

# Softwaretechnik 1 - 6. Tutorium

Tutorium 18

Felix Bachmann | 17.08.2018

KIT - INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION (IPD)



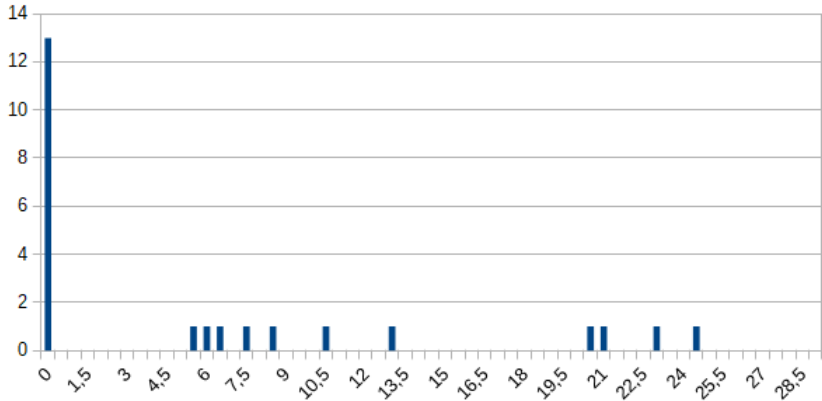
- 1 Orga
  - Feedback
- 2 Testen
  - KFO
- 3 Wiederholung und Klausuraufgaben
  - Planung & Definition
  - Entwurf
  - Implementierung
  - Testen
  - Abnahme, Einsatz & Wartung
  - Rest
- 4 Ende

## 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

n=24



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!
- 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++;  
}
```



## Aufgabe 2: Parallelisierung

- 5 Abgaben, meist richtig

## Aufgabe 2: Parallelisierung

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

## Aufgabe 2: Parallelisierung

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

## Aufgabe 3: Abnahmetests

- 4 Abgaben
- Test brauchen immer Asserts!

## Aufgabe 4: Wettbewerb

■ 3 Abgaben

# Kontrollflussorientiertes Testverfahren (KFO)

- Ziel: “sinnvolle“ Testfälle finden

Vorgehen:

- ① gegeben: zu testender Code

# Kontrollflussorientiertes Testverfahren (KFO)

- Ziel: “sinnvolle“ Testfälle finden

Vorgehen:

- ① gegeben: zu testender Code
- ② Code  $\implies$  Zwischensprache
  - Sprünge umwandeln
  - Grundblöcke finden
  - Grundblöcke prüfen

# Kontrollflussorientiertes Testverfahren (KFO)

- Ziel: “sinnvolle“ Testfälle finden

Vorgehen:

- ① gegeben: zu testender Code
- ② Code  $\implies$  Zwischensprache
  - Sprünge umwandeln
  - Grundblöcke finden
  - Grundblöcke prüfen
- ③ Zwischensprache  $\implies$  Kontrollflussgraph

# Kontrollflussorientiertes Testverfahren (KFO)

- Ziel: “sinnvolle“ Testfälle finden

Vorgehen:

- ① gegeben: zu testender Code
- ② Code  $\implies$  Zwischensprache
  - Sprünge umwandeln
  - Grundblöcke finden
  - Grundblöcke prüfen
- ③ Zwischensprache  $\implies$  Kontrollflussgraph
- ④ am Kontrollflussgraphen Testfälle finden:



# Kontrollflussorientiertes Testverfahren (KFO)

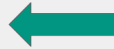
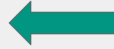
- Ziel: “sinnvolle“ Testfälle finden

Vorgehen:

- ① gegeben: zu testender Code
- ② Code  $\implies$  Zwischensprache
  - Sprünge umwandeln
  - Grundblöcke finden
  - Grundblöcke prüfen
- ③ Zwischensprache  $\implies$  Kontrollflussgraph
- ④ am Kontrollflussgraphen Testfälle finden:
  - Anweisungsüberdeckung
  - Zweigüberdeckung
  - Pfadüberdeckung

## ■ Sprünge umwandeln

```
1  int a = 9;
2  System.out.println("Blahblah");
3  while(a == 9) {
4      int z = 0;
5      for(int i = 0; i <= 8; i++) {
6          z++;
7      }
8      int k = 0;
9      if(a == z + k) {
10         a = 8;
11     }
12 }
```



## ■ Sprünge umwandeln

