Einführung Softwaretest

Carsten Gips (HSBI)

Software-Fehler und ihre Folgen

1982: Absturz eines F117 Kampfjets: in der Softwaresteuerung Höhen- und Seitenruder vertauscht
http://de.wikipedia.org/wiki/Programmfehler

2005: TOKIOTER BÖRSE: Chef tritt ab wg. Softwarefehler (mehrere 100 Millionen Dollar Verlust)
http://www.manager-magazin.de/unternehmen/karriere/0,2828,druck-391434,00.html

2006: VW ruft 3500 Passat-Modelle zurück. Ein Softwarefehler kann zum Absterben des Motors führen.
http://auto.t-online.de/volkswagen-rueckruf-fuer-den-passat-wegen-softwarefehler/id_12821896/index

2007: Defektes Computersystem für den Tod von zehn Soldaten verantwortlich
http://www.heise.de/newsticker/meldung/Defektes-Computersystem-fuer-den-Tod-von-zehn-Soldaten-verantwortlich-186999.html?view=print

2008: Volvo-Rückruf: XC90 mit Software-Problem (Zündung vs. Klimaanlage)
http://www.auto-motor-und-sport.de/news/volvo-rueckruf-xc90-mit-software-problem-711983.html

2010: Softwarefehler in EC-Karten-Sicherheitschip: Kunden bekommen kein Geld am Automaten
http://www.tagesspiegel.de/wirtschaft/ec-karten-panne-zum-jahreswechsel/1658102.html

2010: Rückrufe kosten Toyota mehr als eine Milliarde Euro (Softwareprobleme)
http://www.spiegel.de/wirtschaft/unternehmen/0,1518,druck-675874,00.html

2010: Ford: Rückrufe wg. Softwareproblemen im Bremssystem
http://www.automobil-produktion.de/2010/02/ruckruf-jetzt-auch-ford/

2011: Honda startet umfangreiche Rückrufaktion, weltweit etwa 2,5 Millionen Autos betroffen (Softwareprobleme können zu Schäden am Getriebe führen)
http://www.handelsblatt.com/unternehmen/industrie/softwareprobleme-honda-startet-umfangreiche-rueckrufaktion/v_detail_tab_print/4470752.html

2012: Report "Softwareentwicklung 2012" (Accso): "Softwarefehler kosten die deutsche Volkswirtschaft Milliarden"
http://www.elektronikpraxis.vogel.de/index.cfm?pid=5416&pk=361330&print=true&printtype=article

Ursachen für Fehler?

Irgendjemand muss mit Deinen Bugs leben!

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. Code for readability.

– John F. Woods

Was wann testen? Wichtigste Teststufen

- **Modultest**
  - Testen einer Klasse und ihrer Methoden
  - Test auf gewünschtes Verhalten (Parameter, Schleifen, . . .)
- **Integrationstest**
  - Test des korrekten Zusammenspiels mehrerer Komponenten
  - Konzentration auf Schnittstellentests
- **Systemtest**
  - Test des kompletten Systems unter produktiven Bedingungen
  - Orientiert sich an den aufgestellten Use Cases
  - Funktionale und nichtfunktionale Anforderungen testen

=> Verweis auf Wahlfach “Softwarequalität”

JUnit: Test-Framework für Java

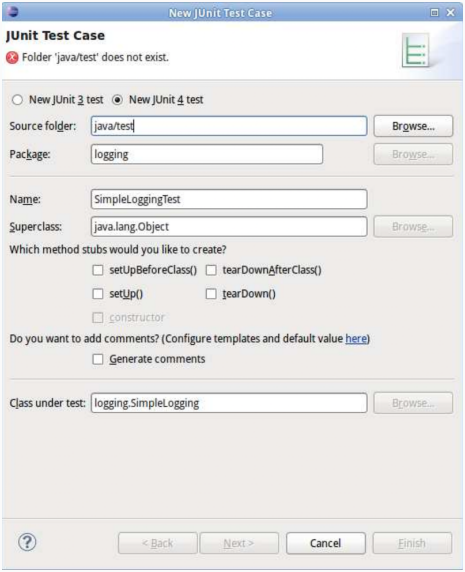
- JUnit 3
  - Tests müssen in eigenen Testklassen stehen
  - Testklassen müssen von Klasse `TestCase` erben
  - Testmethoden müssen mit dem Präfix "`test`" beginnen

- **JUnit 4**
  - Annotation `@Test` für Testmethoden
  - Kein Zwang zu spezialisierten Testklassen
  - Freie Namenswahl für Testmethoden

- *JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage*
  - Erweiterung um mächtigere Annotationen
  - Aufteilung in spezialisierte Teilprojekte

Anlegen und Organisation der Tests mit JUnit

- Anlegen neuer Tests: Klasse auswählen, Kontextmenü `New > JUnit Test Case`
- **Best Practice**:
  Spiegeln der Paket-Hierarchie
  - Toplevel-Ordner `test` (statt `src`)
  - Package-Strukturen spiegeln
  - Testklassen mit Suffix "`Test`"



JUnit 4+5: Definition von Tests

Annotation `@Test` vor Testmethode schreiben

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

public class FactoryBeispielTest4 {
    @Test
    public void testGetTicket() {
        fail("not implemented");
    }
}
```

JUnit 4: Ergebnis prüfen

Klasse `org.junit.Assert` enthält diverse **statische** Methoden zum Prüfen:

```
// Argument muss true bzw. false sein
void assertTrue(boolean);
void assertFalse(boolean);

// Gleichheit im Sinne von equals()
void assertEquals(Object, Object);

// Test sofort fehlschlagen lassen
void fail();

...
```

Anmerkung zum statischen Import

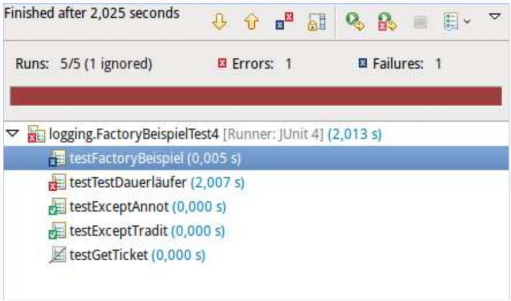
- Beispiel normaler Import:

```
import org.junit.Assert;
Assert.fail("message");
```

- Beispiel statischer Import:

```
import static org.junit.Assert.fail;
fail("message");
```

Mögliche Testausgänge bei JUnit



1. **Error**:
   - Unbehandelte Exception
   - Abbruch (Timeout)
2. **Failure**: Testausgang negativ
   - Assert fehlgeschlagen
   - `Assert.fail()` aufgerufen
3. **OK**

Anmerkung zu Asserts pro Testmethode

Wrap-Up

- Testen ist genauso wichtig wie Coden
- Richtiges Testen spart Geld, Zeit, ...
- Tests auf verschiedenen Abstraktionsstufen

- JUnit als Framework für (Unit-) Tests; hier JUnit 4 (mit Ausblick auf JUnit 5)
  - Testmethoden mit Annotation `@Test`
  - Testergebnis mit `assert*` prüfen

LICENSE

Exceptions

- Citation "*Always code as if . . .*": John F. Woods