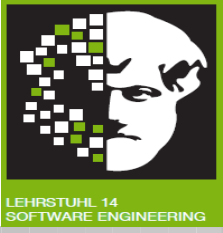
Willkommen zur Vorlesung
Softwarekonstruktion
im Wintersemester 2011/2012

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV



09. Wiederholung

[inkl Beiträge von Prof. Volker Gruhn]

Abschlussdiskussion

Kapitel 2: Allgemeine Prinzipien des SW-Engineering
Kapitel 3: Spezifikation im Allgemeinen
Kapitel 4: Algebraische Spezifikation inkl. JML
Kapitel 5: Petrinetze
Kapitel 6: Modellgetriebene SW-Entwicklung
Kapitel 7: Object Constraint Language (OCL)
Kapitel 8: Testen im Allgemeinen, Kontrollflussorientierte Testverfahren, Datenflussorientierte Testverfahren

- Welche Kapitel finden Sie wenig geeignet, auf den Beruf vorzubereiten ?
- Welche Inhalte der Veranstaltung entsprachen nicht Ihren Erwartungen ?
- Welche hätten Sie sich stattdessen gewünscht ?
- Welche Kapitel müssen Ihrer Meinung nach verbessert werden
  (mehr Erläuterungen, Beispiele etc.) ?
- Bei welchen Kapiteln haben Sie begleitende Literatur (Bücher etc.) vermisst ?

Abschlussdiskussion

- Länge der Vorlesung (120 min inkl 5 min Pause):

  ok – zu lang

- Termin (nachmittags 16.00-18.00):

  ok – besser 10.00-12.00 – besser 14.00-16.00

- Sonstige Hinweise / Tipps / Anregungen / Wünsche:

Deadlines Klausuranmeldung

LEHRSTUHL 14
SOFTWARE ENGINEERING

30. Januar 2012: Abgabe der letzten Hausübung

30. Januar 2012: Deadline Anmeldung **Studienleistung**

6./7. Februar 2012: Eintrag der erfolgreich absolvierten Studienleistungen über BOSS

12. Februar 2012: Deadline Anmeldung **Klausur** (Bachelor: über BOSS; Diplom: wenn möglich über BOSS, sonst per email an sebastian.pape@cs.tu-dortmund.de)

21. Februar 2012: Klausur

26. März 2012: Nachklausur

Die Deadlines werden durch das zentrale Anmeldungssystem vorgegeben und sind daher leider nicht verlängerbar.

Termine für die **Klausureinsichten** und die Deadline der Anmeldung für die **Nachklausur** werden über unsere Webseite bekannt gegeben

Werbe-Block: Hiwi-Tätigkeiten, Abschlussarbeiten

Es gibt verschiedene Möglichkeiten für eine Beschäftigung als Hiwi am Fraunhofer ISST oder am LS 14 / TUD:

- Unterstützung der folgenden Projekte (bespielsweise durch Java-Programmierung eines UML-Analyse Werkzeuges oder konzeptuelle Arbeiten im Bereich modell-basierte Sicherheitsanalyse): "Secure Change", "Architectures for Auditable Business Process Execution (APEX)", „SecureClouds“, „ClouDAT“
- Unterstützung in der Lehre (Tutorien, Folienerstellung etc)

Abschlussarbeiten können in inhaltlicher Beziehung zu einer Hiwi-Tätigkeit am Fraunhofer ISST oder LS 14 / TUD durchgeführt werden.

Sie können insbesondere in Zusammenhang mit Anwendungsprojekten am ISST durchgeführt werden, wodurch sich vielfältige Möglichkeiten zu Kooperation mit Unternehmen ergeben, zB:

- Apex: Versicherungen / Banken (Münchener Rückversicherung, Signal Iduna, Wüstenrot), Softwarehersteller (SAP, IDS Scheer)
- Secure Change: Telekom / Smartcards (Telefonica, Gemalto)
- Csec: Microsoft Research Cambridge
- Secure Clouds / ClouDAT: Cloud-Software-Anbieter (LinogistiX), IT-Berater (Admeritia, ITESYS, TÜV-IT)

Informationen unter: http://jan.jurjens.de/jobs/hiwis.html

Werbe-Block: Weitere relevante Lehrveranstaltungen

SS 2012:

- Methodische Grundlagen des Software Engineering (Master-Basismodul Software) (4+2 SWS)
- Modellbasierte Softwaretechniken für sichere Systeme (Spezial-Vorlesung Diplom/Master + Übung (2+2 SWS))
- Seminar „Ausgewählte Themen des Modellbasierten Sicherheits-Engineerings“ (2 SWS)

Zuordnung der Wahlveranstaltungen zu Schwerpunktgebieten (Diplom):

- Sicherheit und Verifikation
- Software-Konstruktion

Forschungsbereich Master: Software, Sicherheit und Verifikation

Informationen unter: http://jan.jurjens.de/teaching

Themen der Vorlesung

Kapitel 1: Intro mit Vorstellung der Professur und Gliederung der Vorlesung

Kapitel 2: Allgemeine Prinzipien des SW-Engineering

Kapitel 3: Spezifikation im Allgemeinen

Kapitel 4: Algebraische Spezifikation (inkl. Tutorial zu JML)

Kapitel 5: Petrinetze

Kapitel 6: Modellgetriebene SW-Entwicklung

Kapitel 7: Object Constraint Language (OCL)

Kapitel 8: Testen im Allgemeinen, Kontrollflussorientierte Testverfahren, Datenflussorientierte Testverfahren

Kap. 1: Ziele, Motivation

1 Zielsetzung der Vorlesung

1.1 Ziele

1.2 Motivation

- Begriffsbildung
- Relevanz
- Defizite
- Einordnung in die Informatik
- Phänomene des Software Engineerings
- Software-Krise

Kap. 2: Prinzipien des SW-Engineering

- 40 Jahre Software Engineering
- Software Engineering - Wann braucht man‘s?
- Eigenschaften von Software
- Allgemeine Prinzipien des Software-Engineering

- Allgemeine Modellierungskonzepte, Modellierung
- Was bedeutet Spezifizieren?

