SE Besprechung

Übung 6 – Softwaretests

SE, 15.12.09

Dustin Wüest

Tutoren für Informatik IIa – Modellierung

- Voraussetzung
  - Informatik IIa (oder vergleichbar) erfolgreich besucht
- Aufgaben
  - Korrektur der Aufgaben (6 Übungen)
  - Übungsbesprechung (2 Gruppen)
  - Prüfungsaufsicht an den Zwischentests (3 Tests)
- Lohn
  - Geld
  - 2 Punkte ECTS
  - Erfahrung in der Lehre / Vertiefung in Modellierung von Software
- Bei Interesse: wueest@ifi.uzh.ch

TA für Informatik IIa – Modellierung

- Voraussetzung
  - Informatik IIa (oder vergleichbar) erfolgreich besucht
  - Masterstudium (evt. Bachelor höheren Semesters)
- Aufgaben
  - Aufgabenstellungen (6 Übungen)
  - Zwischentests erstellen (3 Tests)
  - Mithilfe bei Korrektur der Zwischentests
  - Mithilfe bei Vorbereitung der Klausur, evt. Prüfungsaufsicht
- Lohn
  - Geld
  - 4 Punkte ECTS
  - Erfahrung in der Lehre / Vertiefung in Modellierung von Software

Studierende nach PPO 2001

- Bitte so bald wie möglich bei mir **melden**
- Bei geringer Anzahl an Studierenden nach PPO 2001 findet für diese eine normale Schlussklausur und eine **zusätzliche mündliche Prüfung** statt
- Bei grösserer Anzahl an Studierenden nach PPO 2001 findet für diese eine **erweiterte Schlussklausur** statt

Prüfung

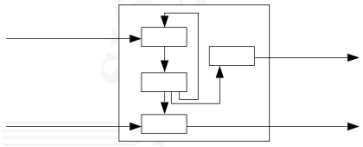
- Inhalt
  - ca. 1/3 Multiple Choice
  - Anwendungsfragen
    z.T. ähnlich wie Übungen und Mini-Übungen
    zum Stoff der Vorlesung
- Ort, Dauer
  - Dienstag, 12. Januar 10.15h im 0.K.02
  - 90 Minuten (120 Punkte)
- Hilfsmittel
  - 1 Blatt A4 handgeschrieben
  - (Wörterbuch)

SE Besprechung

Übung 6 – Softwaretests

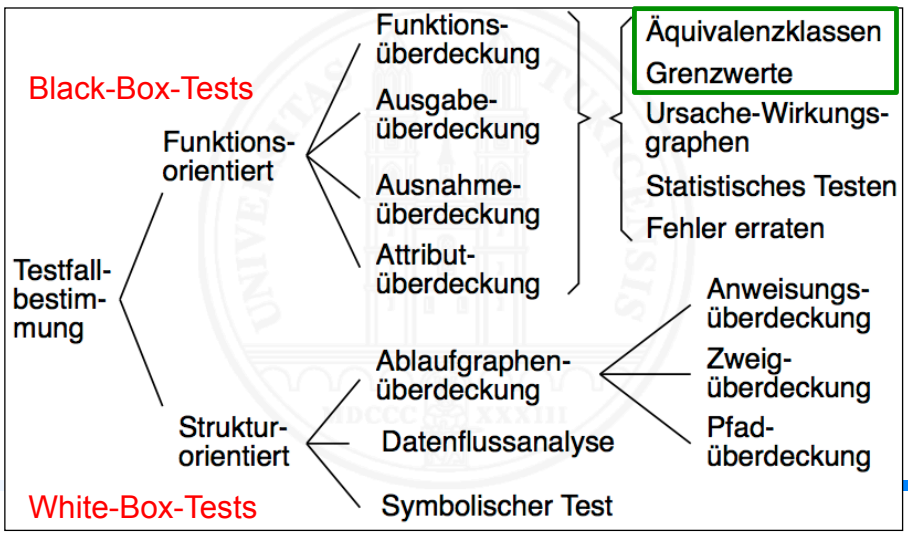
Aufgabe 6.1 – Testen

- a), b) → Black-Box-Test
- c), d), e) → White-Box



Aufgabe 6.1 – Testen

- Ziel beim Auswählen von Testfällen:
  mit möglichst wenig Testfällen möglichst viele Fehler finden



Aufgabe 6.1 – Testen, a)

