Modultests mit JUnit

MW II/Fallstudie Systemanalyse

WWI10SWMA - Studienhalbjahr 3

Dozent: Gregor Tielsch

Modultests mit JUnit

Fehlerfreie Software

- völlig fehlerfreie Software weder erreichbar noch nachweisbar
- steigt die Komplexität sinkt die Überblickbarkeit
- bei gut brauchbaren Programmen spricht man eher von Stabilität und Robustheit
- Software gilt als stabil bzw. robust, wenn Fehler nur sehr selten auftreten

http://de.wikipedia.org/wiki/Programmfehler

Folgen von Programmfehlern

Beim Kampfflugzeug F-16 brachte der Autopilot das Flugzeug in Rückenlage, wenn der Äquator überflogen wurde. Dies kam daher, dass man keine "negativen" Breitengrade als Eingabedaten bedacht hatte. Dieser Fehler wurde sehr spät während der Entwicklung der F-16 an Hand eines Simulators entdeckt und beseitigt.

http://de.wikipedia.org/wiki/Programmfehler

Folgen von Programmfehlern

1999 verpasste die NASA-Sonde Mars Climate Orbiter den Landeanflug auf den Mars, weil die Programmierer das falsche Maßsystem verwendeten - Pound-force · Sekunde statt Newton · Sekunde. Die NASA verlor dadurch die Sonde.

http://de.wikipedia.org/wiki/Programmfehler

Hohe Software-Qualität

- Statische Methoden
  - Code-Analyse durch formale Prüfungen
  - Code Reviews / Inspections
- Dynamische Methoden
  - Debugging
  - Testen

Software-Tests

- Nach ANSI/IEEE Std. 610.12-1990 ist Test

„the process of operating a system or component under specified conditions, observing or recording the results and making an evaluation of some aspects of the system or component.“

Testarten

- Modultest (Komponententest oder Unit-Test)
- Integrationstest
- Systemtest
- Abnahmetest

Ziele der Testautomatisierung

Qualität verbessern

- Tests sollten uns dabei helfen die Qualität zu verbessern.
  - Tests als Spezifikation
  - Tests wirken „fehlerimprägnierend“ (bug repellent)
  - Lokalisieren Fehler

Verständnis über das System verbessern

- Tests sollten uns dabei helfen das System Under Test (SUT) zu verstehen.
  - System Under Test -> das zu testende System in einem Testszenario
  - Tests als Dokumentation

Risiken reduzieren

Tests sollten Risiken reduzieren.

- Tests als Sicherheitsnetz
- Tests dürfen nicht schaden

-> Testcode aus Produktivcode heraushalten

Einfach auszuführen

- Es sollte einfach sein, Tests auszuführen
  - Vollautomatisierung: Tests laufen ohne manuelle Intervention und ohne hohen Aufwand
  - Selbstprüfend: der Test enthält die Prüfung, ob der erwartete Zustand des SUT korrekt ist
  - Wiederholbar: Tests können mehrfach hintereinander ausgeführt werden und führen stets zum gleichen Ergebnis ohne manuelle Intervention

Einfach zu schreiben und zu warten

- Es sollte einfach sein, Tests zu schreiben und zu warten
  - Einfache Tests: Tests sollten kurz sein und genau eine Bedingung testen; Ausnahme Akzeptanztests, die ein Nutzungsszenario des Kunden dokumentieren
  - Ausdrucksstarke Tests: der Testcode sollte verständlich und ausdrucksstark sein, ggf. Testcode in Hilfsmethoden auslagern
  - Separation of Concerns: 1) Testcode von Produktivcode trennen und 2) jeder Test sollte sich auf ein Szenario und eine Bedingung konzentrieren

Zusammenfassung: Ziele

- Prüft Korrektheit eines Softwarebausteins
- Jederzeit und beliebig oft wiederholbar
- Ergebnis bei gleichen Parametern gleich
- Reduziert das Restrisiko verbleibender Fehler
- Deckt Fehler auf, beweist nicht Fehlerfreiheit