Algebraische Spezifikation

algebra X

introduces sorts Y,Z;

Sorten

operations

op1 : X -> X

op2 : X -> Y

Operationen

constraints op1, op2 **so that for all**

x,y : X, z:Y

...

Aussagen

end X;

Signatur und Algebra

Signatur $\Sigma=(S,F)$

Sorten ↗ ↖ Operationssymbole

(Namen für Mengen)

Stelligkeit, Vor- und Nachbereich für Operationssymbole:

$f: s_1,\ldots,s_n \in S^* \rightarrow s \in S$

Algebra $A=((A_s)_{s\in S}, (f_A)_{f\in F})$

ordnet den Namen für Mengen konkrete Mengen zu

ordnet den Operationssymbolen konkrete Abbildungen zwischen den vorne bestimmten Mengen zu

Σ-Homomorphismus – Beziehung zwischen zwei Algebren

Σ-Algebra: Beispiel

Signatur Σ_{Test}	Σ_{Test}-Algebra A
Nat	$A_{Nat} =_{def} \mathbb{N}$
Bool	$A_{Bool} =_{def} \{true, false\}$
zero:$\rightarrow$ Nat	$zero_A =_{def} 0$
one: $\rightarrow$ Nat	$one_A =_{def} 1$
succ : Nat $\rightarrow$ Nat	$\forall n \in \mathbb{N}.\ succ_A(n) =_{def} n + 1$
add : Nat x Nat $\rightarrow$Nat	$\forall n,m \in \mathbb{N}.\ add_A(n,m) =_{def} n + m$
equal : Nat x Nat $\rightarrow$ Bool	$equal_A (zero_A, zero_A) =_{def} true$ $\forall n,m \in \mathbb{N}.\ equal_A(succ_A(n),succ_A(m)) =_{def} equal_A(n,m)$ $\forall n,m \in \mathbb{N}.\ equal_A (n,m) =_{def} equal_A (m,n)$ $\forall m \in \mathbb{N}.\ equal_A (zero_A, add_A (one_A,m)) =_{def} false$

Anmerkung: Die Axiome definieren jeweils eindeutig eine Funktion (müsste genaugenommen jeweils bewiesen werden).

Homomorphismus: Beispiel

Σ-Algebra $A = (A_s, f_A)$

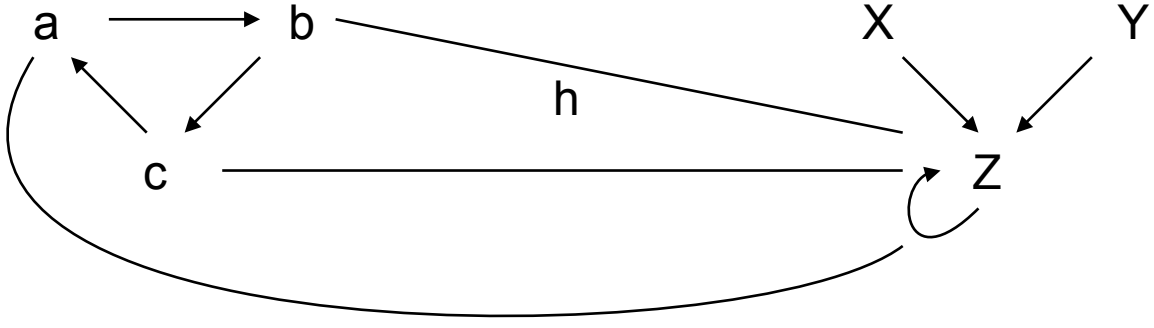
$A_s = \{a,b,c\}$

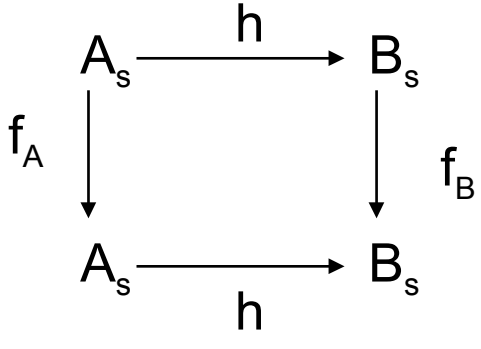
$f_A = \{a \to b, b \to c, c \to a\}$

Σ-Algebra $B = (B_s, f_B)$

$B_s = \{X,Y,Z\}$

$f_B = \{X \to Z, Y \to Z, Z \to Z\}$





h Homomorphismus-Abbildung

- Wird a auf X abgebildet, so muss auch das Bild von a (nämlich b) auf das Bild von X (nämlich Z) abgebildet werden, usw.
- Damit gibt es nur einen Homomorphismus, nämlich den, der a,b,c auf Z abbildet.

