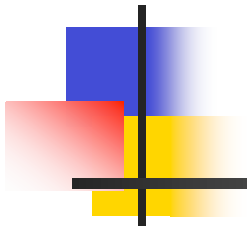
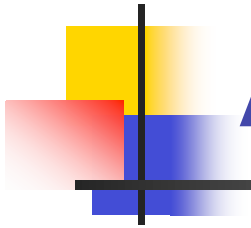


# Software Engineering

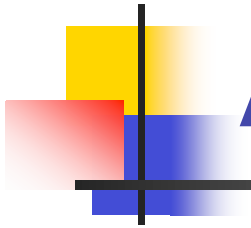
Besprechung zur Uebung 6

Softwaretests



# Allgemeines, Formelles

- Für alle Gruppen
  - Abholung der Übungen: BIN 2.B.17
- Abschluss der Übungen - Gesamtpunkteliste
  - Bitte kontrolliert die Punkteliste auf Korrektheit
- Prüfung
  - Findet am 15. Jänner ab 10:15 Uhr im Hörsaal 1.B.01 statt
  - Eine Musterprüfung zur Vorbereitung ist am Web
- Prüfung, PPO 01
  - Studierende nach PPO 01 belegen eine erweitere Schlussklausur
  - Klausur ist nicht länger, nur Aufgaben sind teilweise anders
  - Eine Zusatzübung + Lösung zur Vorbereitung ist am Web



# Aufgabe 2.1: Testen (1)

- Ziel
  - Das Programm mit der Absicht ausführen Fehler zu finden
- Aufgabe
  - Die Aufgabenstellung war etwas schwierig, da Bedingung der do-while Schleife sehr falsch
- Kriterien
  - Unterscheidung Black-Box Test (Aufgabe a) und White-Box (Rest)
  - Punkteverteilung: a) i.&ii. 3P, iii. 3P, b) 2P, c) 1,5P, d) 1,5P, e) 2P.
  - Genaues Aufzeigen des Fehlers und Korrektur war in der Aufgabe nicht explizit gefordert ...

# Aufgabe 2.1: Testen (2)

- Angabe

Kontext:
- Felder mit 30 Zeichen Länge
- Initialisierung mit Leerzeichen
- Speicherung von Namen darin

gegebener Code:

```
/**
 * Die Methode liefert die Position des letzten nicht leeren
 * Zeichens in der Zeichenkette text oder -1, wenn text nur aus
 * Leerzeichen besteht oder gar keine Zeichen enthält.
 *
 * @param text Eingabeparameter, Zeichenkette (Feld von elementaren
 *  char Datentypen)
 * @return letztePos Funktionswert, ganze Zahl >= -1
 */
int lokalisiereLetzes (char[] text) {

  final char leer = ' ';
  int letztePos;

  letztePos = text.length - 1;

  do {
    if (text[letztePos] == leer) {
      letztePos = letztePos - 1;
    }
  }
  while (letztePos >= 0 || text[letztePos] != leer);

  return letztePos;
}
```

# Aufgabe 2.1: Testen (3)

a) Black-Box Test

Array-Länge möglich von 0 bis 30.

- i. Äquivalenzklassen
  - 1. Kein Leerzeichen, Länge = 1, Bsp.: “a”
  - 2. Kein Leerzeichen, Länge > 1, Bsp.: “abc”
  - 3. Nur ein Leerzeichen, Bsp.: “ “
  - 4. Nur Leerzeichen, Länge > 1, Bsp.: “  “
- ii. Grenzwerte
  - 1. Leere Zeichenkette, Länge = 0
  - 2. Maximal lange Zeichenkette, Länge = 30
  - 3. Bsp Länge = 5; genau ein nicht leeres Zeichen am Ende
  - 4. Bsp Länge = 5; genau ein nicht leeres Zeichen am Anfang

# Aufgabe 2.1: Testen (4)

- iii. Testvorschrift

## Aufbau

**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**

## Testfälle

| Testfall | Eingabe | Erwartetes Resultat | Befund |
|---|---|---|---|
| 1 | „a“ | 0 | |
| 2 | „abc“ | 2 | |
| 3 | „ “ (1 leeres Zeichen) | -1 | |
| 4 | „ “ (2 leere Zeichen) | -1 | |
| 5 | „“ Leere Zeichenkette (Länge null) | -1 | |
| 6 | „abcde....xyz“ Zeichenkette der Länge 30 ohne Leerzeichen | 29 | |
| 7 | „abcd „ | 3 | |
| 8 | „ abcd“ | 4 | |
| 9 | „abc def„ | 6 | |

decken Äquivalenzklassen und Grenzwerte ab.

