

Softwaretechnik 1 - 2. Tutorium

Tutorium 03

Felix Bachmann | 29.05.2017

KIT - INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION (IPD)



2. Übungsblatt Statistik

Allgemein

Geben Sie Ihre Lösung mit Deckblatt (mit Name, Matrikelnummer und die Nummer Ihres Tutoriums deutlich lesbar) ab, damit Ihr Tutor Korrekturhinweise und Ihre Punkte notieren kann. Werfen Sie es in die Holzkiste vor Raum 369 im Informatik-Hauptgebäude 50.34. Verwenden Sie ausschließlich das Deckblatt zur SWT1 aus ILIAS.

■ \Rightarrow nur das offizielle Deckblatt verwenden!

Allgemein

Geben Sie Ihre Lösung mit Deckblatt (mit Name, Matrikelnummer und die Nummer Ihres Tutoriums deutlich lesbar) ab, damit Ihr Tutor Korrekturhinweise und Ihre Punkte notieren kann. Werfen Sie es in die Holzkiste vor Raum 369 im Informatik-Hauptgebäude 50.34. Verwenden Sie ausschließlich das Deckblatt zur SWT1 aus ILIAS.

- \implies nur das offizielle Deckblatt verwenden!
- häufigster Fehler: Aufgaben nicht abgeben

Aufgabe 1 (Lastenheft): Ø 2,29 bzw. 3,44 von 5

- Unnötiges aus Vorlage durfte man löschen (z.B. “Siehe <https://en.wikibooks.org/wiki/LaTeX/Glossary>“ oder “Szenarien“)

Aufgabe 1 (Lastenheft): Ø 2,29 bzw. 3,44 von 5

- Unnötiges aus Vorlage durfte man löschen (z.B. “Siehe <https://en.wikibooks.org/wiki/LaTeX/Glossary>“ oder “Szenarien“)
- Anwendungsfälle beschreiben, was man in dem System tun kann
⇒ beinhalten immer Verben! (z.B. “Hobbyfotograph“ oder “JMJRST“ ist kein Anwendungsfall)

Aufgabe 2 (Klassendiagramm): \emptyset 2,54 bzw. 4,07 von 8

- oft wurden abstrakte Klassen “Maler-Filter” und “Kunst-Filter” vergessen

Aufgabe 2 (Klassendiagramm): Ø 2,54 bzw. 4,07 von 8

- oft wurden abstrakte Klassen “Maler-Filter” und “Kunst-Filter” vergessen
- es wurden Dinge ergänzt, die so nicht explizit im Text standen

Aufgabe 2 (Klassendiagramm): Ø 2,54 bzw. 4,07 von 8

- oft wurden abstrakte Klassen “Maler-Filter” und “Kunst-Filter” vergessen
- es wurden Dinge ergänzt, die so nicht explizit im Text standen
- Ausrichtung des Bildes wurde nicht als Enum modelliert

Aufgabe 2 (Klassendiagramm): Ø 2,54 bzw. 4,07 von 8

- oft wurden abstrakte Klassen “Maler-Filter” und “Kunst-Filter” vergessen
- es wurden Dinge ergänzt, die so nicht explizit im Text standen
- Ausrichtung des Bildes wurde nicht als Enum modelliert
- falsche UML-Syntax (insb. Methode, Attribute)

Aufgabe 3 (Durchführbarkeitsanalyse): \emptyset 0,81 bzw. 1,95 von 3

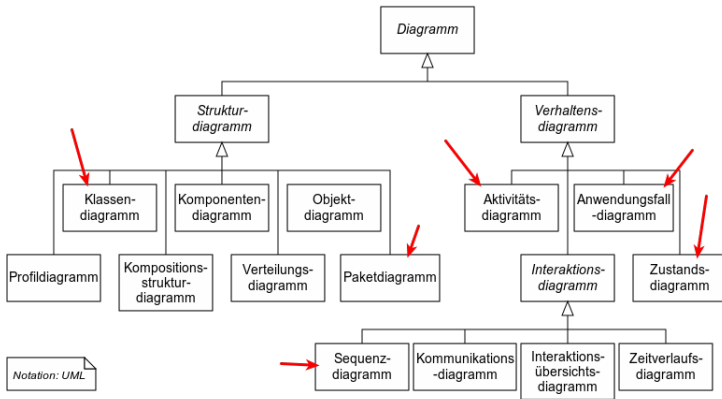
- sehr häufig nicht abgegeben

Aufgabe 3 (Durchführbarkeitsanalyse): Ø 0,81 bzw. 1,95 von 3

- sehr häufig nicht abgegeben
- Fragen beantworten, nicht stellen!
z.B. “Es werden 3 Java-Entwickler benötigt.“ ergänzen durch “Da wir 5 zur Zeit untätige Java-Entwickler in der Firma haben, ist das Projekt aus personeller Sicht für die Pear Corp. durchführbar.“