Homomorphismus: Beispiel Nat

Signatur $\Sigma_{Nat'}$	$\Sigma_{Nat'}$-Algebra A	$\Sigma_{Nat'}$-Algebra A2
Nat	$A_{Nat} =_{def} \mathbb{N}$	$A2_{Nat} =_{def} \{0,2,4,...\}$
Bool	$A_{Bool} =_{def} \{true, false\}$	$A2_{Bool} =_{def} \{true, false\}$
one: $\rightarrow$ Nat	$one_A =_{def} 1$	$one_{A-2} =_{def} 2$
succ : Nat $\rightarrow$ Nat	$succ_A (n) =_{def} n + 1$	$succ_{A-2} (n) =_{def} n + 2$
add : Nat x Nat $\rightarrow$ Nat	$add_A (n,m) =_{def} n + m$	$add_{A-2} (n,m) =_{def} n + m$
T: $\rightarrow$ Bool	$T_A =_{def}$ true	$T_{A-2} =_{def}$ true

A und A2 sind $\Sigma_{Nat'}$-homomorph, wenn ein $\Sigma_{Nat'}$-Homomorphismus existiert.

$h_{Nat} : A_{Nat} \rightarrow A2_{Nat}$, $n \rightarrow 2n$

$h_{Bool}: A_{Bool} \rightarrow A2_{Bool}$, $b \rightarrow b$

Homomorphismus: Beispiel Nat

Ein Homomorphismus $h: A \rightarrow A2$ besteht aus den Abbildungen $h_{Nat} : A_{Nat} \rightarrow A2_{Nat}$ und $h_{Bool} : A_{Bool} \rightarrow A2_{Bool}$, d.h. $h = (h_{Nat}, h_{Bool})$ mit

- $h_{Nat}\ (one_A) = one_{A\text{-}2}$ $\quad [\Leftrightarrow h_{Nat}\ (1) = 2]$
- $h_{Bool}\ (T_A) = T_{A\text{-}2}$ $\quad [\Leftrightarrow h_{Bool}\ (true) = true]$
- $\forall\ x,y \in A_{Nat} . h_{Nat}\ (add_A(x,y)) = add_{A\text{-}2}(h_{Nat}(x), h_{Nat}(y))$

  Beispielhaft einsetzen:
  - $x=1,y=2 : h_{Nat}\ (add_A(1,2)) = add_{A\text{-}2}(h_{Nat}(1), h_{Nat}(2))$

    $\Leftrightarrow\ h_{Nat}\ (1+ 2) = add_{A\text{-}2}(2, 4)$

    $\Leftrightarrow\ 2 * 3 = 2 + 4$

  $\Rightarrow$ Überprüfung für alle Konstanten und für alle Operationssymbole !

Terme

∀ Σ=(S,F,X) – Signatur mit Variablen

- Eine Familie $X = (X_s)_{s \in S}$ von Mengen,
  die durch S indiziert ist, heißt S-sortiert (S-indiziert).
  Deren Elemente heißen Variablen.

Menge der Variablen einer bestimmten Sorte

Namen für Sorten (entspr. "Typen")

Σ-Terme: Durch Operationen verknüpfte Variable (unter Beachtung der jeweiligen Stelligkeiten, Vor- und Nachbereiche der Operationen)

Werte von Termen

- ass ordnet jedem Variablennamen ein Element der "passenden" Menge A_s zu

  Variablenbelegung

- xeval(ass): $T_\Sigma(X) \rightarrow A$

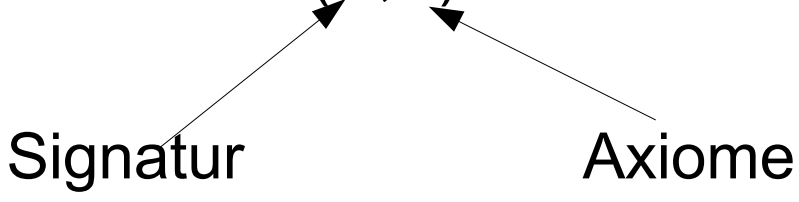
  Wert eines Terms bei der durch ass vorgegebenen Variablenbelegung

