

SE Besprechung

Übung 6 – Softwaretests

Tutoren für Informatik IIa – Modellierung

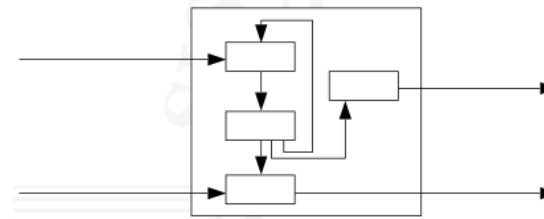
- Voraussetzung
 - Informatik IIa (oder vergleichbar) erfolgreich besucht
- Aufgaben
 - Korrektur der Aufgaben (6 Übungen)
 - Übungsbesprechung (2 Gruppen)
 - Prüfungsaufsicht an den Zwischentests (3 Tests)
- Lohn
 - Geld
 - 2 Punkte ECTS
 - Erfahrung in der Lehre / Vertiefung in Modellierung von Software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

Prüfung

- Inhalt
 - ca. 1/3 Multiple Choice
 - Anwendungsfragen
 - z.T. ähnlich wie Übungen und Mini-Übungen
 - zum Stoff der Vorlesung
- Ort, Dauer
 - Dienstag, 18. Januar 2010 um 10.15h
 - 90 Minuten (120 Punkte)
- Hilfsmittel
 - 1 Blatt A4 handgeschrieben
 - (Wörterbuch)

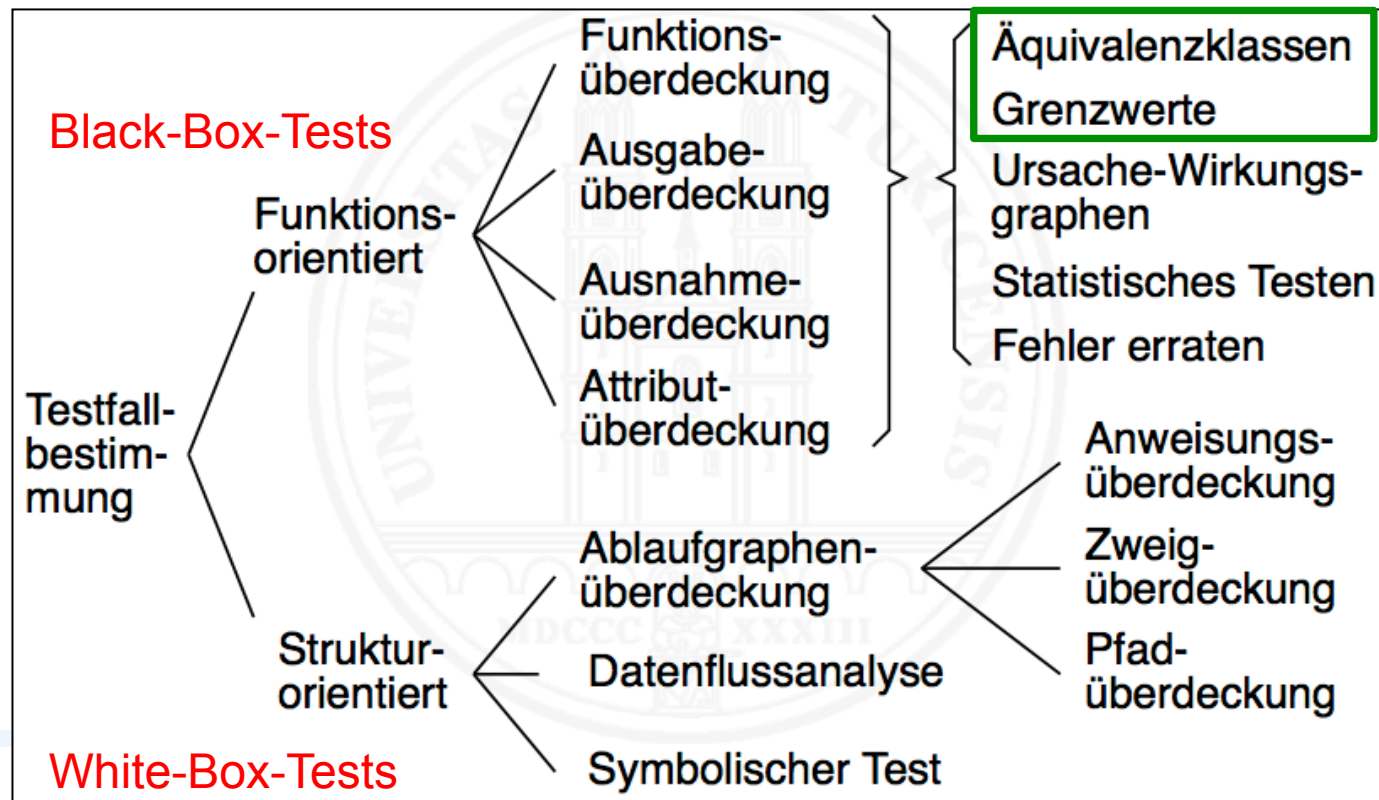
Aufgabe 6.1 – Testen

- 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

- Prüft einen String darauf, ob es sich bei ihm um einen gültigen UPC/EAN Barcode handelt
- Rückgabewert ist true oder false

