

Softwaretechnik 1 - 6. Tutorium

Tutorium 17

Felix Bachmann | 23.07.2019

KIT - INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION (IPD)



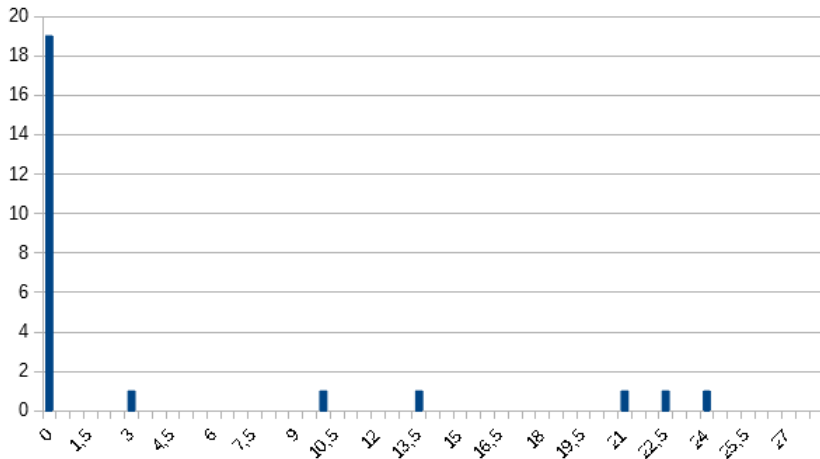
- 1 Orga
 - Feedback
- 2 Testen
 - Definitionen
- 3 Wiederholung und Klausuraufgaben
 - Planung & Definition
 - Entwurf
 - Implementierung
 - Testen
 - Abnahme, Einsatz & Wartung
 - Rest
- 4 Ende
 - Feierabend

Klausur, Übungsschein

- Hauptklausur am 01.08.19, 11:00
- Nachklausur wahrscheinlich am 07.10.19
- Anmeldung sollte nun für alle möglich sein

- bringe ich zu den Übungsleitern (R346, in der Nähe vom Holzkasten)
 - siehe <https://ps.ipd.kit.edu/3273.php>
- falls ihr die noch haben wollt, holt sie euch dort ab
- Übungsleiter beißen (meistens) nicht

6. Übungsblatt Statistik



Aufgabe 1: Kontrollfluss-orientiertes Testen

Aufgabe 1: Kontrollfluss-orientiertes Testen

- alles außer Kontrollfluss-Zeug so lassen wie es ist!

Aufgabe 1: Kontrollfluss-orientiertes Testen

- alles außer Kontrollfluss-Zeug so lassen wie es ist!
- Kontrollfluss-Zeug, das nicht `if(x) goto` ist auflösen!

Aufgabe 1: Kontrollfluss-orientiertes Testen

- alles außer Kontrollfluss-Zeug so lassen wie es ist!
- Kontrollfluss-Zeug, das nicht `if(x) goto` ist auflösen!
- außerdem Kurzschlussauswertung in zwei `if` auflösen

```
if(x && y) {  
    z++;  
}
```

```
if(x || y) {  
    z++;  
}
```

- Kurzschlussauswertung (&& bzw. ||) muss berücksichtigt werden

- Kurzschlussauswertung (&& bzw. ||) muss berücksichtigt werden
- Erinnerung:
 - && und || werten die rechte Seite nur aus, wenn notwendig
 - & und | werten immer beide Seiten aus

- Kurzschlussauswertung (&& bzw. ||) muss berücksichtigt werden
- Erinnerung:
 - && und || werten die rechte Seite nur aus, wenn notwendig
 - & und | werten immer beide Seiten aus

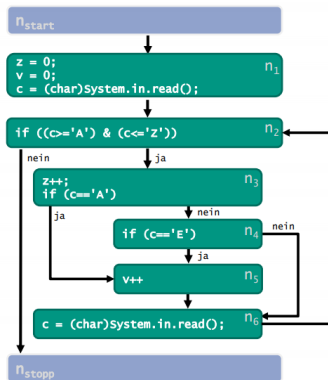
```
...
int z=0;
int v=0;
char c = (char)System.in.read();
while ((c>='A') & (c<='Z'))
{
    z++;
    if ((c=='A') || (c=='E'))
    {
        v++;
    }
    c = (char)System.in.read();
}
...
```

```

110: int z=0;
120: int v=0;
130: char c = (char)System.in.read();
140: if not ((c>='A') & (c<='Z')) goto 210;
150: z++;
160: if (c=='A') goto 180;
170: if not (c=='E') goto 190;
180: v++;
190: c = (char)System.in.read();
200: goto 140;
210: ...
    
```