Gleichungen

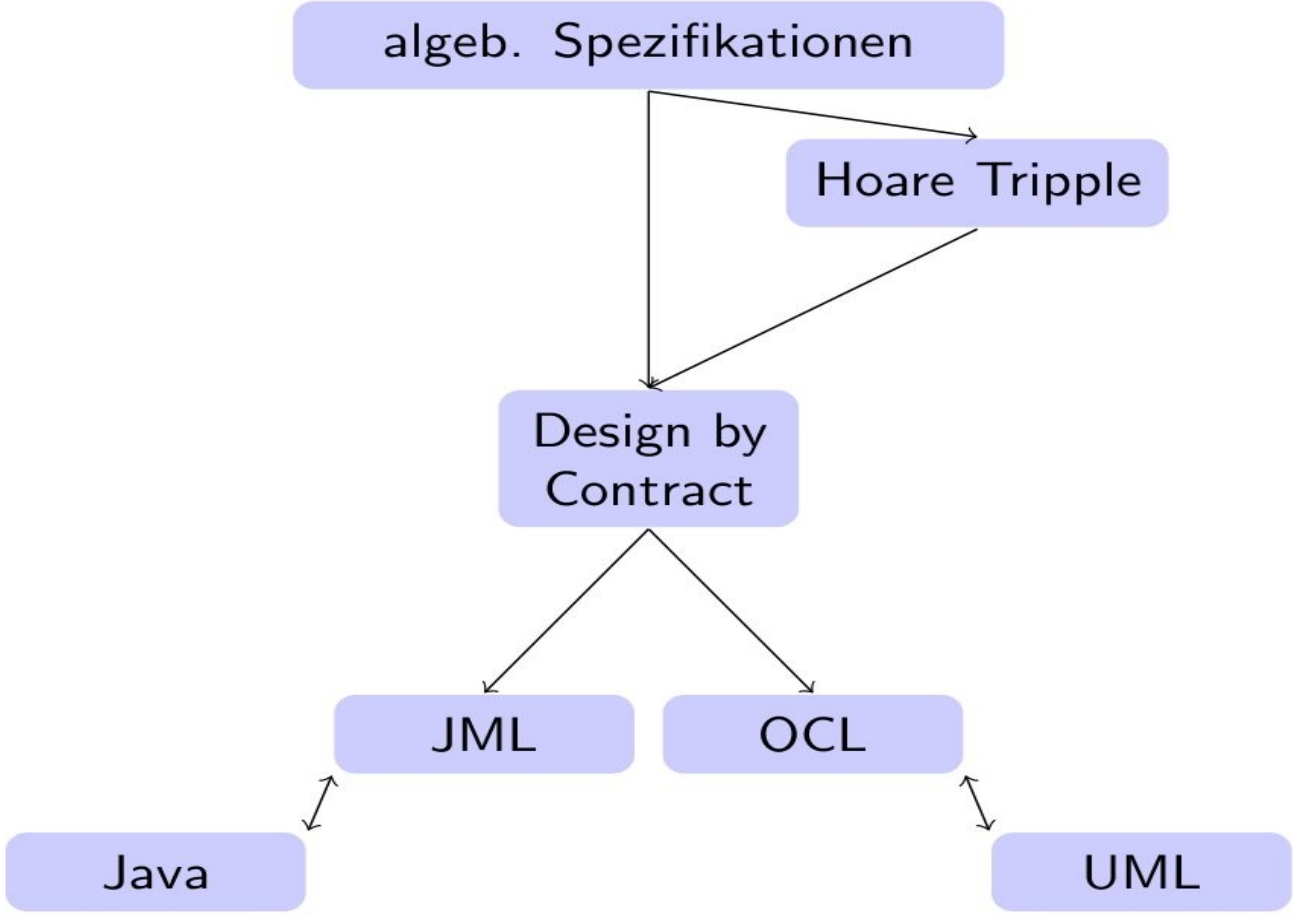
- Sei A eine Algebra mit der Signatur $\sum$ =(S, F,X).
  Für t, t‘$\in$ T ($\sum$,X) s $\in$ S heißt
  - e ≡ t =s t‘ eine $\sum$-Gleichung.
    Gleichheit zweier (typgleicher) Terme

- wohldefinierte Formeln
- WFF($\sum$) – Aussagen über solche Gleichungen

- Algebraische Spezifikation SP=($\sum$,E)

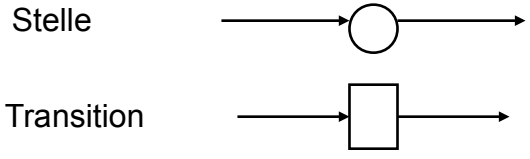


Übersicht - Inhaltliche Abhängigkeiten

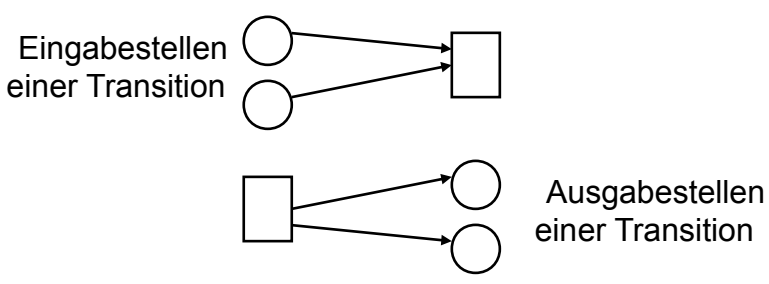


Petrinetze

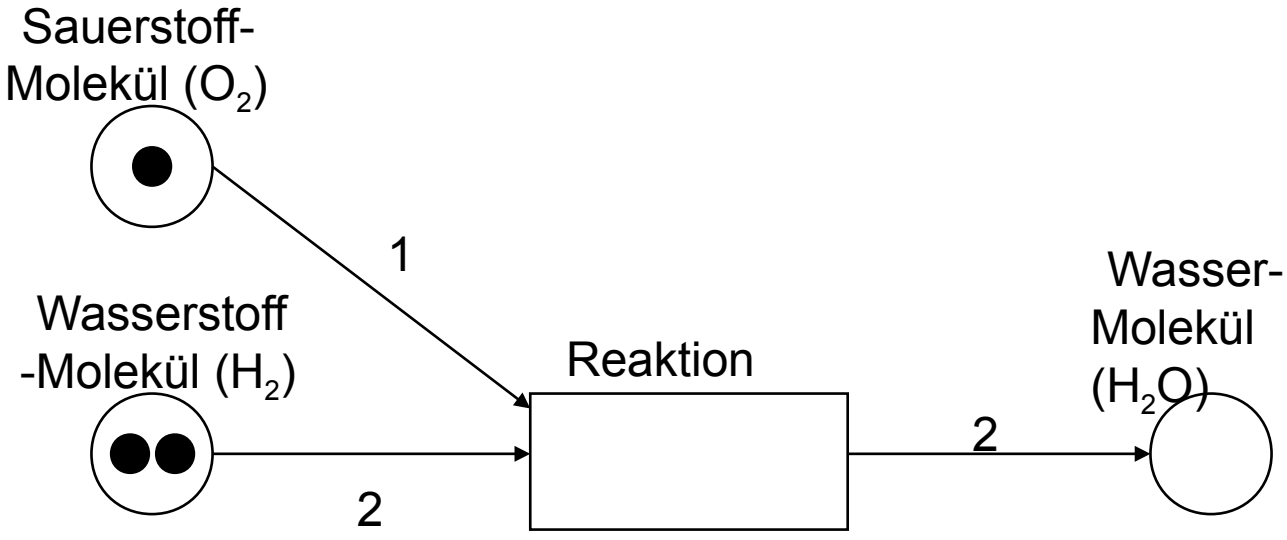
- Petrinetz
  - Ist ein gerichteter Graph, der aus zwei verschiedenen Sorten von Knoten (Stellen, Transitionen) besteht (bipartiter Graph)
    - Stelle: Zwischenablage von Informationen
    - Transitionen: Verarbeitung von Informationen



