Qualitätssicherung von Software

Prof. Dr. Holger Schlingloff

Humboldt-Universität zu Berlin
und
Fraunhofer FIRST

Bücher zum Testen

- **Myers, Glenford J.:** *The Art of Software Testing*. Wiley & Sons (1979) (auch in deutsch erhältlich: „*Methodisches Testen von Programmen*“; Neuauflage in Vorbereitung)
- **Andreas Spillner, Tilo Linz:** *Basiswissen Softwaretest.* DPUNKT Verlag (2003) (für ASQF Zertifizierung)
- **Bart Broekman, Edwin Notenboom:** *Testing Embedded Software*, Addison-Wesley (2003) (DC)
- **Harry Sneed, Mario Winter:** *Testen objektorientierter Software.* Hanser (2002)
- **Edward Kit:** *Software Testing in the Real World – Improving the process*. Addison-Wesley (1995)
- **Dorothy Graham, Mark Fewster:** *Software Test Automation - Effective Use of Test Execution Tools.* Addison-Wesley (2000)
- **Cem Kaner, Jack Falk, Hung Q. Nguyen:** *Testing Computer Software*. Wiley (2 ed. 1999)
- **Marnie L. Hutcheson:** *Software Testing Fundamentals - Methods and Metrics*. Wiley (2003)
- **Georg E. Thaller:** *Software-Test*. Heise (2002)

Kapitel 2. Testverfahren

2.1 Testen im SW-Lebenszyklus

2.2 funktionsorientierter Test

- Modul- oder Komponententest
- Integrations- und Systemtests

2.3 strukturelle Tests, Überdeckungsmaße

2.4 Test spezieller Systemklassen

- Test objektorientierter Software
- Test graphischer Oberflächen
- Test eingebetteter Realzeitsysteme

2.5 automatische Testfallgenerierung

2.6 Testmanagement und -administration

Modul- oder Komponententest

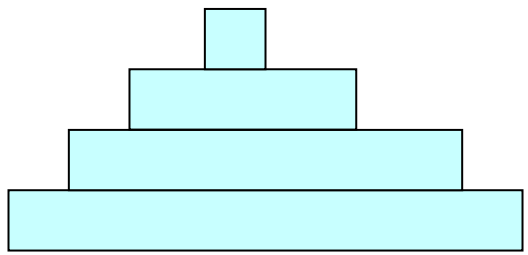
- erste Teststufe im analytischen Teil des V-Modells
- erstmaliger Test der ausführbaren Softwarebausteine nach der Programmierung
  - Prozeduren, Funktionen *(imperativ)*
  - Module, Units, Klassen, Interfaces *(objektorientiert)*
  - Blöcke (in der *modellbasierten* Entwicklung)
- Bausteine dürfen auch verschachtelt sein
- Jeder einzelne Baustein wird unabhängig von den anderen getestet
  - keine externen Einflüsse oder Wechselwirkungen
  - klare Fehlereingrenzung und -lokalisierung

Ziele des Komponententests

- Aufdeckung von Fehlern im Modul
  - falsche Berechnungen
  - fehlende Programmpfade
  - Sonderfallbehandlung
  - unzulässige Ein- und Ausgabewerte
  - Robustheit gegenüber falscher Benutzung
  - sinnvolle Fehlermeldungen bzw. Ausnahmebehandlung
- nichtfunktionale Gesichtspunkte sind sekundär
  - Effizienz, Platz- und Zeitverbrauch
  - Spezifikation und Dokumentation
  - Wartbarkeit, Änderbarkeit, ...

Vorgehensweise

- Bottom-up Ansatz
  - Start mit Klassen die von keinen anderen abhängen
  - Alle Funktionen der Klasse müssen getestet werden, alle Datenfelder verwendet und alle Zustände durchlaufen werden
  - Dann Test von Klassen die auf bereits getesteten aufbauen
  - Schichtenartige Architektursicht; Aggregation von Einzeltests in Testsuiten



eXtreme Programming

Wann soll man Modul-Tests schreiben?