Wo sind wir? Pflichtenheft!

- ① Zielbestimmung
- ② Produkteinsatz
- ③ Produktumgebung
- ④ Funktionale Anforderungen
- ⑤ Produktdaten
- ⑥ Nichtfunktionale Anforderungen
- ⑦ Globale Testfälle
- ⑧ Systemmodelle
 - Szenarien
 - Anwendungsfälle
 - Objektmodelle \Rightarrow UML-Klassendiagramme (letztes mal)
 - **Dynamische Modelle**
 - UML-Zustandsdiagramm
 - UML-Aktivitätsdiagramm
 - UML-Sequenzdiagramm
- ⑨ Glossar

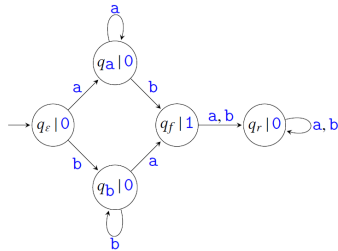


Wozu braucht man das?

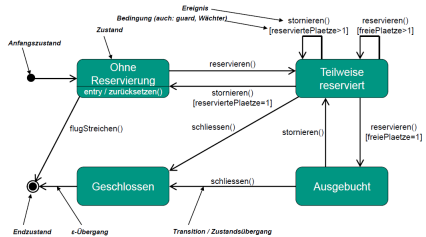
Wozu braucht man das?

- Zustand **eines Objektes** beschreiben
- Zustandsüberföhrungsfunktion?