  - Kanten verbinden Stellen mit Transitionen, aber nie Stellen mit Stellen oder Transitionen mit Transitionen
  - Stellen werden mit Objekten (sog. Tokens oder Marken) belegt
  - Durch den Durchlauf der Tokens durch das Petri-Netz wird das dynamische Verhalten des Systems beschrieben



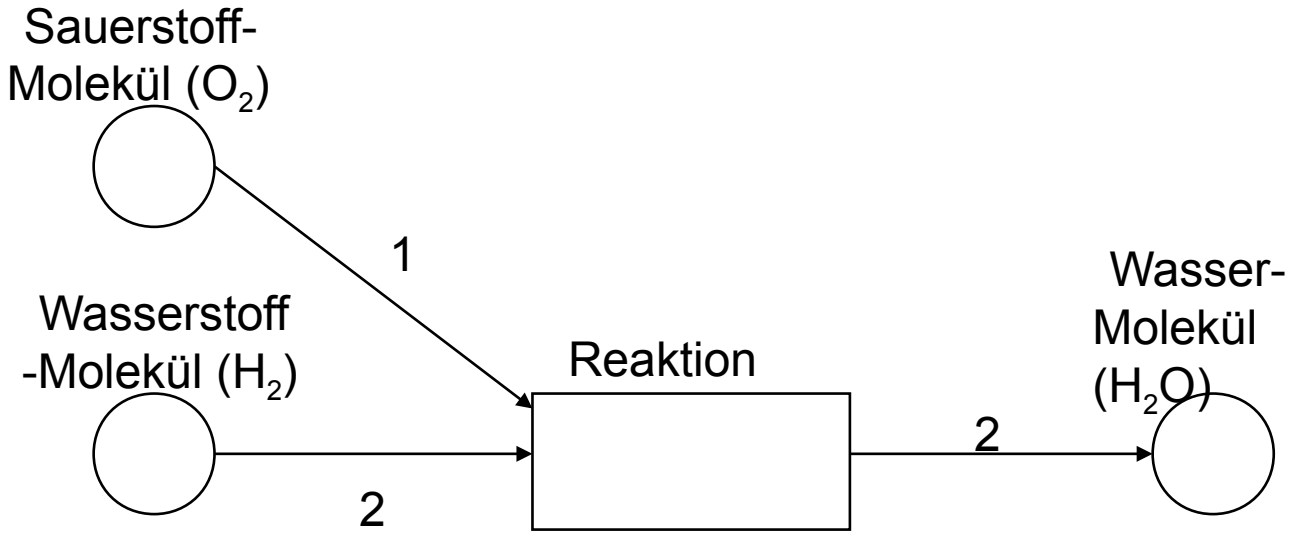
Interpretation



Zustand eines Systems,

beschrieben durch Marken

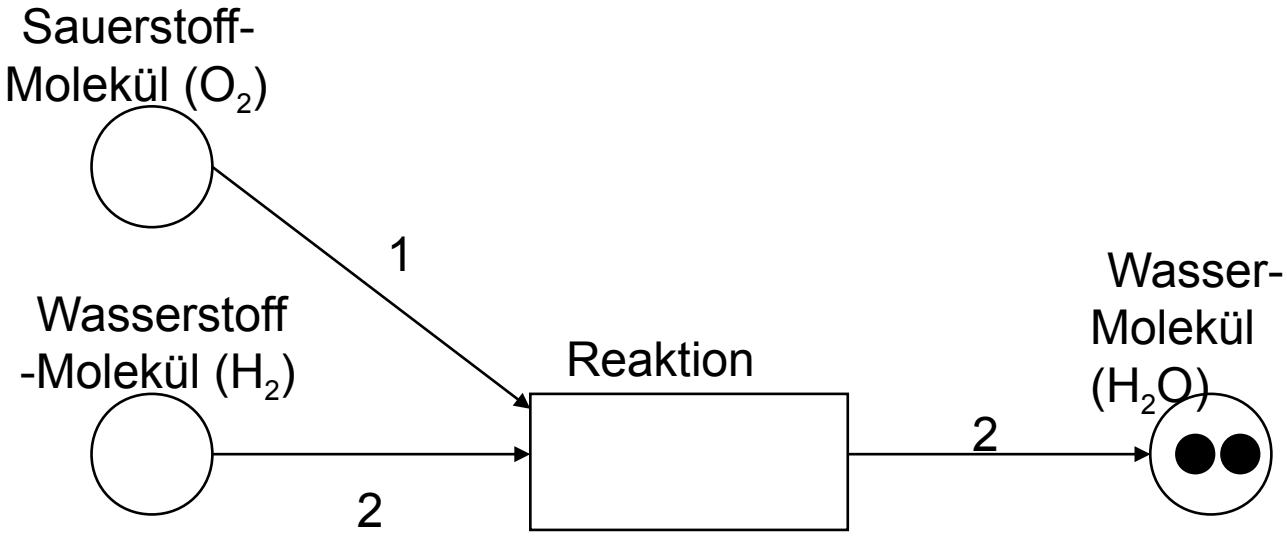
Interpretation



etwas passiert…

Zustandsänderung (Transition)