- Wenn die Klasse fertig ist?
  - testen bevor andere damit konfrontiert werden
- Parallel zur Implementierung der Klasse?
  - testen um eigene Arbeit zu erleichtern
- Vor der Implementierung der Klasse!
  - Tests während Implementierung immer verfügbar
  - Konzentration auf Interface statt Implementierung
  - durch Nachdenken über Testfälle Design-Fehler finden

Ein Beispiel

```
public final class IMath {
   /*
    * Returns an integer approximation
    * to the square root of x.
    */
   public static int isqrt(int x) {
      int guess = 1;
      while (guess * guess < x) {
         guess++;
      }
      return guess;
   }
}
```

```
/** A class to test the class IMath. */
public class IMathTestNoJUnit {
   /** Runs the tests. */
   public static void main(String[] args) {
      printTestResult(0);
      printTestResult(1);
      printTestResult(2);
      printTestResult(3);
      printTestResult(4);
      printTestResult(7);
      printTestResult(9);
      printTestResult(100);
   }
  private static void printTestResult(int arg) {
      System.out.print("isqrt(" + arg + ") ==>  ");
      System.out.println(IMath.isqrt(arg));
   }
}
```

Beispiel: Yoonsik Cheon, University of Texas at El Paso, www.cs.utep.edu/~cheon/cs3331/notes/unit-testing.ppt

Fragen zum Beispiel

- Was ist die Ausgabe der Tests?
- Vorteile gegenüber manuellem Test?
- Welche Fehler werden (nicht) gefunden?
- Probleme mit dieser Art zu testen?
- Was kann verbessert werden?

JUnit (vgl. Übung!)

```
import junit.framework.*;
public class IMathTest extends TestCase {
  public void testIsqrt() {
    assertEquals(0, IMath.isqrt(0));
    assertEquals(1, IMath.isqrt(1));
    …
    assertEquals(10, IMath.isqrt(100));
  }
  public static Test suite() {
    return new TestSuite(IMathTest.class);
  }
public static void main (String[] args) {
    junit.textui.TestRunner.run(suite());
  }
}
```

- Kontrollierte Testausführung und -auswertung
- Public domain (www.junit.org)
- sofort einsetzbar, in viele IDEs integriert
- unterstützt „Test durch Entwickler“ Paradigma
- Testautomatisierung!

JUnit (Forts.)

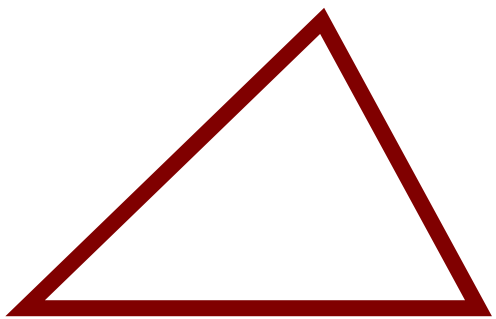
- Typisch für eine Reihe ähnlicher Tools
- Vorteile der Verwendung
  - automatisierte, wiederholbare Tests (nach jedem „Build“!)
  - kombinierbar zu Testklassen und Testsuiten
  - Einbeziehung von Testdaten
  - automatische Testauswertung
  - Möglichkeit des Tests von Ausnahmen
  - Möglichkeit der Verwendung interner Schnittstellen
- Was JUnit dem Tester nicht abnimmt
  - Testentwurf (Auswahl und Beurteilung der Testfälle)

Komponententest-Entwurf

- sehr entwicklungsnahe Testarbeit, oftmals direkt am ausführbaren Code
  - Problematik der Spezifikation (Wozu eine Spezifikation, wenn der Code selbst vorliegt?)
- fließender Übergang zum Debugging; oft von den Entwicklern selbst durchgeführt
  - Problematik der Zielsetzung (Fehler finden oder Ablauffähigkeit demonstrieren?)
  - Problematik des Hintergrundwissens aus der Entwicklung (für den Testentwurf oft notwendig, aber stillschweigende Annahmen behindern die Objektivität bzw. Abdeckung)

Ein “klassisches” Beispiel (Myers)

- Funktion mit drei int-Eingabewerten (x,y,z), die als Längen von Dreiecksseiten interpretiert werden
- Berechnet, ob das Dreieck gleichseitig, gleichschenklig oder ungleichseitig ist



- Schreiben Sie für diese Funktion Testfälle auf! (jetzt!)

Auswertung nach Punkten (1)

1. Haben Sie einen Testfall für ein zulässiges gleichseitiges Dreieck?
2. Haben Sie einen Testfall für ein zulässiges gleichschenkliges Dreieck? (Ein Testfall mit 2,2,4 zählt nicht.)
3. Haben Sie einen Testfall für ein zulässiges ungleichseitiges Dreieck? (Beachten Sie, dass Testfälle mit 1,2,3 und 2,5,10 keine Ja-Antwort garantieren, da kein Dreieck mit solchen Seiten existiert.)
4. Haben Sie wenigstens drei Testfälle für zulässige, gleichschenklige Dreiecke, wobei Sie alle drei Permutationen der beiden gleichen Seiten berücksichtigt haben? (z.B. 3,3,4; 3,4,3; 4,3,3)

Auswertung nach Punkten (2)