Zustandsdiagramm \approx endlicher Automat

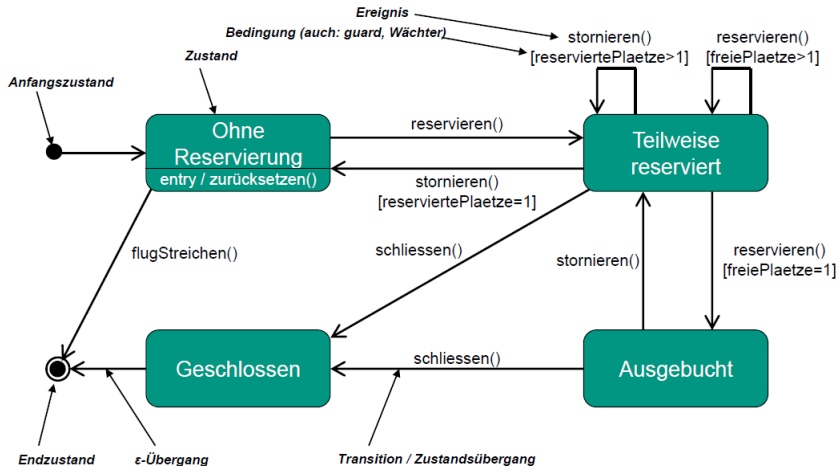


(a) GBI: DEA

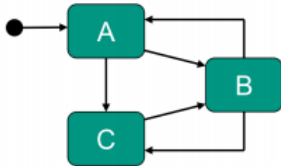


(b) SWT: Zustandsdiagramm

Zustandsdiagramm: Syntax

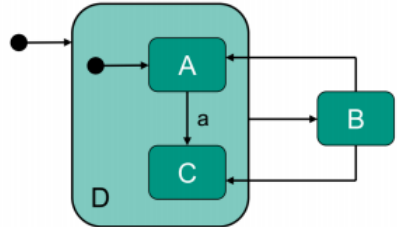


Zustandsdiagramm: Hierarchie



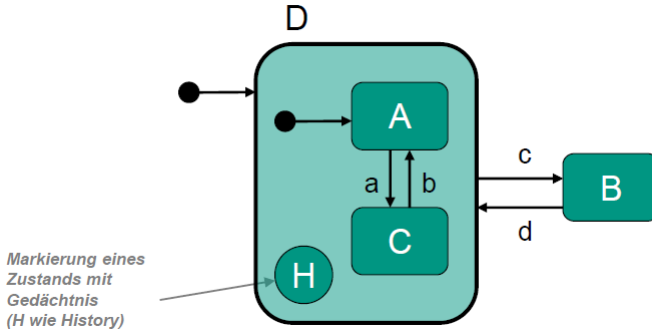
Ohne Hierarchie

\cong

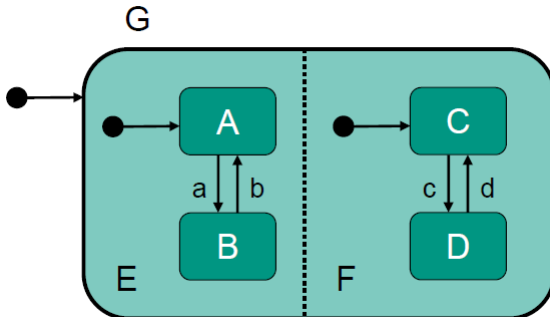


Mit Hierarchie

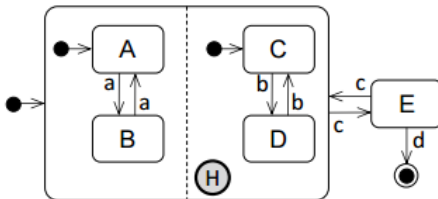
- History-Element, damit sich Hierarchie den letzten Zustand merkt



- mehrere Zustandsdiagramme in einem

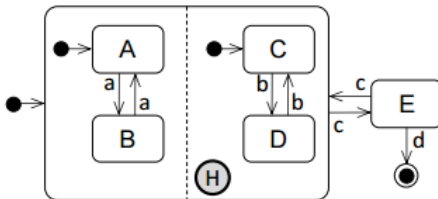


Gegeben ist der folgende UML-Zustandsautomat. Geben Sie an, in welcher Zustandskombination sich der Zustandsautomat, jeweils ausgehend vom Startzustand, nach den beiden Eingabefolgen befindet.



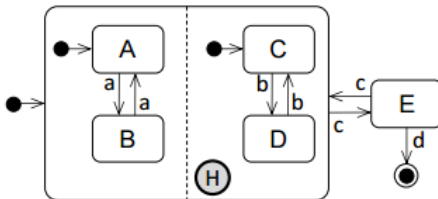
■ a, b, c, c

Gegeben ist der folgende UML-Zustandsautomat. Geben Sie an, in welcher Zustandskombination sich der Zustandsautomat, jeweils ausgehend vom Startzustand, nach den beiden Eingabefolgen befindet.



- a, b, c, c \implies AxD
- c, c, a, b, b, a, c, c, a

Gegeben ist der folgende UML-Zustandsautomat. Geben Sie an, in welcher Zustandskombination sich der Zustandsautomat, jeweils ausgehend vom Startzustand, nach den beiden Eingabefolgen befindet.



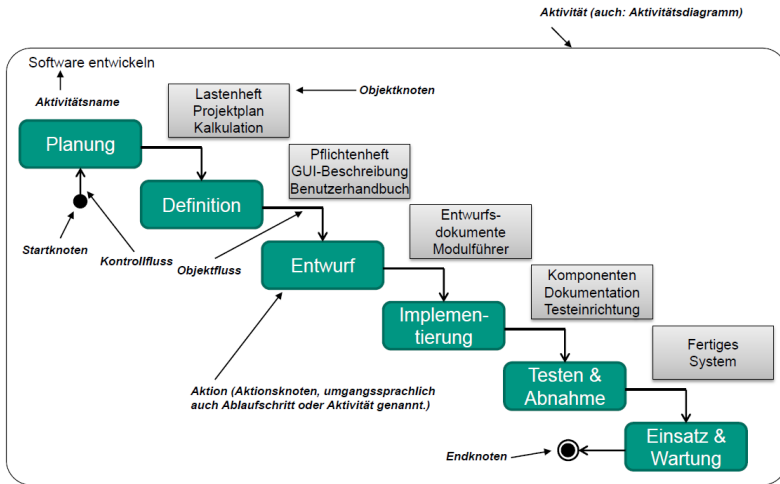
- $a, b, c, c \Rightarrow AxD$
- $c, c, a, b, b, a, c, c, a \Rightarrow BxC$

Wozu braucht man das?