- Sinn der Äquivalenzklassen
  - Methode kann nicht mit allen möglichen Werten getestet werden
  - Deshalb: gute Auswahl von Testfällen
  - Ziel: Werte, welche ein gleiches / ähnliches Verhalten der Methode bewirken, zu einer Klasse zusammenfassen

    ABER: Black-Box-Test, Code wird nicht angeschaut

    → Ähnliche Eingabewerte (oder Ausgabewerte) zu Klassen zusammenfassen
  - Methode wird “stichprobenweise” getestet: Von jeder Klasse wird ein (oder evtl. ein paar) Wert(e) zum Testen ausgewählt

Aufgabe 6.1 – Testen, a)

- Äquivalenzklassen wählen
  - Jeder mögliche Input gehört zu genau einer Äquivalenzklasse
  - Äquivalenzklassen sind disjunkt
  - Gleichartige Inputs / Outputs zu Klassen gruppieren

  Methodenkommentar und Signatur

  → Nimmt einen Float als Input und rechnet *modulo 3*

  → (Rückgabewert ist immer >= 0)
    - Interpretation 1: -4 % 3 = 3 – (4 % 3) = 2
    - Interpretation 2: -4 % 3 = 4 % 3 = 1

Aufgabe 6.1 – Testen, a)

Methodenkommentar und Signatur
→Nimmt einen Float als Input und rechnet *modulo 3*

Mögliche Äquivalenzklassen:

I) Positive Zahlen zwischen 0 und 3 (3 > x >= 0)

II) Positive Zahlen grösser 3, aber kein Vielfaches (x = 4.25, 7, ...)

III) Positives Vielfaches von 3 (x = 3, 6, 9, ...)

IV) Negative Zahlen zwischen 0 und -3 (-3 < x < 0)

V) Negative Zahlen kleiner als -3, kein Vielfaches (x = -4.25, -7, ...)

VI) Negatives Vielfaches von 3 (x = -3, -6, -9, ...)

Aufgabe 6.1 – Testen, a)

- Grenzfälle wählen
  - An den Grenzen zulässiger Datenbereiche
  - An den Rändern der Äquivalenzklassen

Mögliche Grenzfälle:

I) x = 0

II) Sehr grosse positive / negative Zahl
(x = Float.MAX_VALUE (oder evtl. x = +/- 60))

III) Zahl nahe 0, nahe +/- 3 (x = 0.001, 2.999, 3.001, -3.001, ...)

(Ungültige Eingaben, z.B. String → Code kompiliert nicht)

Aufgabe 6.1 – Testen, a)

- Testvorschrift

1. Einleitung

1.1 Zweck
Art und Zweck des im Dokument beschriebenen Tests

1.2 Testumfang
Welche Konfigurations-Einheiten der entwickelten Lösung getestet werden

1.3 Referenzierte Unterlagen
Verzeichnis aller Unterlagen, auf die im Dokument Bezug genommen wird

2. Testumgebung

2.1 Überblick
Testgliederung, Testgüte, Annahmen und Hinweise

2.2 Testmittel
Test-Software und -Hardware, Betriebssystem, Testgeschirr, Werkzeuge

2.3 Testdaten, Testdatenbank
Wo die für den Test benötigten Daten bereit liegen oder bereitzustellen sind

2.4 Personalbedarf
wieviel Personen zur Testdurchführung benötigt werden

3. Annahmekriterien
Kriterien für
- erfolgreichen Test-Abschluss
- Test-Abbruch
- Unterbrechung und Wiederaufnahme des Tests

4. Testfälle

Aufgabe 6.1 – Testen, b)