Kurzschlussauswertung
muss berücksichtigt werden

(a) 1. Zwischensprache



(b) KFG

Aufgabe 2: Parallelisierung

- 5 Abgaben, meist richtig

Aufgabe 2: Parallelisierung

- 5 Abgaben, meist richtig
- Anzahl Prozessoren berechnen
 - `int nproc =`
`Runtime.getRuntime().availableProcessors();`

Aufgabe 2: Parallelisierung

- 5 Abgaben, meist richtig
- Anzahl Prozessoren berechnen
 - `int nproc =`
`Runtime.getRuntime().availableProcessors();`

Aufgabe 3: Abnahmetests

- 4 Abgaben
- Test brauchen immer Asserts!

Aufgabe 4: Wettbewerb

■ 3 Abgaben

- Was verursacht was?
- Defekt, Irrtum, Versagen

- Was verursacht was?
- Defekt, Irrtum, Versagen
- Irrtum (Aktion) → Defekt (Bug) → Ausfall (failure)
- dumme Eselsbrücke: IDA

- Was verursacht was?
- Defekt, Irrtum, Versagen
- Irrtum (Aktion) → Defekt (Bug) → Ausfall (failure)
- dumme Eselsbrücke: IDA
- „Testing shows the presence of bugs, not their absence.“ (Edsger W. Dijkstra)

Dynamische Verfahren

- Testfälle schreiben und ausführen (z.B. mit JUnit)

Dynamische Verfahren

- Testfälle schreiben und ausführen (z.B. mit JUnit)
- white box testing

Dynamische Verfahren

- Testfälle schreiben und ausführen (z.B. mit JUnit)
- white box testing
 - kontrollflussorientiert
 - datenflussorientiert
- black box testing

Dynamische Verfahren

- Testfälle schreiben und ausführen (z.B. mit JUnit)
- white box testing
 - kontrollflussorientiert
 - datenflussorientiert
- black box testing
 - funktionale Tests

Dynamische Verfahren

- Testfälle schreiben und ausführen (z.B. mit JUnit)
- white box testing
 - kontrollflussorientiert
 - datenflussorientiert
- black box testing
 - funktionale Tests
 - Leistungstests

Dynamische Verfahren

- Testfälle schreiben und ausführen (z.B. mit JUnit)
- white box testing
 - kontrollflussorientiert
 - datenflussorientiert
- black box testing
 - funktionale Tests
 - Leistungstests

Statische Verfahren

- Inspektion („scharfes Hinsehen“, Code Review, etc.)

Dynamische Verfahren

- Testfälle schreiben und ausführen (z.B. mit JUnit)
- white box testing
 - kontrollflussorientiert
 - datenflussorientiert
- black box testing
 - funktionale Tests
 - Leistungstests

Statische Verfahren

- Inspektion („scharfes Hinsehen“, Code Review, etc.)
- statische Analyse mit Tools (formale Verifikation, SpotBugs, etc.)

Dynamische Verfahren

- Testfälle schreiben und ausführen (z.B. mit JUnit)
- white box testing
 - kontrollflussorientiert
 - datenflussorientiert
- black box testing
 - funktionale Tests
 - Leistungstests

Statische Verfahren

- Inspektion („scharfes Hinsehen“, Code Review, etc.)
- statische Analyse mit Tools (formale Verifikation, SpotBugs, etc.)
- Programm wird nicht ausgeführt!

- Ich kenne die Klausur auch nicht!