Wozu braucht man das?

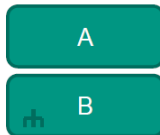
- Ablaufbeschreibungen (Kontrollfluss, Objektfluss)
- i.A. **mehrere verschiedene** Objekte

- ist ebenfalls nicht neues!



■ Aktionen

- Elementare Aktion
- Verschachtelte Aktion

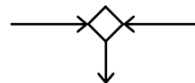
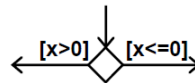


■ Knoten

- Startknoten
 - Startpunkt eines Ablaufs
- Endknoten
 - Beendet alle Aktionen und Kontrollflüsse
- Ablaufende
 - Beendet einen einzelnen Objekt- und Kontrollfluss



- Entscheidung
 - bedingte Verzweigung
- Zusammenführung
 - „oder“-Verknüpfung
- Teilung
 - Aufteilung eines Kontrollflusses
- Synchronisation
 - „und“-Verknüpfung



■ Objektknoten

- Eingabe- und Ausgabedaten einer Aktion
- Darstellung durch Stecker (engl. *pin*)

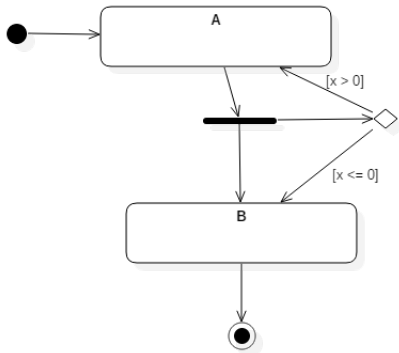


■ Alternative Darstellungen



- Start am Startknoten mit einer Marke
- Aktivitäten werden erst ausgeführt, wenn an jedem Eingang eine Marke anliegt
- wurde eine Aktivität ausgeführt, erscheinen an all ihren Ausgängen Marken

Wie kommt man hier zum Endknoten?

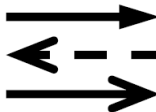


Wozu braucht man das?

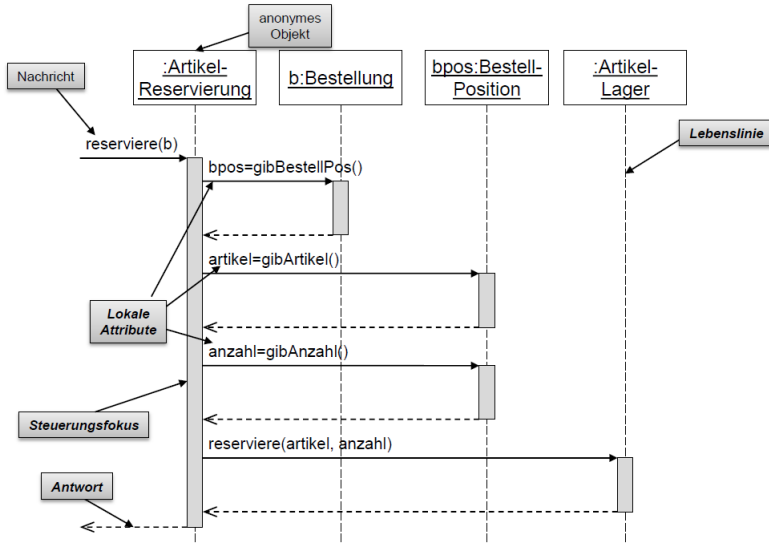
Wozu braucht man das?