Die xUnit-Familie

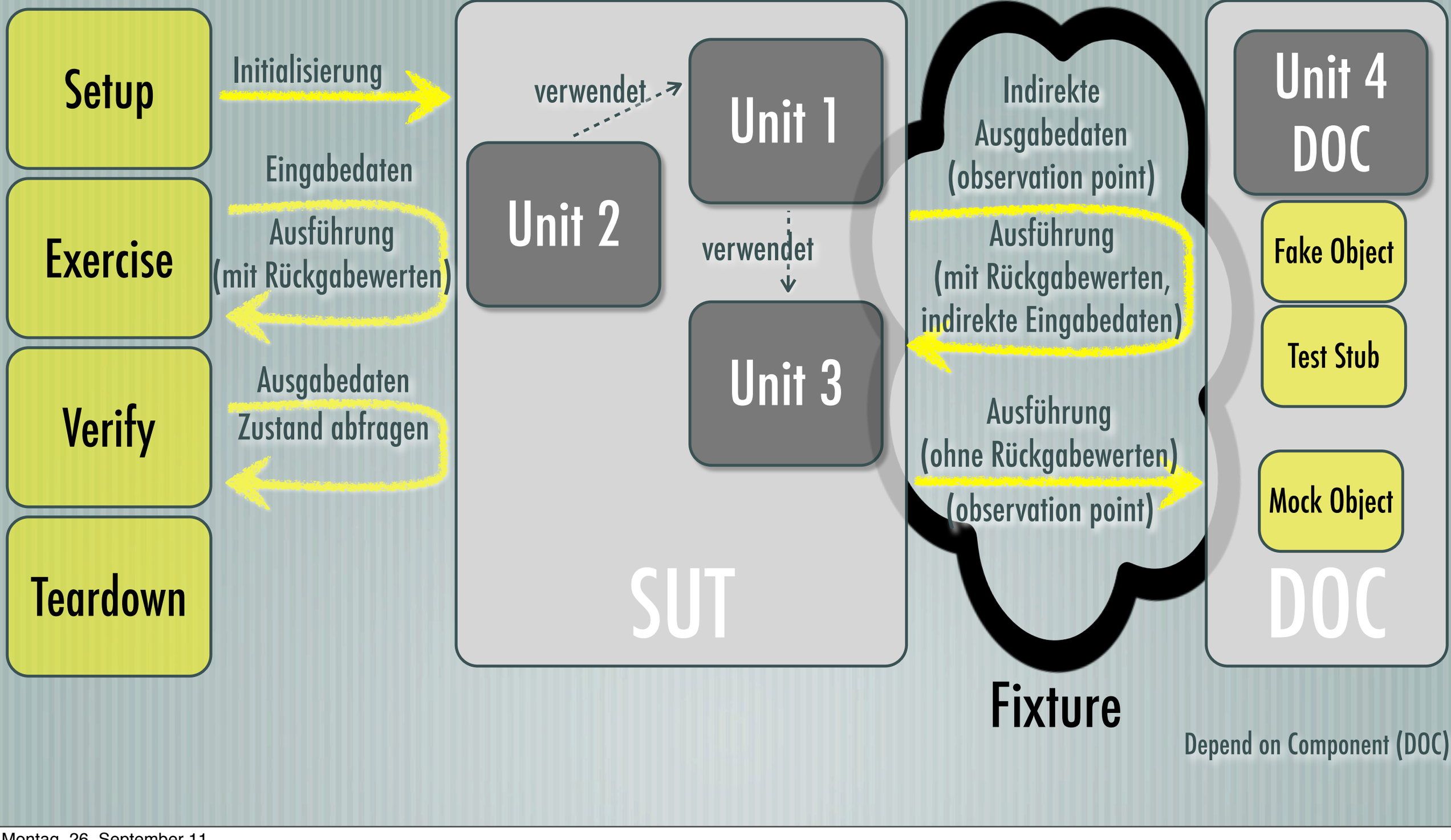
- mit xUnit bezeichnet man eine Familie von Frameworks für automatisierte Softwaretests von Modulen (Units)
- SUnit für Smalltalk (erster xUnit Vertreter entwickelt von Kent Beck ~ 1996)
- weitere Sprachen: C# (NUnit), JavaScript (JSUnit), Objective-C (OCUnit), PHP (PHPUnit),

  ABAP (ABAPUnit) ...