# Aufgabe 2.1: Testen (5)

## b) Schreibtischtest (White Box)

| Testfall | Eingabe | Erwartetes Resultat | **Befund** |
|---|---|---|---|
| 1 | „a“ | 0 | Endl.schl. |
| 2 | „abc“ | 2 | Endl.schl. |
| 3 | „ “ (1 leeres Zeichen) | -1 | ArrayOutOf-BoundsException |
| 4 | „ “ (2 leere Zeichen) | -1 | ArrayOutOf-BoundsException |
| 5 | „“ Leere Zeichenkette (Länge null) | -1 | ArrayOutOf-BoundsException |
| 6 | „abcde....xyz“ Zeichenkette der Länge 30 ohne Leerzeichen | 29 | Endl.schl. |
| 7 | „abcd „ | 3 | Endl.schl. |
| 8 | „ abcd“ | 4 | Endl.schl. |
| 9 | „abc def„ | 6 | Endl.schl. |

```
/**
 * Die Methode liefert die Position des letzten nicht leeren
 * Zeichens in der Zeichenkette text oder -1, wenn text nur aus
 * Leerzeichen besteht oder gar keine Zeichen enthält.
 *
 * @param text Eingabeparameter, Zeichenkette (Feld von elementaren
 *  char Datentypen)
 * @return letztePos Funktionswert, ganze Zahl >= -1
 */
int lokalisiereLetzes (char[] text) {

  final char leer = ' ';
  int letztePos;

  letztePos = text.length - 1;

  do {
    if (text[letztePos] == leer) {
      letztePos = letztePos - 1;
    }
  }
  while (letztePos >= 0 || text[letztePos] != leer);

  return letztePos;
}
```

- Befunde bereiteten teilw. Probleme.
- Werden Probleme im Code entdeckt --> dokumentieren!
  - *Abbruchbedingung sollte lauten:*
    - **letztePos >= 0 && text[letztePos] == leer**
  - Bsp.: while anstatt do-while Schleife beseitigt die if-Abfrage.

<u>Abbruchbedingung (Soll):</u>
Und: beide Argumenten müssen ‘true’ sein.
Java: Wenn erstes Argument ‘false’ ist und ein Und (&&) folgt, wird *nicht* weiter geprüft.

# Aufgabe 2.1: Testen (6)

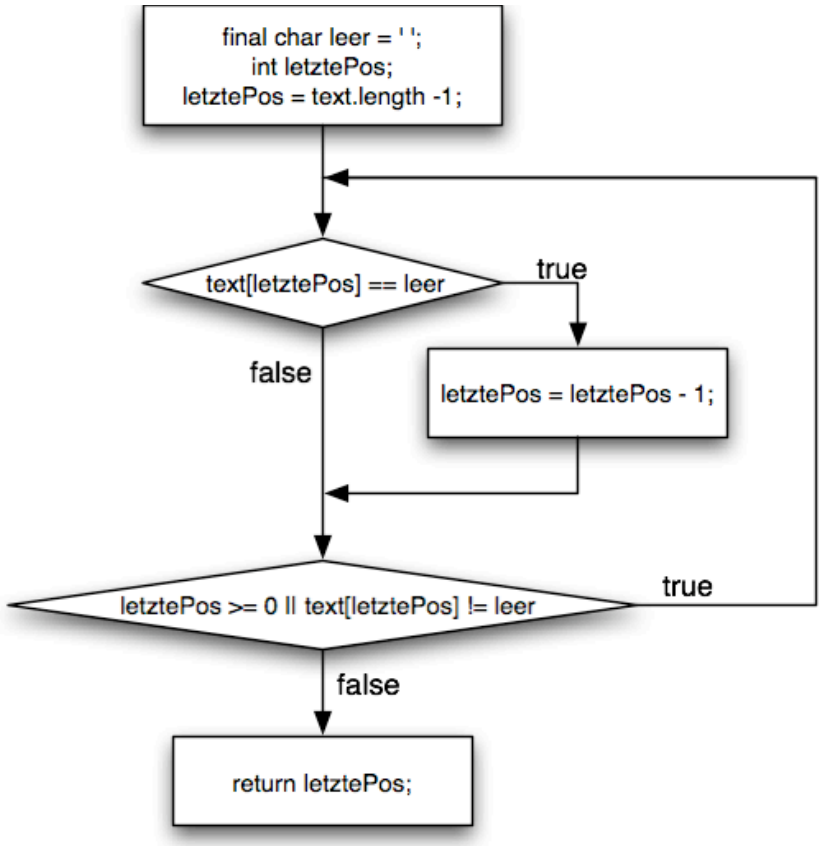
c) Zweigüberdeckung

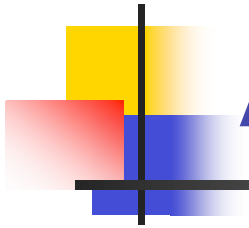
- = Anzahl durchlaufene Zweige / Anz. Zweige
- Jeder Pfeil ist ein Zweig
- Da es nicht terminiert:
  - Letzter Zweig wird nicht durchlaufen
  - 5 / 6
  - = 83,3 %

d) Zweigüberdeckung

- Da das Programm nicht terminiert wird 100% Zweigüberdeckung nicht erreicht.