Aufgabe 6.1 – Testen, a)

Methodenkommentar und Signatur

→ Prüft einen String darauf, ob es sich bei ihm um einen gültigen UPC/EAN Barcode handelt

Mögliche Äquivalenzklassen:

- 1) Ungültiges Format
- 2) Gültiges Format
 - 2.1) Falsche Länge
 - 2.1.1) Falsche Kennziffer
 - 2.1.2) Gültige Kennziffer
 - 2.2) Gültige Länge
 - 2.2.1) Falsche Kennziffer
 - 2.2.2) Gültige Kennziffer

Aufgabe 6.1 – Testen, a)

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

Mögliche Grenzfälle:

Erlaube Werte der Länge:

8

12

13

14

Grenzfälle:

7

8

9

11

12

14

15

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

wieviele Personen zur Testdurchführung benötigt werden

3. Annahmekriterien

Kriterien für

- erfolgreichen Test-Abschluss
- Test-Abbruch
- Unterbrechung und Wiederaufnahme des Tests

4. Testfälle

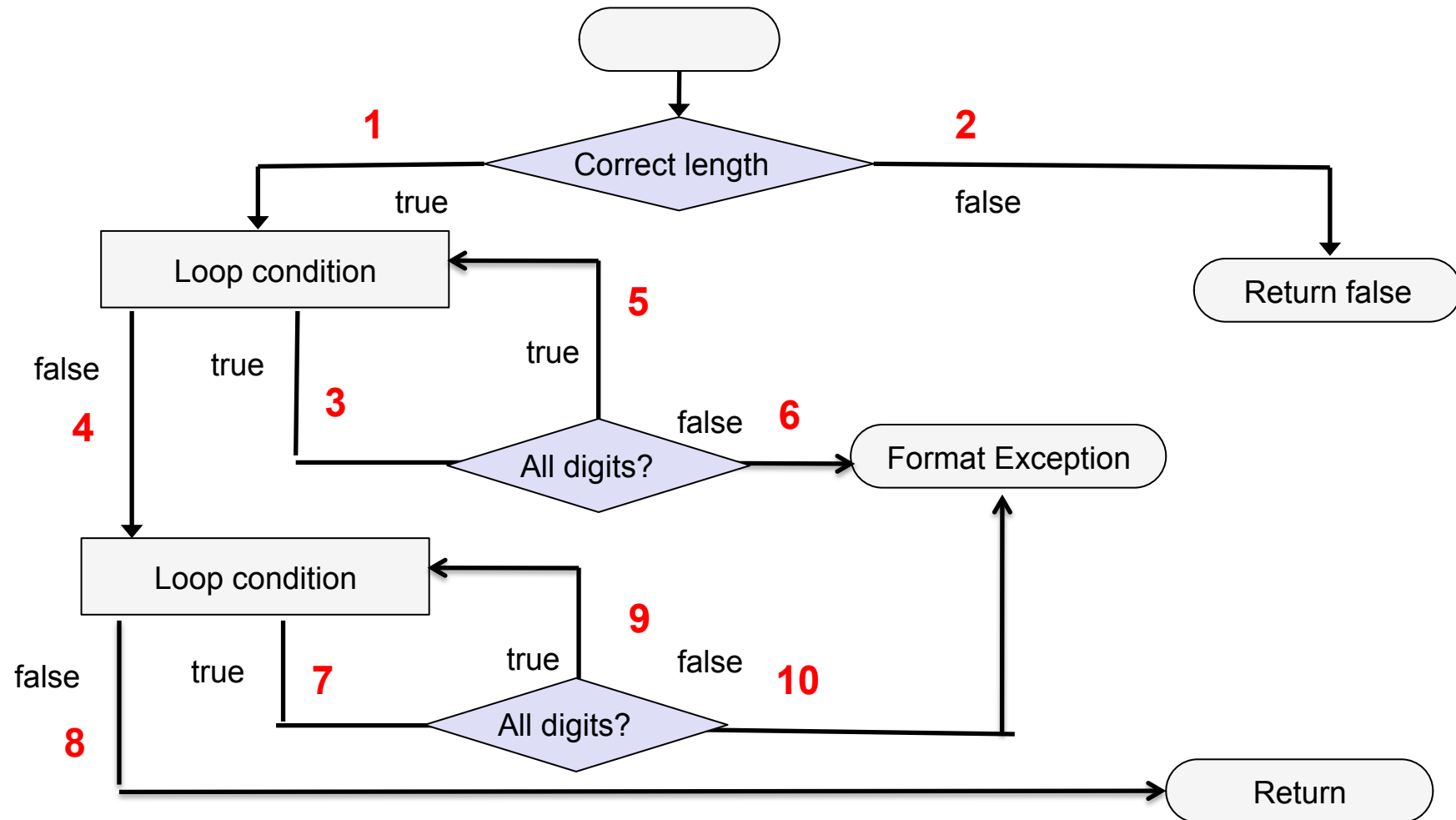
Aufgabe 6.1 – Testen, b)