```
1  int a = 9;
2  System.out.println("Blahblah");
3  if not (a == 9) goto 14;
4      int z = 0;
5      int i = 0;
6      if not (i <= 8) goto 10;
7          z++;
8          i++;
9      goto 6;
10     int k = 0;
11     if not (a == z + k) goto 13;
12         a = 8;
13 goto 3;
14
```

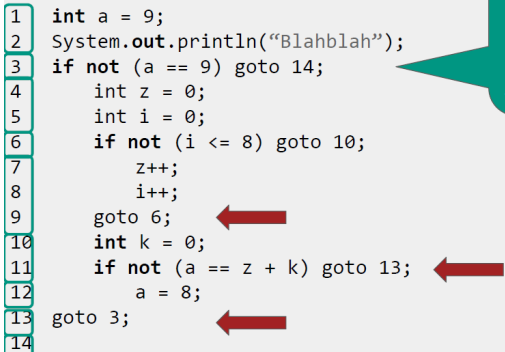
- Grundblöcke finden (Code bis goto ist ein Grundblock)

```
1  int a = 9;  
2  System.out.println("Blahblah");  
3  if not (a == 9) goto 14;  
4      int z = 0;  
5      int i = 0;  
6      if not (i <= 8) goto 10;  
7          z++;  
8          i++;  
9      goto 6;  
10     int k = 0;  
11     if not (a == z + k) goto 13;  
12         a = 8;  
13 goto 3;  
14
```

Grundblöcke dürfen  
nur am Ende einen  
Sprung (goto)  
haben (müssen  
aber nicht)

- Grundblöcke prüfen (goto dürfen nur an Anfang eines Grundblocks verweisen)

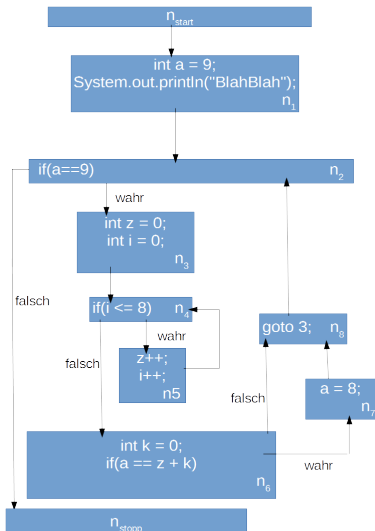
```
1  int a = 9;  
2  System.out.println("Blahblah");  
3  if not (a == 9) goto 14;  
4      int z = 0;  
5      int i = 0;  
6      if not (i <= 8) goto 10;  
7          z++;  
8          i++;  
9      goto 6;  
10     int k = 0;  
11     if not (a == z + k) goto 13;  
12         a = 8;  
13     goto 3;  
14
```



# KFO: Zwischensprache nach Kontrollflussgraph

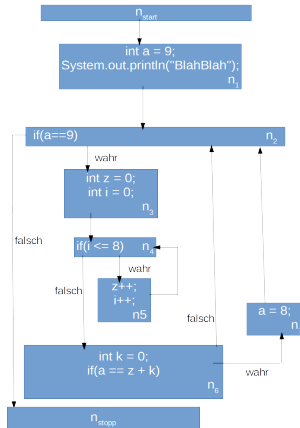
- Grundblöcke benennen
- Grundblöcke und Verzweigungen hinzeichnen
- Start- und Endzustand hinzufügen

# KFO: Zwischensprache nach Kontrollflussgraph



# KFO: Zwischensprache nach Kontrollflussgraph

- goto-Knoten kann man auch weglassen

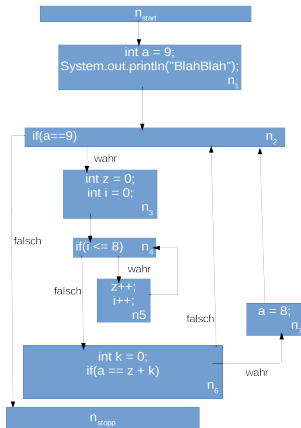




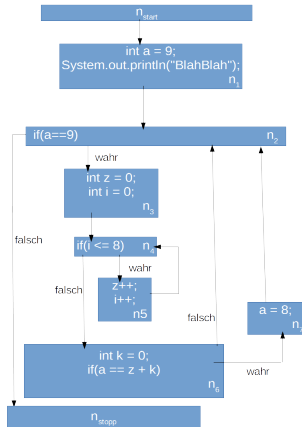
- Pfade finden, sodass jeder Grundblock traversiert wird

- Pfade finden, sodass jeder Grundblock traversiert wird  
⇒ Entdeckung nicht erreichbarer Code-Abschnitte

- Pfade finden, sodass jeder Grundblock traversiert wird  
⇒ Entdeckung nicht erreichbarer Code-Abschnitte
- aber: kein ausreichendes Testkriterium



## ■ Pfad für Anweisungsüberdeckung?



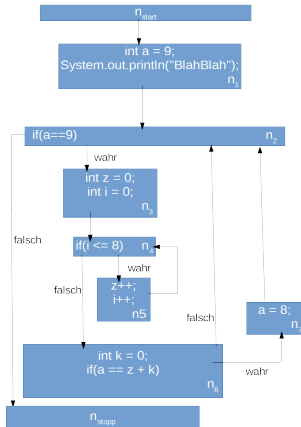
- Pfad für Anweisungsüberdeckung?  
 $(n_{start}, n_1, n_2, n_3, n_4, n_5, n_4, n_6, n_7, n_2, n_{stopp})$

- Pfade finden, sodass jeder Zweig (=Kante) traversiert wird

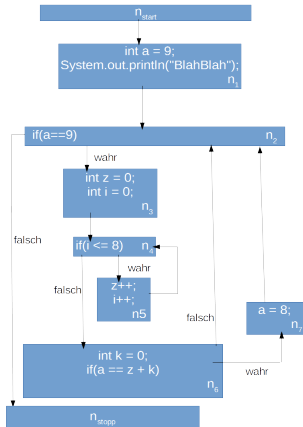
- Pfade finden, sodass jeder Zweig (=Kante) traversiert wird  
⇒ Entdeckung nicht erreichbarer Kanten

- Pfade finden, sodass jeder Zweig (=Kante) traversiert wird  
⇒ Entdeckung nicht erreichbarer Kanten
- aber: Schleifen werden nicht ausreichend getestet





## ■ Pfad für Zweigüberdeckung?



## ■ Pfad für Zweigüberdeckung?

$(n_{start}, n_1, n_2, n_3, n_4, n_5, n_4, n_6, n_2, n_3, n_4, n_5, n_4, n_6, n_7, n_2, n_{stopp})$

- Finde **alle** vollständige, unterschiedlichen Pfade

- Finde **alle** vollständige, unterschiedlichen Pfade
- vollständiger Pfad = Anfang bei  $n_{start}$ , Ende bei  $n_{stop}$

- Finde **alle** vollständige, unterschiedlichen Pfade
- vollständiger Pfad = Anfang bei  $n_{start}$ , Ende bei  $n_{stop}$
- nicht praktikabel, da
  - Schleifen die Anzahl der möglichen Pfade stark erhöhen

- Finde **alle** vollständige, unterschiedlichen Pfade
- vollständiger Pfad = Anfang bei  $n_{start}$ , Ende bei  $n_{stop}$
- nicht praktikabel, da
  - Schleifen die Anzahl der möglichen Pfade stark erhöhen
  - manche Pfade nicht ausführbar sind (sich gegenseitig ausschließende Bedingungen)

- 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

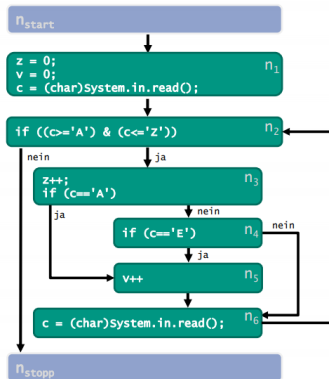
```
...
int z=0;
int v=0;
char c = (char)System.in.read();
while ((c>='A') & (c<='Z'))
{
    z++;
    if ((c=='A') || (c=='E'))
    {
        v++;
    }
    c = (char)System.in.read();
}
...
```