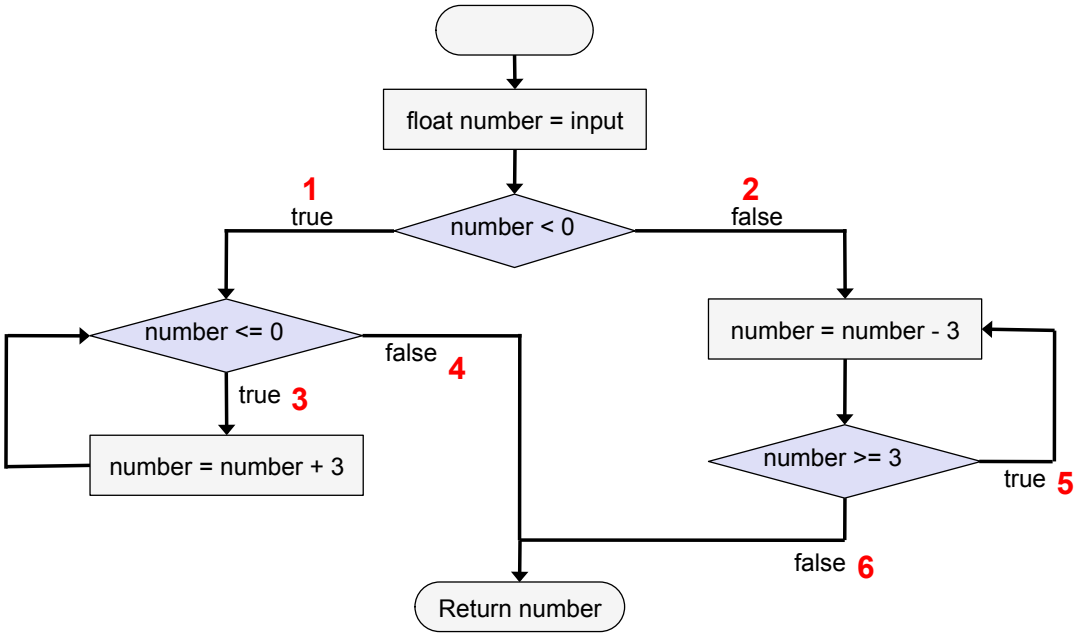
Testfall	Eingabe	Erwartetes Resultat	Befund
1 (K I)	2	2	-1
2 (K II)	4.5	1.5	1.5
3.1 (K III)	3	0	0
3.2 (K III)	9	0	0
4 (K IV)	-2	1	1
5 (K V)	-7.3	1.7	1.7
6 (K VI)	-6	0	3
7 (G)	0	0	-3
8 (G)	123’456’788	2	2*
9 (G)	0.0001	0.0001	-3

* Endlosschleife bei Test in Eclipse

Aufgabe 6.1 – Testen, b)

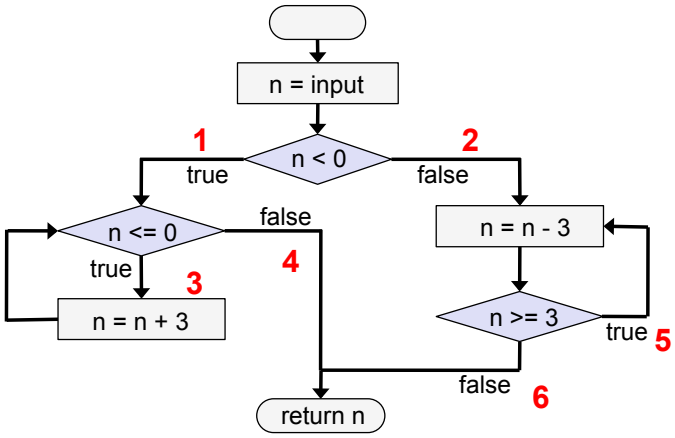
- Falsche Resultate für folgende Fälle (war in der Aufgabe *nicht* verlangt)
  - 0 <= x < 3
  - x negativ und Vielfaches von 3
  - für zu grosse Werte (pos. und neg.)
  - (falls -x % 3 = x % 3, dann für alle negativen Werte falsch, ausser für -1.5, -4.5, ...)

Aufgabe 6.1 – Testen, c)



Aufgabe 6.1 – Testen, c)

- 6 Zweige
- Mein Test: 100% Zweigüberdeckung



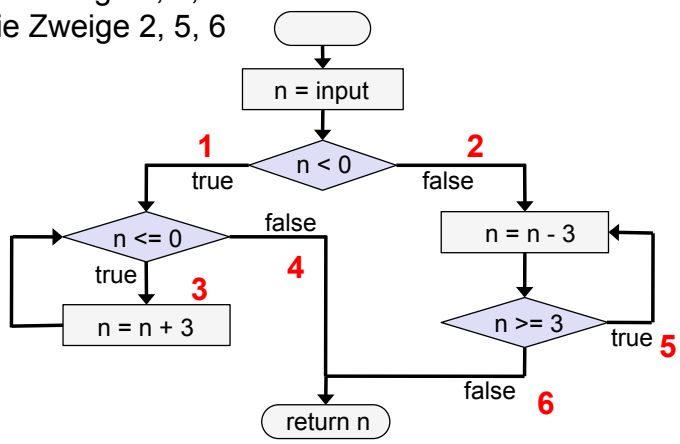
Aufgabe 6.1 – Testen, d)

- Wieviele Testfälle mind. für 100% Zweigüberdeckung?