5. Haben Sie einen Testfall, bei dem eine Seite gleich Null ist?
6. Haben Sie einen Testfall, bei dem eine Seite einen negativen Wert hat?
7. Haben Sie einen Testfall mit 3 ganzzahligen Werten, in dem die Summe zweier Zahlen gleich der dritten ist? (D.h., wenn das Programm 1,2,3 als ungleichseitiges Dreieck akzeptiert, so enthält es einen Fehler.)
8. Haben Sie wenigstens drei Testfälle für Punkt 7, wobei Sie alle drei Permutationen für die Länge jeweils einer Seite als Summe der beiden anderen Seiten berücksichtigt haben? (z.B. 1,2,3; 1,3,2; 3,1,2.)
9. Haben Sie einen Testfall mit drei ganzzahligen Werten größer Null, bei dem die Summe aus zwei Zahlen kleiner als die dritte ist? (z.B. 1,2,4 oder 12,15,30)

Auswertung nach Punkten (3)

10. Haben Sie wenigstens drei Testfälle für Punkt 9, wobei Sie alle drei Permutationen berücksichtigt haben? (z.B. 1,2,4; 1,4,2; 4,1,2.)
11. Haben Sie einen Testfall, in dem alle drei Seiten gleich Null sind (d.h. 0,0,0)?
12. Haben Sie wenigstens einen Testfall mit nichtganzzahligen Werten?
13. Haben Sie wenigstens einen Testfall, in dem Sie eine falsche Anzahl von Werten angeben (z.B. zwei statt drei ganzzahlige Werte)?
14. Haben Sie zusätzlich zu jedem Eingangswert in allen Testfällen die erwarteten Ausgabewerte angegeben?