```

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

## Hauptklausur SS2016 A6

- 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!  
⇒ 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:
  - 1-2 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:
  - 1-2 Aufgaben zu UML-Diagrammen
  - 1 Aufgabe zu 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:
  - 1-2 Aufgaben zu UML-Diagrammen
  - 1 Aufgabe zu Entwurfsmustern
  - 1 Aufgabe zu Parallelität

- ➊ Aufgabe 1: Wahr-/Falsch-Fragen (ein paar gesammelt auf [www.github.com/malluce/swt1-tut](https://www.github.com/malluce/swt1-tut)) und Wissensfragen
- ➋ meistens:
  - 1-2 Aufgaben zu UML-Diagrammen
  - 1 Aufgabe zu Entwurfsmustern
  - 1 Aufgabe zu Parallelität
  - 1 Aufgabe zu 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:
  - 1-2 Aufgaben zu UML-Diagrammen
  - 1 Aufgabe zu Entwurfsmustern
  - 1 Aufgabe zu Parallelität
  - 1 Aufgabe zu Testen bzw. Qualitätssicherung
  - 1 Aufgabe 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:
  - 1-2 Aufgaben zu UML-Diagrammen
  - 1 Aufgabe zu Entwurfsmustern
  - 1 Aufgabe zu Parallelität
  - 1 Aufgabe zu Testen bzw. Qualitätssicherung
  - 1 Aufgabe 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  $\wedge$  Folien anschauen)
  - > (Klausuren rechnen XOR Folien anschauen)



# Das war's dann wohl...

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

## SIMPLY EXPLAINED