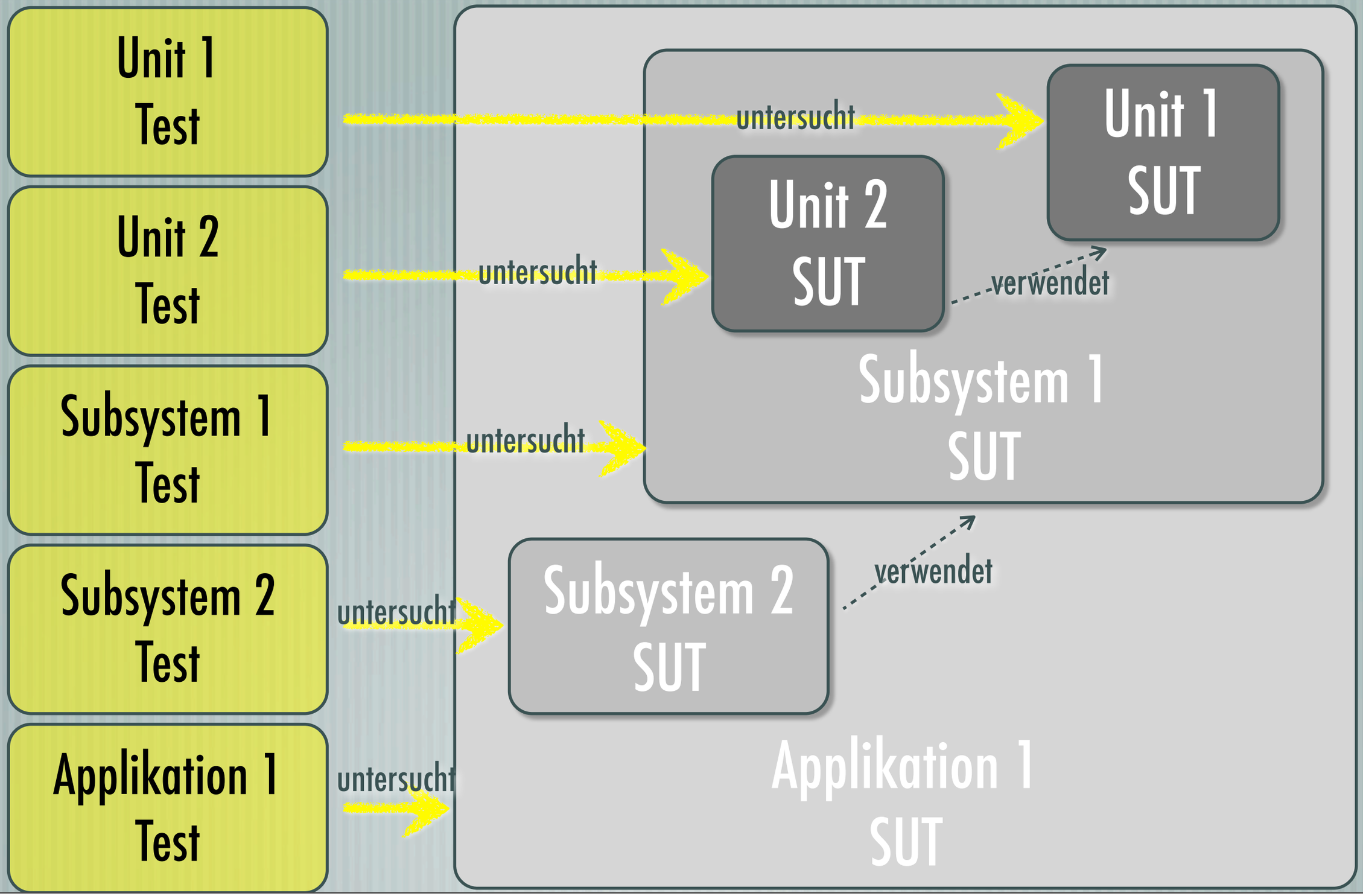
Phasen eines Unit-Tests

- Setup: das SUT wird initialisiert und für den Test in einem bestimmten Zustand vorbereitet; Ressourcen für den Test werden bereitgestellt
- Exercise: der Test interagiert mit dem SUT; es werden Eingabedaten (Input) übergeben und ein bestimmtes Verhalten aufgerufen (control points)
- Verify: der Test versucht die Effekte der Interaktion mit dem SUT herauszufinden (observation points) und mit einem definierten Erwartungswert zu vergleichen (Bedingungen)
- Teardown: das SUT wird wieder zurückgesetzt und Ressourcen freigegeben

Phasen eines Unit-Tests



System Under Test (SUT)



JUnit

- JUnit für Java (xUnit portiert von Kent Beck und Erich Gamma auf dem Flug von Zürich zur OOPSLA 1997 in Atlanta)
- Programm und Test werden in Java entwickelt
- Tests können in der Entwicklungsumgebung ausgeführt oder in den Build-Prozess integriert werden
- JUnit Version 3.8.1 für Java 1.4.x und kleiner
- JUnit Version 4.x ab Java 1.5 (nutzt Annotations)

JUnit bis Version 3.8.x

- Package junit.framework.*
- Testklasse wird von TestCase abgeleitet
- Methoden müssen mit test beginnen
- setUp() und tearDown() überschreiben
- TestCase leitet ab von Assert
- Assert bietet statische Methoden zur Überprüfung
  - static public void assertEquals(Object expected, Object actual)
- TestSuite als Sammlung von TestCases

JUnit ab Version 4

- Package org.junit.*
- Einfache Java-Klassen als Testklassen (POJOs)
- @Test gekennzeichnet Testmethoden
- Setup und Teardown-Phasen
  - @Before, @After (einmaliger Aufruf vor und nach jeder Testmethode)