Interpretation



Folgezustand

Petrinetz beschreibt Ursache und Wirkung / mögliche Abläufe eines Systems

Stellen/Transitions-Netze

- Marken sind nicht unterscheidbar
- Stellen können mehr als eine Marke enthalten
- Kanten haben Gewichte
- Beim Schalten wird den Stellen des Vorbereichs/des Nachbereichs der aktivierten Transitionen soviele Tokens entnommen/hinzugefügt, wie das Gewicht der Kante anzeigt
- Stellen mit Kapazitäten (max. Tokenmenge)
- Es darf nur geschaltet werden, wenn Kapazität der jeweiligen Stelle nicht überschritten wird

Stellen/Transitions-Netze

- Ein 6-Tupel (S,T,F,K,W,M_0) heißt **Stellen-/Transitionen-Netz (S/T-Netz)**, falls gilt:

  (i) (S,T,F) ist ein Netz aus **Stellen** S und **Transitionen** T

  (ii) $K: S \rightarrow \mathbb{N} \cup \{\infty\}$ erklärt eine (möglicherweise unbeschränkte) **Kapazität** für jede Stelle.

  (iii) $W: F \rightarrow \mathbb{N}$ bestimmt zu jedem Pfeil des Netzes ein **Gewicht**.

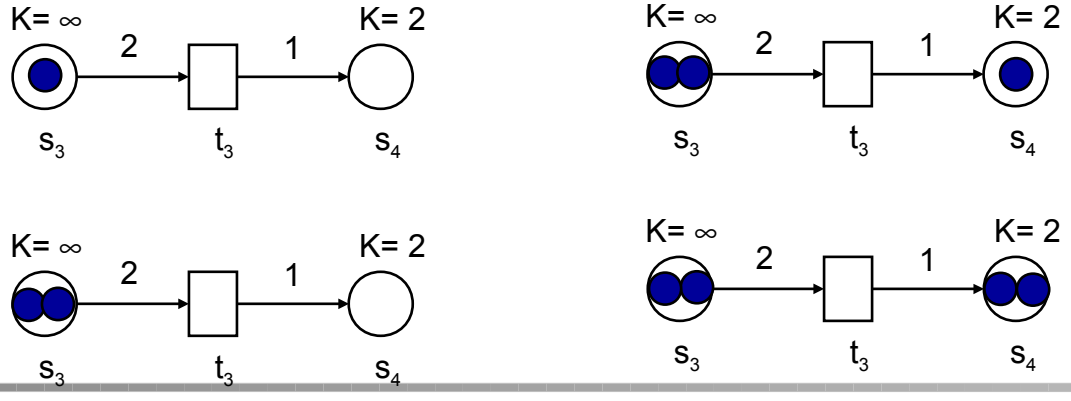
  (iv) $M_0: S \rightarrow \mathbb{N}_0$ ist eine **Anfangsmarkierung**, die die Kapazitäten respektiert, d.h. für jede Stelle $s \in S$ gilt: $M_0(s) \leq K(s)$.

- Ein Netz $N=(S,T,F,K,W,M_0)$ wird auch mit (N,M_0) bezeichnet

Stellen/Transitions-Netze

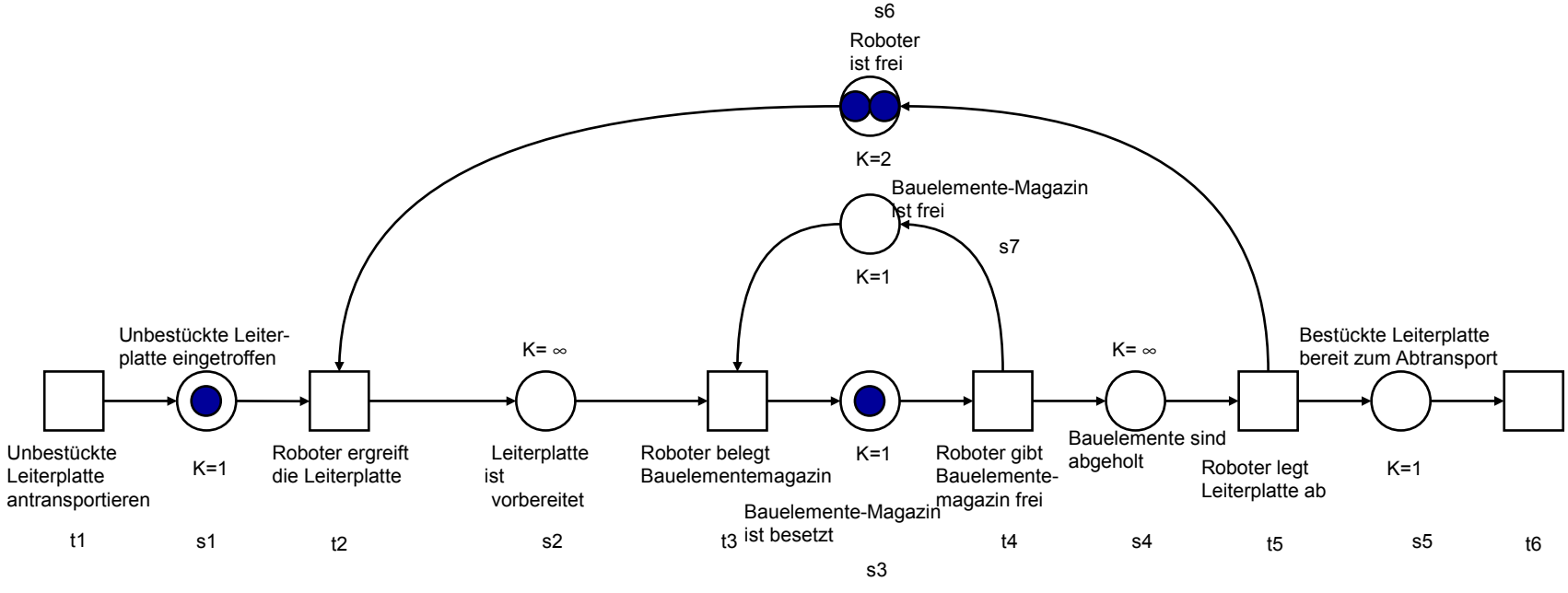
- Sei N ein S/T-Netz.
  - Eine Abbildung M:S $\rightarrow \mathbb{N}_0$ heißt **Markierung von N**, mit $\forall s \in S$:
    - $M(s) \leq K(s)$.
  - $\forall$ M(N) ist die Menge aller Markierungen von N