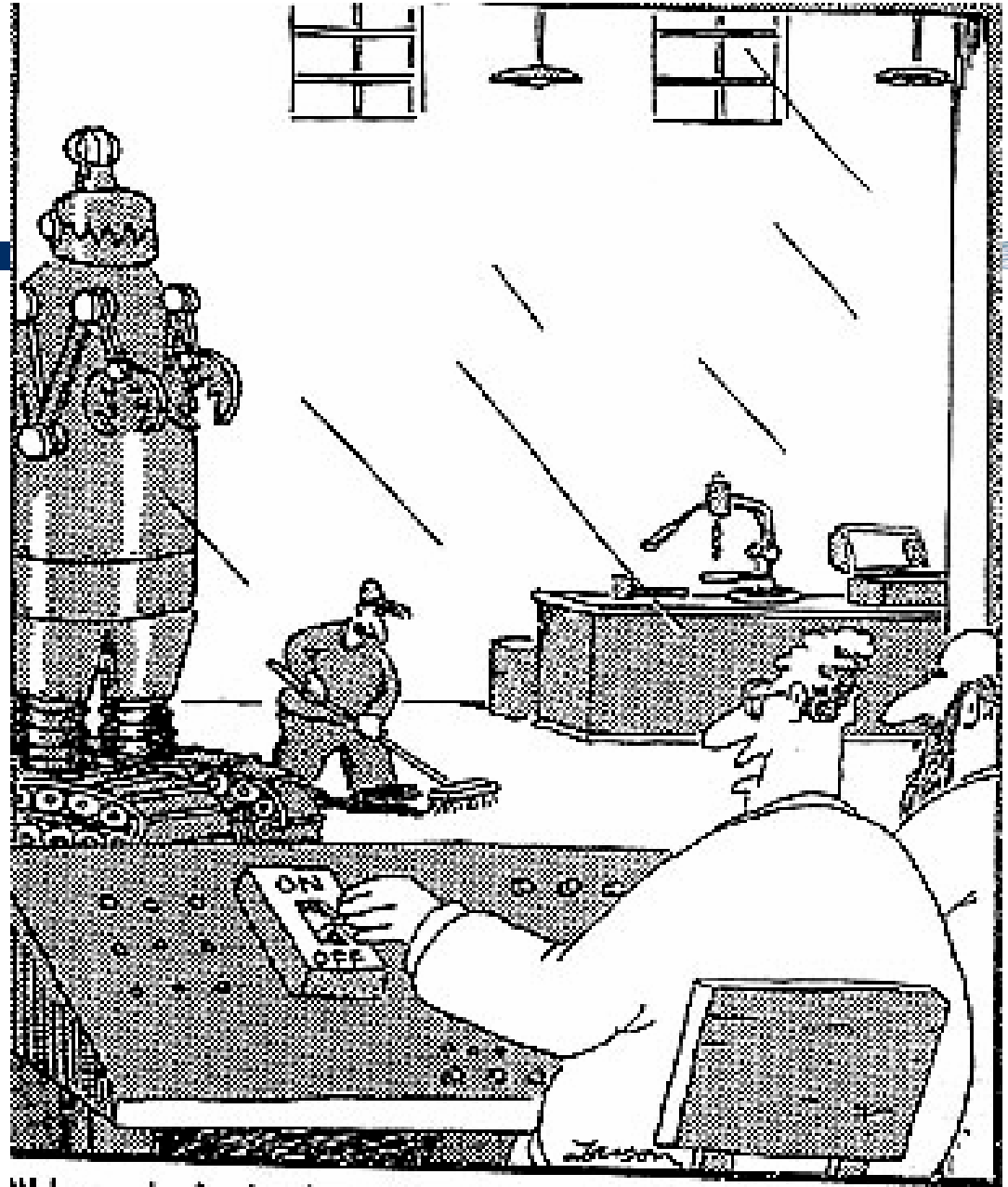
Zusatzpunkte

15. Test mit maximalen Werten
16. Test auf Zahlbereichs-Überlaufbehandlung
17. Test mit unzulässigen Eingabezeichenfolgen

Reflexion

- Durchschnittswerte erfahrener Programmierer: 7-8
- *„Diese Übung sollte zeigen, dass das Testen auch eines solch trivialen Programms keine leichte Aufgabe ist. Und wenn das wahr ist, betrachten Sie die Schwierigkeit, ein Flugleitsystem mit 100.000 Befehlen, einen Compiler oder auch nur ein gängiges Gehaltsabrechnungsprogramm zu testen.“* (1979)
- Heute: 1-10 MLoC

Pause!



"Hey, who's that? ... Oh—Mitch, the janitor. Well, our first test run has just gotten a little more interesting."

Testentwurf im Komponententest

- Aus Komplexitätsgründen ist es nicht möglich, alle Eingabewerte(-folgen) zu testen
- Problem: Welche Untermenge aller denkbaren Testfälle bietet die größte Wahrscheinlichkeit, möglichst viele Fehler zu finden?

➔Techniken zur Testdaten- und Testfallbestimmung

- *Äquivalenzklassenbildung*
  - Auswahl „repräsentativer“ Daten
- *Grenzwertanalyse*
  - Wertebereiche und Bereichsgrenzen
- *Entscheidungstabellen* und *Klassifikationsbäume*

Äquivalenzklassenbildung

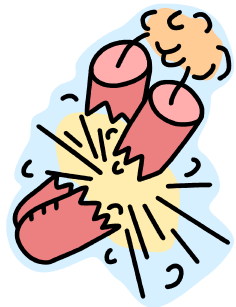
- **1. Schritt:** Partitionierung des Eingabedatenraumes in eine endliche Zahl von Äquivalenzklassen (bezüglich des vermuteten Ausfallverhaltens)
  - im Beispiel: „drei gleiche Eingaben größer Null“
- **2. Schritt:** Auswahl der Testfälle anhand je eines Repräsentanten der Äquivalenzklasse
  - im Beispiel: (2,2,2)

Vorgehensweise zur Äquivalenzklassenbildung

- Betrachten der Definitionsbereiche für Ein-/Ausgabewerte
- Für jeden Wert ergeben sich gültige und ungültige Klassen
  - Wertebereiche, Aufzählungen: enthalten oder nicht enthalten
  - Eingabewerte, die (möglicherweise) unterschiedlich verarbeitet werden: für jeden Wert eine gültige und insgesamt eine ungültige Klasse
  - Ausgaben, die auf verschiedene Weise berechnet werden: je eine Klasse, die auf diese Ausgabe führt
  - Eingabebedingungen, die vorausgesetzt werden: je eine gültige und eine ungültige Klasse
- Aufspaltung einer Klasse in kleinere Klassen, falls Grund zur Annahme besteht, dass nicht alle Elemente gleich behandelt werden
- Tabellierung der zu jedem Parameter gehörigen Klassen

Vorgehensweise zur Testfallauswahl

	Äq1	Äq2	Äq3	...
Par1	Wert1.1	Wert1.2		
Par2	Wert2.1	...		
...				
ParN				



- Vollständig: Kartesisches Produkt der Klassen
  - meist nicht praktikabel
- Heuristisch: Auswahl gemäß folgender Strategie
  - Bildung von Testfällen, die möglichst viele noch nicht behandelte gültige Klassen abdecken
  - Bildung von Testfällen, die genau eine ungültige Klasse abdecken
  - Paarweise Kombination von Repräsentanten

im Beispiel: (2,2,3) und (-7,1,2), (5,“a“,2)