

# Softwaretechnik 1 - 6. Tutorium

Tutorium 18

Felix Bachmann | 17.07.2018

KIT - INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION (IPD)

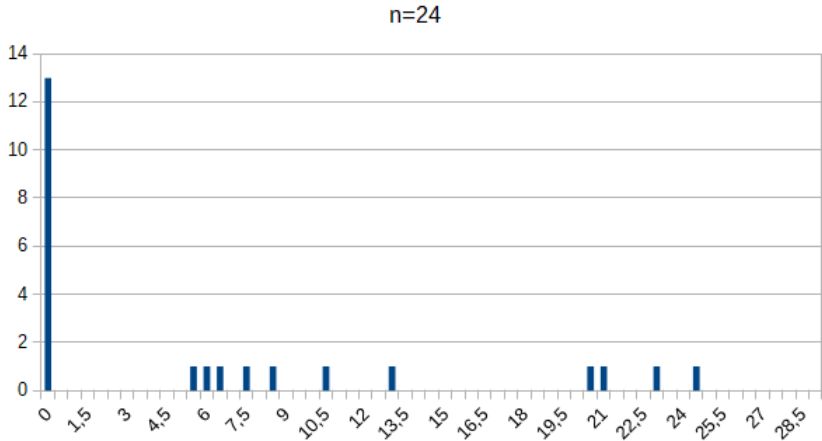


- 1 Orga
  - Feedback
- 2 Testen
  - Definitionen
- 3 Wiederholung und Klausuraufgaben
  - Planung & Definition
  - Entwurf
  - Implementierung
  - Testen
  - Abnahme, Einsatz & Wartung
  - Rest
- 4 Ende
  - Feierabend

## Klausur, Übungsschein

- Hauptklausur: 26. Juli 2018, 15:00 Uhr
- Nachklausur: findet am 08.10 statt
- Anmeldung sollte nun für alle möglich sein

# 6. Übungsblatt Statistik



Ø 6,1 bzw. 13,3 von 26+4

## 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!
- minimale Testfallmenge = Eingabe, um Pfade zu erreichen

## Aufgabe 1: Kontrollfluss-orientiertes Testen

- alles außer Kontrollfluss-Zeug so lassen wie es ist!
- minimale Testfallmenge = Eingabe, um Pfade zu erreichen
- Methoden-Signatur nicht als eigenen Grundblock



## Aufgabe 1: Kontrollfluss-orientiertes Testen

- alles außer Kontrollfluss-Zeug so lassen wie es ist!
- minimale Testfallmenge = Eingabe, um Pfade zu erreichen
- Methoden-Signatur nicht als eigenen Grundblock

```
10 fakultaet *= n;  
20  n--;  
30  if(n > 1) goto 10;
```

## Aufgabe 2: Parallelisierung von Shutterpile

## Aufgabe 2: Parallelisierung von Shutterpile



## Aufgabe 2: Parallelisierung von Shutterpile



## Aufgabe 3: Abnahmetests

## Aufgabe 2: Parallelisierung von Shutterpile



## Aufgabe 3: Abnahmetests



## Aufgabe 4: Parallelisierungswettbewerb

## Aufgabe 4: Parallelisierungswettbewerb

- ungenaue Beschreibung, wie man vorgehen würde

- Was verursacht was?
- Defekt, Irrtum, Versagen



- Was verursacht was?
- Defekt, Irrtum, Versagen
- “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 testings

## Dynamische Verfahren

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

## Dynamische Verfahren

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

## Dynamische Verfahren

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

## Statische Verfahren

- Inspektion

## Dynamische Verfahren

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

## Statische Verfahren

- Inspektion
- statische Analyse mit Tools



## Dynamische Verfahren

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

## Statische Verfahren

- Inspektion
- statische Analyse mit Tools
- 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: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](https://www.github.com/malluce/swt1-tut)) und Wissensfragen

- ① Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](https://www.github.com/malluce/swt1-tut)) und Wissensfragen
- ② meistens Aufgaben zu:
  - UML-Diagrammen

- ① Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](https://www.github.com/malluce/swt1-tut)) und Wissensfragen
- ② meistens Aufgaben zu:
  - UML-Diagrammen
  - Entwurfsmustern

- ① Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](https://www.github.com/malluce/swt1-tut)) und Wissensfragen
- ② meistens Aufgaben zu:
  - UML-Diagrammen
  - Entwurfsmustern
  - Parallelität



- ① Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](https://www.github.com/malluce/swt1-tut)) und Wissensfragen
- ② meistens Aufgaben zu:
  - UML-Diagrammen
  - Entwurfsmustern
  - Parallelität
  - Testen bzw. Qualitätssicherung

- ① Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](https://www.github.com/malluce/swt1-tut)) und Wissensfragen
- ② meistens Aufgaben zu:
  - UML-Diagrammen
  - Entwurfsmustern
  - Parallelität
  - Testen bzw. Qualitätssicherung
  - Rest (z.B. Objektorientierung, Abbott, Prozessmodelle. . .)

- ① Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](https://www.github.com/malluce/swt1-tut)) und Wissensfragen
- ② meistens Aufgaben zu:
  - UML-Diagrammen
  - Entwurfsmustern
  - Parallelität
  - Testen bzw. Qualitätssicherung
  - Rest (z.B. Objektorientierung, Abbott, Prozessmodelle. . .)
- $1/3 \pm \epsilon$  der Punkte reichen (meistens) zum Bestehen

## 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 in Diagramm umwandeln

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

## Nachklausur SS2011 A5



## 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

- Architekturstile
- **Entwurfsmuster**
  - möglichst viele, bestenfalls alle kennen und verstehen
  - Kategorien kennen
  - Klassendiagramm hinzeichnen
  - aus Klassendiagrammen Entwurfsmuster erkennen

- Architekturstile
- **Entwurfsmuster**
  - möglichst viele, bestenfalls alle kennen und verstehen
  - Kategorien kennen
  - Klassendiagramm hinzeichnen
  - aus Klassendiagrammen Entwurfsmuster erkennen
  - Code für einfache Muster (Singleton... ) schreiben



- Architekturstile
- **Entwurfsmuster**
  - möglichst viele, bestenfalls alle kennen und verstehen
  - Kategorien kennen
  - Klassendiagramm hinzeichnen
  - aus Klassendiagrammen Entwurfsmuster erkennen
  - Code für einfache Muster (Singleton. . . ) 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, ...)

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

## Hauptklausur SS2011 A3

## ■ Testphasen



- Testphasen
- Testverfahren
  - **KFO**

- Testphasen
- Testverfahren
  - **KFO**
- Testhelfer

- Testphasen
- Testverfahren
  - **KFO**
- Testhelfer
- Definitionen kennen (Fehlerarten. . .)

- Testphasen
- Testverfahren
  - **KFO**
- Testhelfer
- Definitionen kennen (Fehlerarten. . .)
- KFO im Schlaf können (“Schema F“, lässt sich sehr gut üben. . .)

- 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
- wahrscheinlich Ankreuzaufgaben dazu



## ■ 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

- Klausuren rechnen && Folien anschauen

# Das war's dann wohl...

Viel Erfolg bei der Klausur und im weiteren Studium! :)

## SIMPLY EXPLAINED

