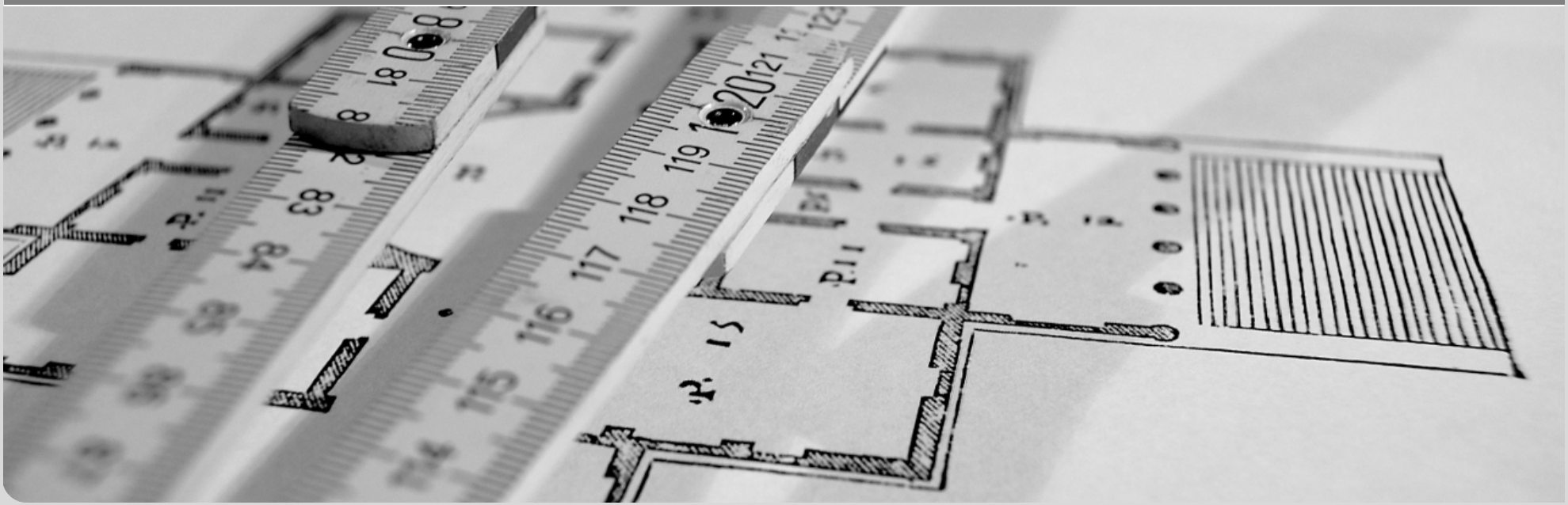
Karlsruher Institut für Technologie

Softwaretechnik 1 - 6. Tutorium

Tutorium 18

Felix Bachmann | 17.08.2018

KIT - INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION (IPD)