- Eine Transition $t \in T$ heißt **M-aktiviert** (schreibe M[t>), falls gilt:
  - $\forall$ $\forall s \in {\bullet t}$: $M(s) \geq W(s,t)$
  - $\forall$ $\forall s \in t{\bullet}$: $M(s) \leq K(s) - W(t,s)$



Erreichbarkeitsmenge im Beispiel

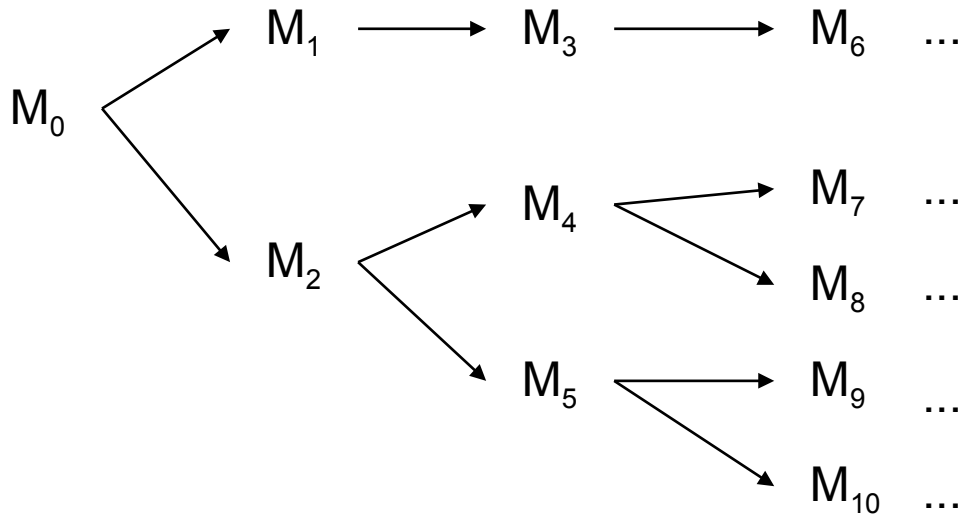
Nr.	s1	s2	s3	s4	s5	s6	s7	Schaltungen
M0	1	0	1	0	0	2	0	t2->M1 t4->M2
M1	0	1	1	0	0	1	0	t4->M3
M2	1	0	0	1	0	2	1	t2->M4 t5->M5



Erreichbarkeitsmenge im Beispiel

Nr.	s1	s2	s3	s4	s5	s6	s7	Schaltungen
M0	1	0	1	0	0	2	0	t2->M1 t4->M2
M1	0	1	1	0	0	1	0	t4->M3
M2	1	0	0	1	0	2	1	t2->M4 t5->M5
M3	0	1	0	1	0	1	0	t5->M6
M4	0	1	0	1	0	1	1	t3->M7 t5->M8
M5	1	0	0	0	1	2	1	t2->M9 t6->M10
M6	0	1	0	0	1	1	0	t6->M11
…								

Erreichbarkeitsmenge im Beispiel



Grundsituation in Petrinetzen

- Konflikt
- Synchronisation
- Kontakt
- Konfusion

- Wie sind sie charakterisiert?
- Was bedeuten sie?
- Wie kann man sie beheben/vermeiden/mit ihnen umgehen?

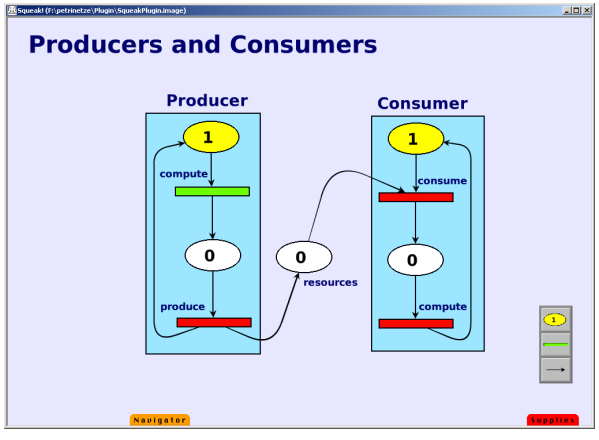
LEHRSTUHL 14
SOFTWARE ENGINEERING

Analyse von Netzen

- Aussagen zu Eigenschaften von Transitionen und Netzen:
- Sicherheit
  (Beschränkung der Zahl der Marken)
- Lebendigkeit
  ("Es kann etwas passieren", d.h. Transitionen aktivierbar)
- Synchronie
  (Verhältnis zwischen Ereignissen)

Modellierung von Petrinetzen

- Empfohlene Übungen
  - Zeichnen von Netzen
  - Verhalten von Netzen unter einer Startmarkierung bestimmen
  - Aussagen über Netze treffen bzw. Netze analysieren