2 Testfälle:
input < 0 → durchläuft die Zweige 1, 3, 4
input >= 6 → durchläuft die Zweige 2, 5, 6

Achtung: bei 0 <= input < 6 wird Zweig **5** nicht durchlaufen! (*do...while*)

Zweigüberdeckung vs. Anweisungsüberdeckung vs. Pfadüberdeckung



Aufgabe 6.1 – Testen, e)

Testfall	Eingabe	Erwartetes Resultat	Befund
1 (K I)	2	2	-1
2 (K II)	4.5	1.5	1.5
3.1 (K III)	3	0	0
3.2 (K III)	9	0	0
4 (K IV)	-2	1	1
5 (K V)	-7.3	1.7	1.7
6 (K VI)	-6	0	3
7 (G)	0	0	-3
8 (G)	123'456'788	2	2*
9 (G)	0.0001	0.0001	-3

"ohne Fehler" = nur positive Befunde

input < 0 z.B. input = –7.3
input >= 6 z.B. input = 9

Testfall	Eingabe	Erwartetes Resultat	Befund
1 (K I)	2	2	-1
2 (K II)	4.5	1.5	1.5
3.1 (K III)	3	0	0
3.2 (K III)	9	0	0
4 (K IV)	-2	2	1
5 (K V)	-7.3	1.3	1.7
6 (K VI)	-6	0	3
7 (G)	0	0	-3
8 (G)	123'456'788	2	2*
9 (G)	0.0001	0.0001	-3

input < 0 z.B. input = –1.5
input >= 6 z.B. input = 9

Aufgabe 6.1 – Testen, f)

- *Welche Fehlerarten werden beim White-Box-Test leichter gefunden?*
  - "dead code" entdecken (Anweisungsüberdeckung)
  - Programmierfehler, falsch implementierte Funktionalität (Lokalisierung von Defekten)
- *Welche Fehlerarten werden beim Black-Box-Test leichter gefunden?*
  - Fehler im Gesamtsystem (Integrationstests, Interaktion zw. Komponenten)
  - Fehler in der Spezifikation / Interfaces / Kommentaren
- Tests sind üblicherweise nicht pure Black-/White-Box-Tests, sondern irgendwo dazwischen

Aufgabe 6.2 – Messen: Faktoren, Fragen

„Einfaches und schnelles Reservieren und Einkaufen von Theaterkarten“

- Faktoren
  - Klarheit des GUIs
    - Weiss der Benutzer, was er gerade macht?
  - Übersichtlichkeit des GUIs
    - Wie schnell kann man Karten reservieren / kaufen?
    - Wie einfach kann man Karten reservieren / kaufen?
  - Anzahl auftauchender Probleme
    - Wieviele Bestellvorgänge werden abgebrochen?

Aufgabe 6.2 – Messen: Messbare Merkmale

Weiss der Benutzer, was er gerade macht?

- Hilfestellungen durch das System vorhanden (Nominalskala: ja/nein)

Wie schnell kann man Karten reservieren / kaufen?

- Zeit vom Beginn bis zum Reservationsabschluss (Verhältnisskala)
- Antwortzeiten des Servers (Verhältnisskala)

Wie einfach kann man Karten reservieren / kaufen?

- Anzahl Klicks bis zum Reservationsabschluss (Absolutskala)
- Kundenzufriedenheit (Ordinalskala: --, -, ~, +, ++)

Wieviele Bestellvorgänge wurden abgebrochen?

- Anzahl (Absolutskala)

Aufgabe 6.2 – Messen

Nominalskala (ja / nein / n.a.)

Ordinalskala (--, -, ~, +, ++)

Intervallskala (z.B. Datumskala: Distanzen bestimmbar, aber keine Multiplikation etc. möglich)

Verhältnisskala (z.B. # Code-Zeilen für Programmgrösse)

Absolutskala (z.B. # Code-Zeilen, z.B. Anzahl Einwohner)

Absolutskala: Keine Transformation möglich (f(x) = ax mit a = 1)

→ "...können nicht halbe Einwohner zählen und * 2 rechnen..."