  - @BeforeClass, @AfterClass (einmaliger Aufruf vor und nach dem Test)
- Zusammenfassung von einzelnen Tests zu einer Test-Suite
  - @RunWith (kennzeichnet eine TestSuite)
  - @SuiteClasses (legt fest welche Tests zur TestSuite gehören)

Beispiel

```
package com.videorentalssample.test;

import org.junit.*;

import com.videorentalssample.productioncode.Customer;
import com.videorentalssample.productioncode.Movie;
import com.videorentalssample.productioncode.Rental;

import static org.junit.Assert.*;

public class VideoStoreTest
{
      private Customer customer;

      @Before
      public void createFreshCustomerFred ()  {
            customer = new Customer ("Fred");
      }

      @Test
      public void testSingleNewReleaseStatement () {
            customer.addRental (new Rental (new Movie ("The Cell", Movie.NEW_RELEASE), 3));
            assertEquals ("Rental Record for Fred\n\tThe Cell\t9.0\nYou owed 9.0\nYou earned 2 frequent renter points\n", customer.statement ());
      }

      @Test
      public void testDualNewReleaseStatement () {
            customer.addRental (new Rental (new Movie ("The Cell", Movie.NEW_RELEASE), 3));
            customer.addRental (new Rental (new Movie ("The Tigger Movie", Movie.NEW_RELEASE), 3));
            assertEquals ("Rental Record for Fred\n\tThe Cell\t9.0\n\tThe Tigger Movie\t9.0\nYou owed 18.0\nYou earned 4 frequent renter points\n",
customer.statement ());
      }

...
```

Setup

Exercise

Verify

Teardown

z.B. customer = null;

Literatur

- http://de.wikipedia.org/wiki/Programmfehler
- http://www.junit.org
- Gerard Meszaros, xUnit Test Patterns: Refactoring Test Code, Addison-Wesley 2007