- Ich kenne die Klausur auch nicht!
 - ⇒ alles, was ich zum Inhalt der Klausur sage ist Spekulation
 - basierend auf Altklausuren

- Ich kenne die Klausur auch nicht!
 \implies alles, was ich zum Inhalt der Klausur sage ist Spekulation
 - basierend auf Altklausuren
- kein Anspruch auf Vollständigkeit der Wiederholung

- 1 Aufgabe 1: Aufwärmen
 - Wahr-/Falsch-Fragen
 - ein paar gesammelt auf
`www.github.com/malluce/swt1-tut/multiple_choice.txt`
 - Wissensfragen

- ① Aufgabe 1: Aufwärmen
 - Wahr-/Falsch-Fragen
 - ein paar gesammelt auf
`www.github.com/malluce/swt1-tut/multiple_choice.txt`
 - Wissensfragen
- ② meistens Aufgaben zu:
 - UML-Diagrammen

1 Aufgabe 1: Aufwärmen

■ Wahr-/Falsch-Fragen

- ein paar gesammelt auf

`www.github.com/malluce/swt1-tut/multiple_choice.txt`

■ Wissensfragen

2 meistens Aufgaben zu:

■ UML-Diagrammen

■ Entwurfsmustern

1 Aufgabe 1: Aufwärmen

■ Wahr-/Falsch-Fragen

- ein paar gesammelt auf

`www.github.com/malluce/swt1-tut/multiple_choice.txt`

■ Wissensfragen

2 meistens Aufgaben zu:

- UML-Diagrammen
- Entwurfsmustern
- Parallelität

1 Aufgabe 1: Aufwärmen

■ Wahr-/Falsch-Fragen

- ein paar gesammelt auf

`www.github.com/malluce/swt1-tut/multiple_choice.txt`

■ Wissensfragen

2 meistens Aufgaben zu:

■ UML-Diagrammen

■ Entwurfsmustern

■ Parallelität

■ Testen bzw. Qualitätssicherung (KFG!)

1 Aufgabe 1: Aufwärmen

■ Wahr-/Falsch-Fragen

- ein paar gesammelt auf

`www.github.com/malluce/swt1-tut/multiple_choice.txt`

■ Wissensfragen

2 meistens Aufgaben zu:

- UML-Diagrammen
- Entwurfsmustern
- Parallelität
- Testen bzw. Qualitätssicherung (KFG!)
- Rest (z.B. Objektorientierung, Git/Versionskontrolle, Prozessmodelle...)

1 Aufgabe 1: Aufwärmen

■ Wahr-/Falsch-Fragen

- ein paar gesammelt auf

www.github.com/malluce/swt1-tut/multiple_choice.txt

■ Wissensfragen

2 meistens Aufgaben zu:

■ UML-Diagrammen

■ Entwurfsmustern

■ Parallelität

■ Testen bzw. Qualitätssicherung (KFG!)

■ Rest (z.B. Objektorientierung, Git/Versionskontrolle, Prozessmodelle...)

■ bisher: $\frac{1}{3} \pm \epsilon$ der Punkte reichen zum Bestehen

■ kann sich eventuell ändern, wegen längerer Bearbeitungszeit

Hauptklausur SS2011 A1

- Lastenheft, Pflichtenheft

- Lastenheft, Pflichtenheft
 - Phasen zuordnen

- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen

- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben

- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben
- UML-Diagramme

- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben
- UML-Diagramme
 - Klassendiagramm

- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben
- UML-Diagramme
 - Klassendiagramm
 - Aktivitäts-, Sequenz-, Zustandsdiagramm

- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben
- UML-Diagramme
 - Klassendiagramm
 - Aktivitäts-, Sequenz-, Zustandsdiagramm
 - Anwendungsfalldiagramm

- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben
- UML-Diagramme
 - Klassendiagramm
 - Aktivitäts-, Sequenz-, Zustandsdiagramm
 - Anwendungsfalldiagramm
 - Syntax kennen!

- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben
- UML-Diagramme
 - Klassendiagramm
 - Aktivitäts-, Sequenz-, Zustandsdiagramm
 - Anwendungsfalldiagramm
 - Syntax kennen!
 - gegebenen Text/Code in Diagramm umwandeln

- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben
- UML-Diagramme
 - Klassendiagramm
 - Aktivitäts-, Sequenz-, Zustandsdiagramm
 - Anwendungsfalldiagramm
 - Syntax kennen!
 - gegebenen Text/Code in Diagramm umwandeln
 - bei Zustandsdiagramm
 - Umwandeln hierarchisch \Leftrightarrow nicht-hierarchisch
 - Umwandeln parallel \Leftrightarrow nicht-parallel

Hauptklausur SS2012 A2

■ Architekturstile

- Architekturstile
- **Entwurfsmuster**

- Architekturstile
- **Entwurfsmuster**
 - möglichst viele, bestenfalls alle kennen und verstehen

- Architekturstile
- **Entwurfsmuster**
 - möglichst viele, bestenfalls alle kennen und verstehen
 - Kategorien kennen

- Architekturstile
- **Entwurfsmuster**
 - möglichst viele, bestenfalls alle kennen und verstehen
 - Kategorien kennen
 - Klassendiagramm hinzeichnen (auch auf Szenario angepasst)

- Architekturstile
- **Entwurfsmuster**
 - möglichst viele, bestenfalls alle kennen und verstehen
 - Kategorien kennen
 - Klassendiagramm hinzeichnen (auch auf Szenario angepasst)
 - aus Klassendiagrammen/Code Entwurfsmuster erkennen

- Architekturstile
- **Entwurfsmuster**
 - möglichst viele, bestenfalls alle kennen und verstehen
 - Kategorien kennen
 - Klassendiagramm hinzeichnen (auch auf Szenario angepasst)
 - aus Klassendiagrammen/Code Entwurfsmuster erkennen
 - Code für Muster schreiben

- Architekturstile
- **Entwurfsmuster**
 - möglichst viele, bestenfalls alle kennen und verstehen
 - Kategorien kennen
 - Klassendiagramm hinzeichnen (auch auf Szenario angepasst)
 - aus Klassendiagrammen/Code Entwurfsmuster erkennen
 - Code für Muster schreiben
 - Code-Schnipsel auf mögliche Verbesserung durch EM untersuchen

Hauptklausur SS2010 A3

■ UML-Abbildung

- UML-Abbildung
- **Parallelität**

- UML-Abbildung
- **Parallelität**
 - grundlegendes Prinzip

- UML-Abbildung
- **Parallelität**
 - grundlegendes Prinzip
 - in Java

- UML-Abbildung
- **Parallelität**
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions

- UML-Abbildung
- **Parallelität**
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions
 - deadlock

- UML-Abbildung
- **Parallelität**
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions
 - deadlock
 - Monitore, wait & notify

- UML-Abbildung
- **Parallelität**
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions
 - deadlock
 - Monitore, wait & notify
 - Semaphore

- UML-Abbildung
- **Parallelität**
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions
 - deadlock
 - Monitore, wait & notify
 - Semaphore
- Rechnungen können (Speedup, Amdahls Law (**Tafel**))

- UML-Abbildung
- **Parallelität**
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions
 - deadlock
 - Monitore, wait & notify
 - Semaphore
- Rechnungen können (Speedup, Amdahls Law (**Tafel**))
- gegebenen Code thread-safe machen

- UML-Abbildung
- **Parallelität**
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions
 - deadlock
 - Monitore, wait & notify
 - Semaphore
- Rechnungen können (Speedup, Amdahls Law (**Tafel**))
- gegebenen Code thread-safe machen
- Lösungsvorschläge zur Synchronisation bewerten

- UML-Abbildung
- **Parallelität**
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions
 - deadlock
 - Monitore, wait & notify
 - Semaphore
- Rechnungen können (Speedup, Amdahls Law (**Tafel**))
- gegebenen Code thread-safe machen
- Lösungsvorschläge zur Synchronisation bewerten
- eigenen Code schreiben

Hauptklausur SS2011 A3

- Testphasen
 - Komponententest, Integrationstest, Systemtest, Abnahmetest

- Testphasen
 - Komponententest, Integrationstest, Systemtest, Abnahmetest
- Testverfahren
 - **Kontrollflussgraph**

- Testphasen
 - Komponententest, Integrationstest, Systemtest, Abnahmetest
- Testverfahren
 - **Kontrollflussgraph !!!**
 - absolute Standardaufgabe

- Testphasen
 - Komponententest, Integrationstest, Systemtest, Abnahmetest
- Testverfahren
 - **Kontrollflussgraph !!!**
 - absolute Standardaufgabe
 - in 33 von 36 der letzten Altklausuren
 - in allen Klausuren seit SS14
 - im Schlaf können ("Schema F", lässt sich sehr gut üben. . .)