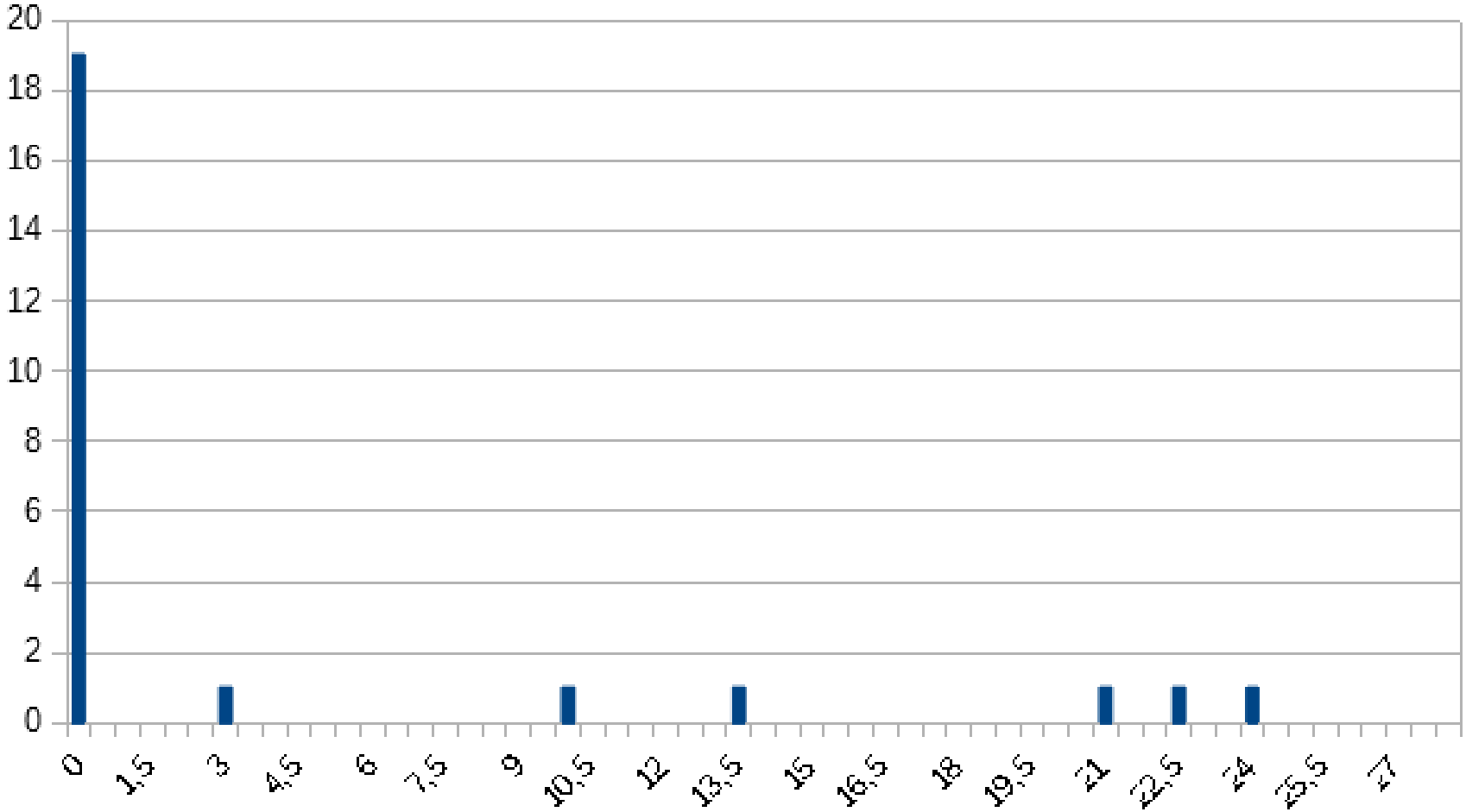
1. Orga
   - Feedback
2. Testen
   - Definitionen
3. Wiederholung und Klausuraufgaben
   - Planung & Definition
   - Entwurf
   - Implementierung
   - Testen
   - Abnahme, Einsatz & Wartung
   - Rest
4. Ende
   - Feierabend

Allgemein

Klausur, Übungsschein

- Hauptklausur am 26.07.18, 15:00
- Nachklausur wahrscheinlich am 08.10.18
- Anmeldung sollte nun für alle möglich sein

6. Übungsblatt Statistik



Häufige Fehler

Aufgabe 1: Kontrollfluss-orientiertes Testen

Häufige Fehler

Aufgabe 1: Kontrollfluss-orientiertes Testen

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

Häufige Fehler

Aufgabe 1: Kontrollfluss-orientiertes Testen

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

Häufige Fehler

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++;
}
```

KFO: Kurzschlussauswertung

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

KFO: Kurzschlussauswertung

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

KFO: Kurzschlussauswertung

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

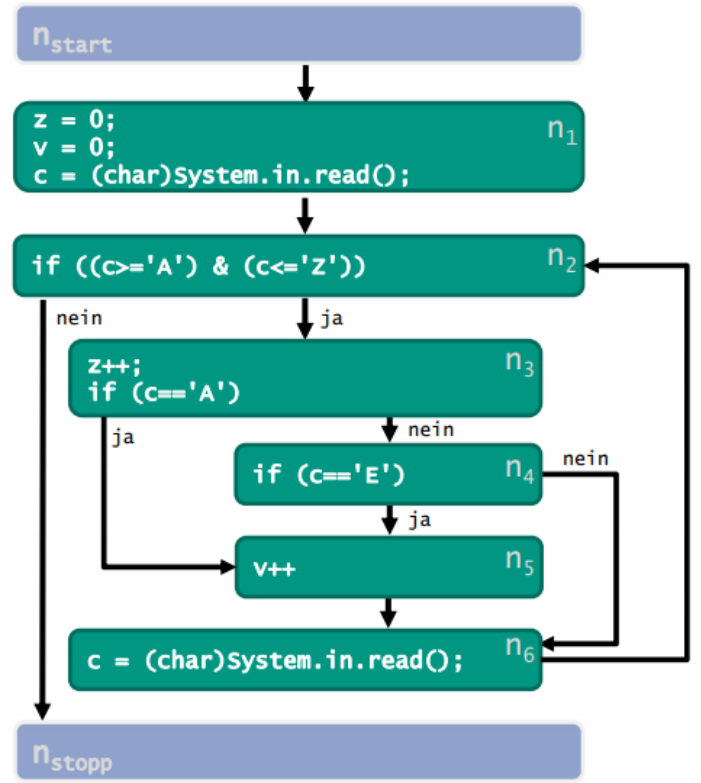
```
…
int z=0;
int v=0;
char c = (char)System.in.read();
while ((c>='A') & (c<='Z'))
{
  z++;
  if ((c=='A') || (c=='E'))
  {
    v++;
  }
  c = (char)System.in.read();
}
…
```

KFO: Kurzschlussauswertung

