-box vs. white box Testen
- Software Entwicklungsprozesse
 - Test-driven development (TDD)
 - Behavior-driven development (BDD)



BEISPIEL

1. Create and setup a "tests" folder

- In the Project sidebar on the left, right-click your project and do New > Directory. Name it "test" or whatever you like.
- Right-click the folder and choose "Mark Directory As > Test Source Root".

2. Adding JUnit library

- Right-click your project and choose "Open Module Settings" or hit F4. (Alternatively, File > Project Structure, Ctrl-Alt-Shift-S is probably the "right" way to do this)
- Go to the "Libraries" group, click the little green plus (look up), and choose "From Maven...".
- Search for "junit" -- you're looking for something like "junit:junit:4.11".
- Check whichever boxes you want (Sources, JavaDocs) then hit OK.
- Keep hitting OK until you're back to the code.

3. Write your first unit test

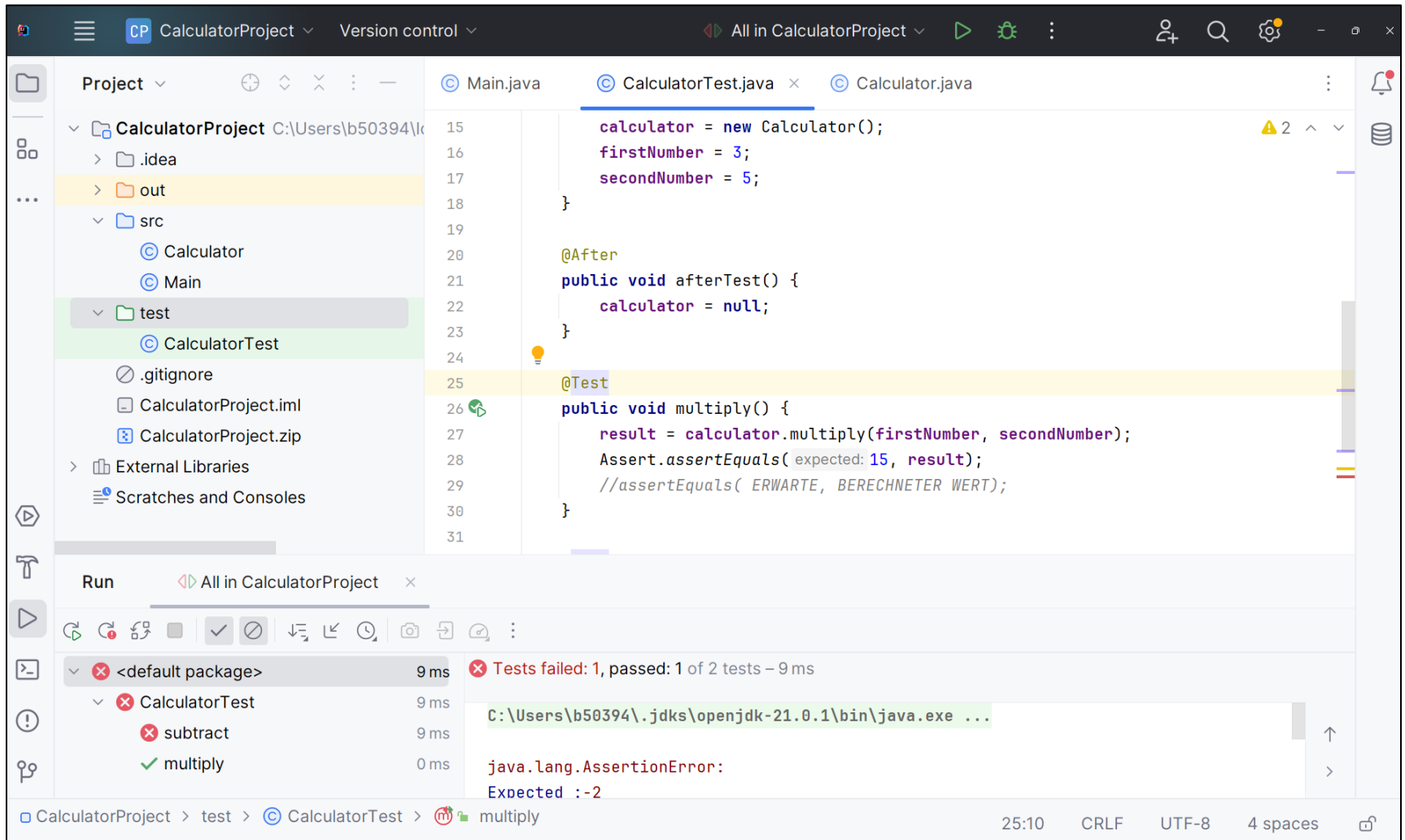
- Right-click on your test folder, "New > Java Class", call it whatever, e.g. MyFirstTest.
- Write a JUnit test -- here's mine:

```
import org.junit.Assert;
import org.junit.Test;

public class MyFirstTest {
    @Test
    public void firstTest() {
        Assert.assertTrue(true);
    }
}
```

4. Run your tests

- Right-click on your test folder and choose "Run 'All Tests'". Presto, testo.
- To run again, you can either hit the green "Play"-style button that appeared in the new section that popped on the bottom of your window, or you can hit the green "Play"-style button in the top bar.



REFERENZEN

- *Andreas Zeller*
Why Programs Fail
A Guide to Systematic Debugging
dpunkt.verlag, 2009
- *Gerard Meszaros*
xUnit Test Patterns, Refactoring Test Code
2007, Addison Wesley
- David Thomas, Andrew Hunt
The Pragmatic Programmer – 20th Anniversary Edition
2019, Pearson Education

LINKS

- *Unit Test*
Martin Fowler
<https://martinfowler.com/bliki/UnitTest.html>
last time visited: 14.03.2021
- *JUnit 4*
JUnit
<https://junit.org/junit4/>
last time visited: 21.04.2020
- *JUnit 5*
JUnit
<https://junit.org/junit5/>
last time visited: 14.03.2021