- Testphasen
 - Komponententest, Integrationstest, Systemtest, Abnahmetest
- Testverfahren
 - **Kontrollflussgraph !!!**
 - absolute Standardaufgabe
 - in 33 von 36 der letzten Altklausuren
 - in allen Klausuren seit SS14
 - im Schlaf können ("Schema F", lässt sich sehr gut üben. . .)
- Testhelfer
- Definitionen kennen (Fehlerarten. . .)

Hauptklausur SS2016 A6

- Aufgaben der verschiedenen “Subphasen” kennen

- Aufgaben der verschiedenen “Subphasen” kennen
- viel Text zum Lernen, aber nicht schwierig. . .

- Aufgaben der verschiedenen “Subphasen” kennen
- viel Text zum Lernen, aber nicht schwierig. . .
- Wartung vs. Pflege

- Aufgaben der verschiedenen “Subphasen” kennen
- viel Text zum Lernen, aber nicht schwierig. . .
- Wartung vs. Pflege
- selten eigene Aufgabe dazu, eher Ankreuzaufgaben in A1

■ Schätzmethoden

- Schätzmethoden
- Prozessmodelle

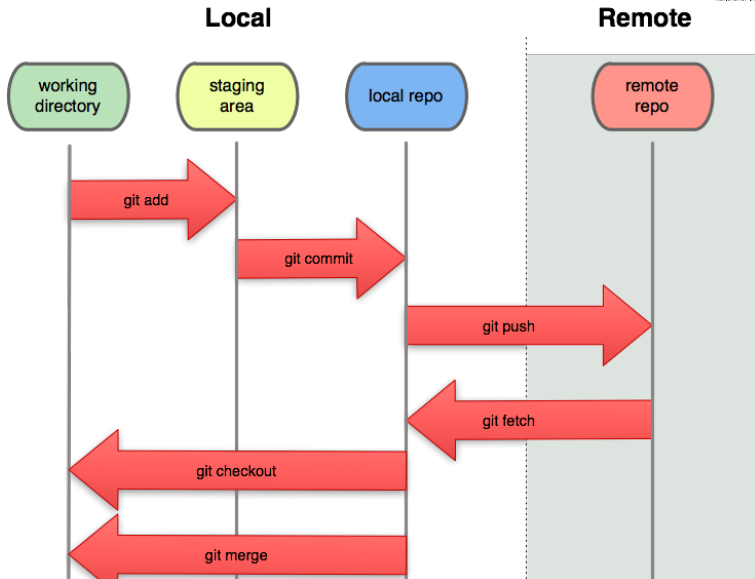
- Schätzmethoden
- Prozessmodelle
- Agile Prozesse

- Schätzmethoden
- Prozessmodelle
- Agile Prozesse
- verschiedene Modelle kennen (XP, Scrum,...)

- Schätzmethoden
- Prozessmodelle
- Agile Prozesse
- verschiedene Modelle kennen (XP, Scrum,...)
- auch eher Ankreuzaufgaben, Wissensfragen

Hauptklausur SS18 A4

Git: Klausuraufgabe



Nachklausur SS2011 A5

- Klausuren rechnen && Folien anschauen
 - Übung nötig, für sehr gute Note aber auch viel Wissen

- Klausuren rechnen && Folien anschauen
 - Übung nötig, für sehr gute Note aber auch viel Wissen
- diesmal habt ihr mehr Zeit für den gleichen Umfang an Aufgaben wie in früheren Klausuren
 - vermutlich wird Bestehensgrenze hochgesetzt

Das war's dann wohl...

- danke für Eure Anwesenheit und Mitarbeit!
- Viel Erfolg bei der Klausur und im weiteren Studium! :)

SIMPLY EXPLAINED