Flussgraph:



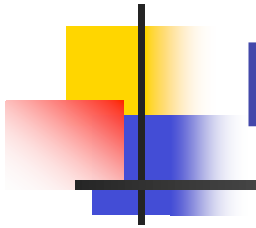


# Aufgabe 2.1: Testen (7)

e) White-Box versus Black-Box (am zuvor gezeigten Beispiel)

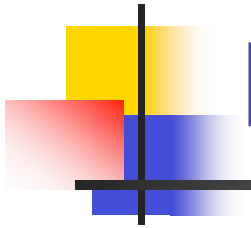
- White Box
  - Prüft Programmablauf und Datenfluss; Güte abhängig von Überdeckung;
  - +
    - “dead code” entdecken (Anweisungsüberdeckung)
    - Lokalisierung von Defekten
    - Falsch Implementierte Funktionalität
    - Programmierfehler
  - -
    - Fehler im Gesamtsystem (--> Integrationstests)
    - Fehler in der Spezifikation, Fehler in den Kommentaren
- Black Box
  - Blick “von weiter weg” auf das System
  - Finden von Äquivalenzklassen und Grenzfällen der Spezifikationen

# Aufgabe 2.2: Zielorientiertes Messen (1)



- Ziel
  - Suche nach Massen, welche das Ziel quantitativ charakterisieren
- Kriterien
  - Nur das messen was zur Erreichen der Ziele beiträgt
  - Festgelegte Interpretation der ausgewählten Masse
- Lösungen
  - Mittel bis gut

# Aufgabe 2.2: Zielorientiertes Messen (2)



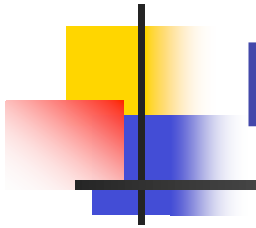
Vorgegebenes **Ziel**:

- *“Einfaches und schnelles Nachvollziehen des Simulationsablaufs und eventuelles Präzisieren von Simulationsparametern”*

a) *Faktoren* zur Zielerreichung und *Fragen* zur Qualitätskontrolle

- Übersichtlichkeit des GUI
  - Wie schnell kann man eine Simulation starten?
  - Wie einfach kann man während der Simulation Parameter anpassen?
- Intuitive Darstellung der Ergebnisse
  - Ist die Art der Darstellung der Ergebnisse leicht verstehbar?
- Hilfestellungen im Programm
  - Gib es im Programm genügend Mechanismen die Hilfestellung bieten?
- Benutzerfreundlichkeit des Editors
  - Wie leicht lassen sich neue Landschaften/Tiere erstellen?
  - Wie leicht lassen sich Paramter definieren?

# Aufgabe 2.2: Zielorientiertes Messen (3)

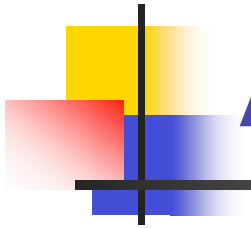


b) Messbare *Merkmale* und mögliche *Skalentypen* zu allen Fragen

- Wie schnell kann man eine Simulation starten?
  - Zeit vom Starten des Programms bis zum Simulationsbeginn.
    - Verhältnisskala (Zeit in Sekunden)
  - Rechenzeit beim Programm- und Simulationsstart
    - Verhältnisskala (Zeit in Sekunden)
- Wie einfach kann man während der Simulation Parameter anpassen?
  - Anzahl Klicks bis zur Anpassung
    - Absolutskala (ganze Zahl)
  - Einfachheit der Anpassung
    - Ordinalskala (Kategorien: einfach, mittel, schwierig)
  - ... weitere Merkmale
- ...

*Direkte Masse vs. Indirekte Masse:*

z.B. Durchlaufzeit ist direkt, Benutzerfreundlichkeit ist nur indirekt messbar.



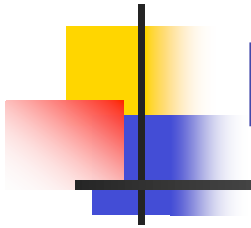
# Aufbau der Prüfung

- Zwei Teile, gesamt 120 Punkte und 90 Minuten Zeit.

- 1. Teil: Wissensfragen (40 Punkte, ca. 30 Minuten Bearbeitungszeit)
  - 10 Themenbereiche
  - Pro Bereich 6 Aussagen
  - Beurteilung der Aussagen mit: “richtig”, “falsch” oder keine Antwort.
    - Korrekte Ankreuzung: Punkte
    - Falsche Ankreuzung: Punkteabzug
    - Keine Antwort: keine Punkte

- 2. Teil: Anwendungsaufgaben (80 Punkte, ca. 60 Minuten Bearb.zeit)
  - Anwendungsaufgaben zum Stoff der Vorlesung
  - Ähnlich den Aufgaben in den Übungen und Mini-Übungen der Vorlesung

# Tutoren/TA für Modellierung FS 08



Frühjahrssemester 2008

- Wir suchen Tutoren und evtl einen Teaching Assistant für Informatik IIa: Modellierung.
  - Vorraussetzungen: Inf.IIa: Modellierung mit Erfolg bestanden; Interesse am Stoff und Freude an der Lehre.
  - Entlohnung: Geld, APS-Punkte, Horizonterweiterung.
  - Bei Interesse bitte bei Christina Cramer oder bei mir melden.

Danke für die Aufmerksamkeit.