Test nr.	Parameter	Expected result	Actual result
1.1 check for a bar code of length 8.	„11111115“	TRUE	TRUE
2.1 check for a bar code of length 8.	„11111112“	FALSE	FALSE
3.4 String of length 11.	„12345678905“	FALSE	TRUE
4.1.2 String of length 8 with a non-digit character at the end.	„3234428b“	Exception("Format Exception")	FALSE

Aufgabe 6.1 – Testen, b)

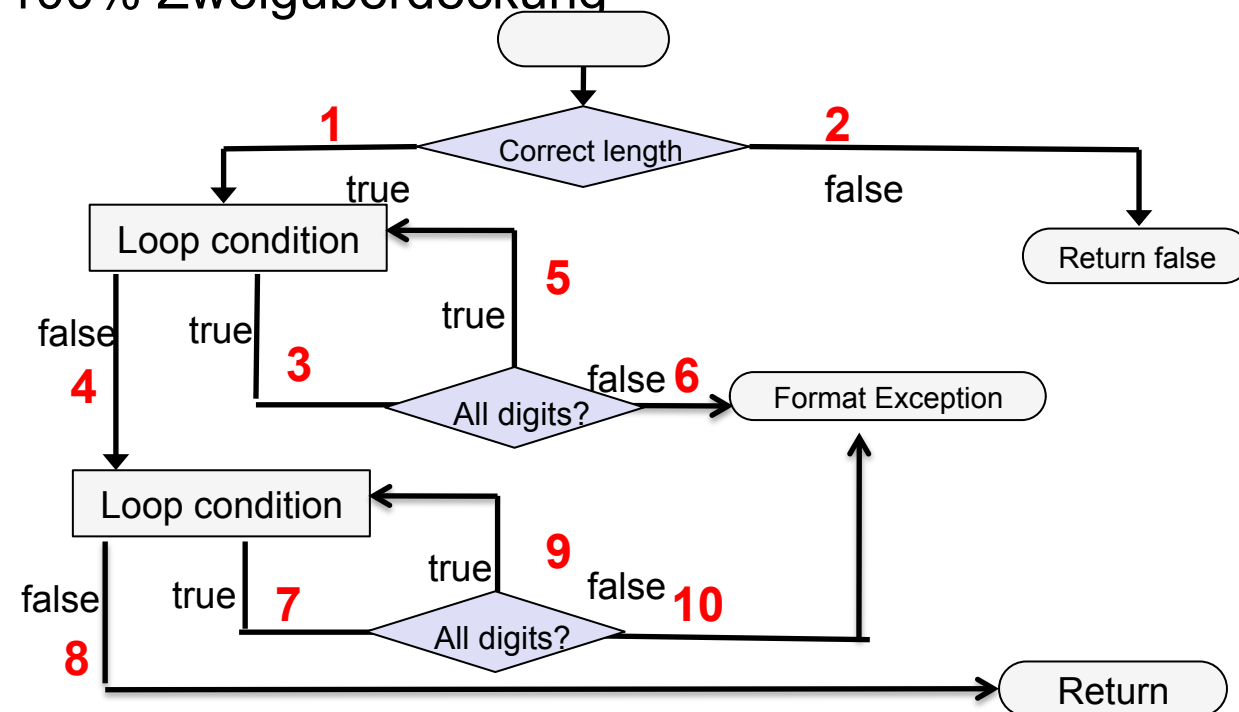
- Falsche Resultate für folgende Fälle (war in der Aufgabe *nicht* verlangt)
 - Barcode hat Länge 11 und eine gültige Prüfziffer
 - Ausgabe **true**, erwartet wurde **false**
 - Barcode hat Buchstabe oder Sonderzeichen an letzter Stelle
 - Ausgabe **false**, erwartet wurde **FormatException**