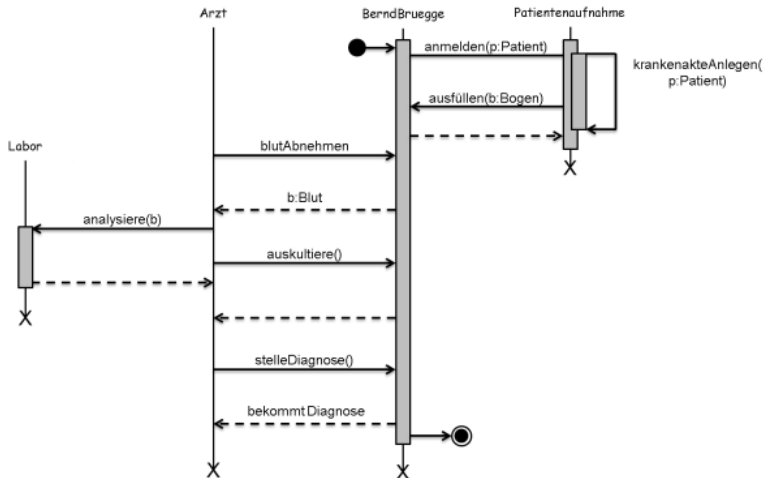
- stellt den möglichen Ablauf eines Anwendungsfalls dar
- **zeitlicher Verlauf** von Methodenaufrufen, Objekterstellung, Objektzerstörung

- Zeit verläuft von oben nach unten
- Lebenslinie
 - gestrichelte senkrechte Linie
 - eine pro Objekt
- Steuerungsfokus
 - dicker Balken über Lebenslinie
 - zeigt, dass Objekt gerade aktiv ist
- Nachrichtentypen
 - Synchrone Nachricht (blockierend)
 - Antwort (optional)
 - Asynchrone Nachricht



Sequenzdiagramm - Beispiel





Hier stimmt was nicht...

Aufgabe 1-3: Plug-In programmieren

Aufgabe 1-3: Plug-In programmieren

- JavaDoc + CheckStyle ...
- Falls ihr Tests schreibt, fügt junit in die jeweilige Untermodul-pom ein
- Java Swing benutzen (schaut euch die Java-Klassen JMenu und JMenuItem an)

Aufgabe 1-3: Plug-In programmieren

- JavaDoc + CheckStyle ...
- Falls ihr Tests schreibt, fügt junit in die jeweilige Untermodul-pom ein
- Java Swing benutzen (schaut euch die Java-Klassen JMenu und JMenuItem an)

Aufgabe 4: Aktivitätsdiagramm

Aufgabe 1-3: Plug-In programmieren

- JavaDoc + CheckStyle ...
- Falls ihr Tests schreibt, fügt junit in die jeweilige Untermodul-pom ein
- Java Swing benutzen (schaut euch die Java-Klassen JMenu und JMenuItem an)

Aufgabe 4: Aktivitätsdiagramm

- separate Diagramme \implies verschachtelte Aktionen

Aufgabe 5: Sequenzdiagramm

Aufgabe 5: Sequenzdiagramm

- auf welchen Objekten/Klassen werden Methoden aufgerufen?
- auf Pfeile `var=methode()` schreiben, wenn Rückgabe von `methode()` in `var` gespeichert wird

Aufgabe 5: Sequenzdiagramm

- auf welchen Objekten/Klassen werden Methoden aufgerufen?
- auf Pfeile `var=methode()` schreiben, wenn Rückgabe von `methode()` in `var` gespeichert wird

Aufgabe 6: Substitutionsprinzip

Aufgabe 5: Sequenzdiagramm

- auf welchen Objekten/Klassen werden Methoden aufgerufen?
- auf Pfeile `var=methode()` schreiben, wenn Rückgabe von `methode()` in `var` gespeichert wird

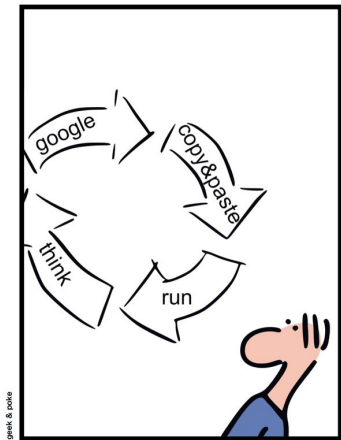
Aufgabe 6: Substitutionsprinzip

- Folien “Folgerung aus dem Substitutionsprinzip” anschauen (Ko-/Kontravarianz)
- mal als Java-Programm hinschreiben und versuchen zu kompilieren

Abgabe

- Deadline am 7.6 um 12:00
- Aufgabe 4+5 handschriftlich (auf saubere Syntax achten!)
- an das Deckblatt denken!!

SIMPLY EXPLAINED



DEVELOPMENT CYCLE