```
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

Häufige Fehler

Aufgabe 2: Parallelisierung

- 5 Abgaben, meist richtig

Häufige Fehler

Aufgabe 2: Parallelisierung

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

Häufige Fehler

Aufgabe 2: Parallelisierung

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

Aufgabe 3: Abnahmetests

- 4 Abgaben
- Test brauchen immer Asserts!

Häufige Fehler

Aufgabe 4: Wettbewerb

- 3 Abgaben

Arten von Fehlern

- Was verursacht was?
- Defekt, Irrtum, Versagen

Arten von Fehlern

- Was verursacht was?
- Defekt, Irrtum, Versagen
- “Testing shows the presence of bugs, not their absence.“ (Edsger W. Dijkstra)

Arten von Tests

Dynamische Verfahren

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

Arten von Tests

Dynamische Verfahren

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

Arten von Tests

Dynamische Verfahren

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

Arten von Tests

Dynamische Verfahren

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

Arten von Tests

Dynamische Verfahren

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

Arten von Tests

Dynamische Verfahren

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

Statische Verfahren

- Inspektion

Arten von Tests

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

Arten von Tests

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!

KFO: Klausuraufgabe mit Kurzschlussauswertung

Karlsruher Institut für Technologie

Hauptklausur SS2016 A6

Disclaimer

- Ich kenne die Klausur auch nicht!

Disclaimer

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

Disclaimer

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

Die typische SWT-Klausur

1. Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf `www.github.com/malluce/swt1-tut`) und Wissensfragen

Die typische SWT-Klausur

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

Die typische SWT-Klausur

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

Die typische SWT-Klausur

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

Die typische SWT-Klausur

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

Die typische SWT-Klausur

1. Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf `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...)

Die typische SWT-Klausur

1. Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf `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

Aufwärmaufgabe

Karlsruher Institut für Technologie

Hauptklausur SS2011 A1

Planung & Definition

- Lastenheft, Pflichtenheft

Planung & Definition

- Lastenheft, Pflichtenheft
  - Phasen zuordnen

Planung & Definition

- Lastenheft, Pflichtenheft
  - Phasen zuordnen
  - Gliederung kennen

Planung & Definition

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

Planung & Definition

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

Planung & Definition

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

Planung & Definition

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

Planung & Definition

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

Planung & Definition

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

Planung & Definition

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

Planung & Definition

- 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

Klassendiagramm-Aufgabe

Karlsruher Institut für Technologie

Nachklausur SS2011 A5

Zustandsdiagramm-Aufgabe

Karlsruher Institut für Technologie

Hauptklausur SS2012 A2

Entwurf

- Architekturstile

Entwurf

- Architekturstile
- **Entwurfsmuster**

Entwurf

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

Entwurf

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

Entwurf

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

Entwurf

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

Entwurf

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

Entwurf

- 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

Entwurfsmuster-Aufgabe

Karlsruher Institut für Technologie

Hauptklausur SS2010 A3

Implementierung

- UML-Abbildung

Implementierung

- UML-Abbildung
- **Parallelität**

Implementierung

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

Implementierung

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

Implementierung

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

Implementierung

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

Implementierung

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

Implementierung

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

Implementierung

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

Implementierung

- 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

Implementierung

- 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

Implementierung

- 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

Parallelität-Aufgabe

Karlsruher Institut für Technologie

Hauptklausur SS2011 A3

Testen

- Testphasen

Testen

- Testphasen
- Testverfahren
  - **KFO**

Testen

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

Testen

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

Testen

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

Abnahme, Einsatz & Wartung

- Aufgaben der verschiedenen “Subphasen“ kennen

Abnahme, Einsatz & Wartung

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

Abnahme, Einsatz & Wartung

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

Abnahme, Einsatz & Wartung

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

Rest

- Schätzmethoden

Rest

- Schätzmethoden
- Prozessmodelle

Rest

- Schätzmethoden
- Prozessmodelle
- Agile Prozesse

Rest

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

Rest

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

Lernen

- Klausuren rechnen && Folien anschauen

Das war's dann wohl...

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