Aufgabe 6.1 – Testen, c)



Aufgabe 6.1 – Testen, c)

- 10 Zweige
- Mein Test: 100% Zweigüberdeckung

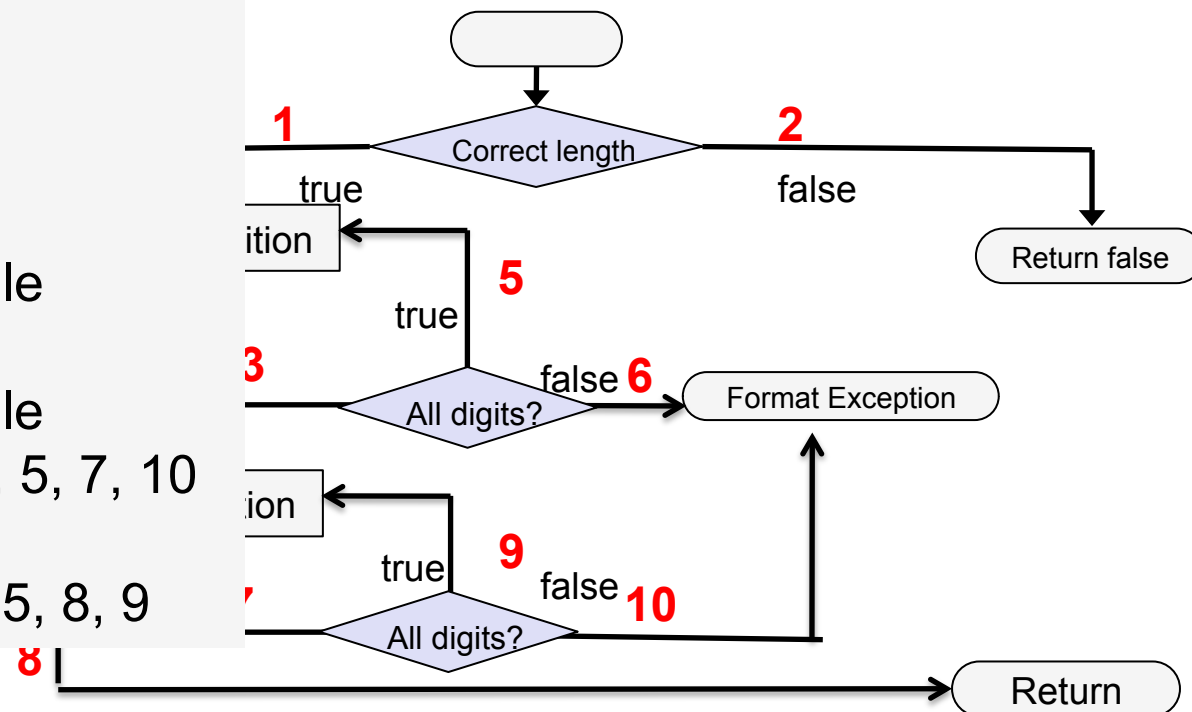


Aufgabe 6.1 – Testen, d)

- Wieviele Testfälle für 100% Zweigüberdeckung?

4 Testfälle:

- String von Länge 4
--> Zweig 2
- Buchstabe an 1. Stelle
--> Zweige 1, 3, 6
- Buchstabe an 2. Stelle
--> Zweige 1, 3, 4, 5, 7, 10
- Korrekter Code
-> Zweige 1, 3, 4, 5, 8, 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

„Einfache und effiziente Handhabung medizinischer Daten“

- Faktoren

- Klarheit des GUIs

- Weiss der Benutzer, was er gerade macht?

- Übersichtlichkeit des GUIs

- Wie schnell kann man Daten erfassen/abrufen?

- Wie einfach kann man Daten erfassen/abrufen?

- Anzahl auftauchender Probleme

- Wieviele Vorgänge 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 Daten erfassen / abrufen?

Zeit vom Beginn bis zum Abschluss der Erfassung (Verhältnisskala)

Antwortzeiten des Servers (Verhältnisskala)

Wie einfach kann man Daten erfassen / abrufen?

Anzahl Klicks bis zur Erfassung (Absolutskala)

Mitarbeiterzufriedenheit (Ordinalskala: --, -, ~, +, ++)

Wieviele Vorgänge 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..."