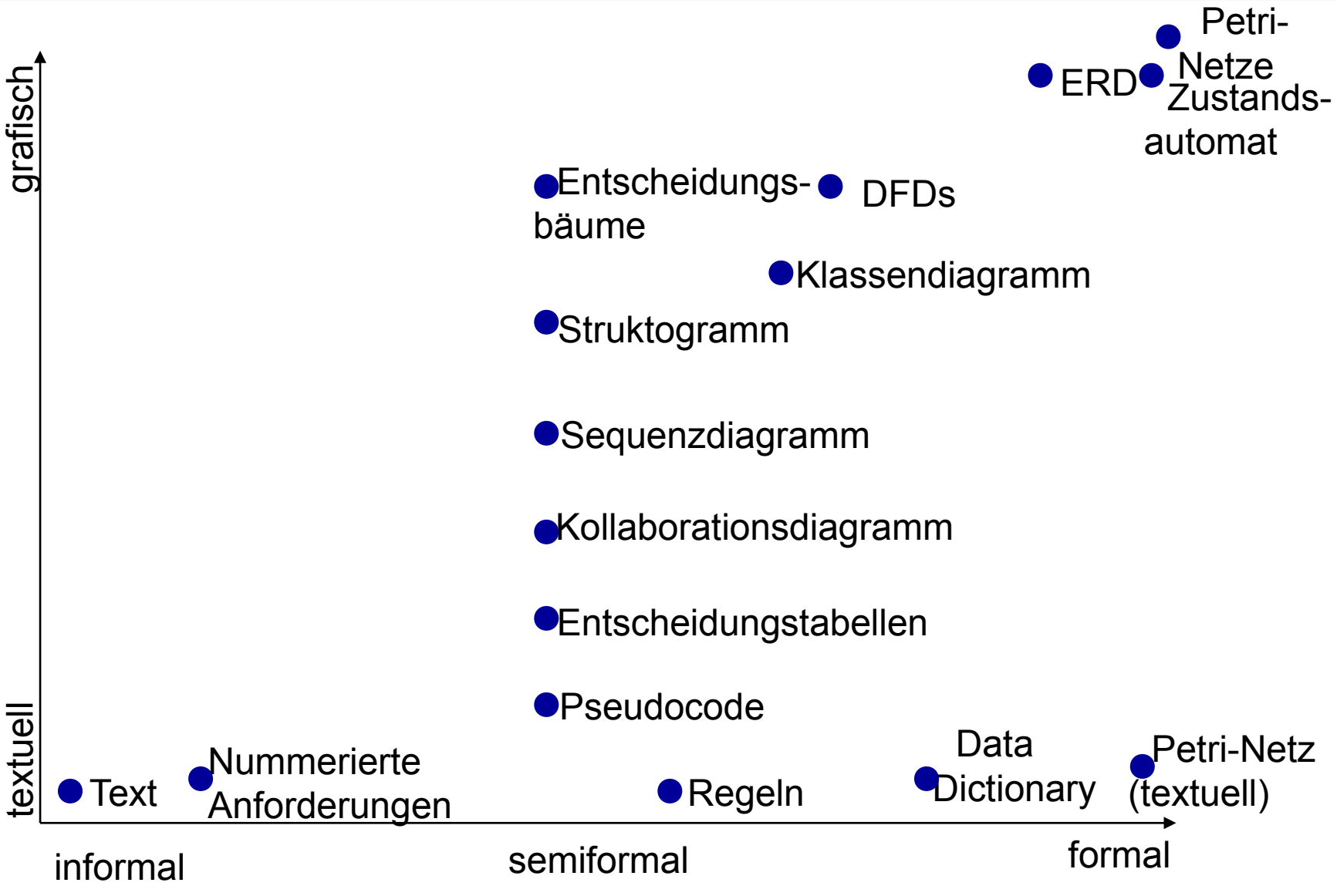


Tool für Petri-Netze
http://scg.unibe.ch/download/petitpetri/

Modellbasierte SW-Entwicklung

- Kernideen der modellbasierten SW-Entwicklung
  - Auch: Modellbasiert, Modellzentriert, Model-driven, Model-based, MDA
  - Modelle sind zentrales Artefakt im SW-Prozess
    - Konsequente Nutzung von erster bis zur letzen Phase des Lebenszyklus zur letzten (Anforderung bis Wartung)
    - Vermeidung von Modellbrüchen im SW-Prozess
  - Einsatz von Softwaremodellen mit fachlicher Semantik
    - Fachliche Anforderung von konkreter Technologie entkoppeln
    - Wiederverwendung fachlicher Aspekte
- Verwendung von Modellen
  - Generierung von Programmcode (Automatisierung)

Einordnung der Modellierungskonzepte



Semantik von Modellen

- Syntax und Semantik von Modellen
  - Formales Modell
    - Syntax und Semantik sind auf mathematischer oder streng logischer Struktur definiert (Algebren, Petrinetze)
  - Semi-Formales Modell
    - Syntax ist präzise definiert.
    - Syntax und Semantik sind nicht komplett formal definiert.
    - Wichtige Teile sind jedoch formal definiert.
  - Modell mit freier Semantik / Skizze
    - Syntax und Semantik sind nicht, oder nur durch natürliche Sprache definiert (Skizzen)

Model und Metamodell der UML

- Metamodell
  - „meta“ bedeutet soviel wie „über“
  - Metamodelle sind Modelle, die Modelle beschreiben
  - Definition aller Elemente einer Modellierungssprache und ihre Beziehungen untereinander

Modell	*Instanz*
XML-Schema	XML-Datei
Grammatik	Programmiersprache Java
Definition S/T-Netz	S/T-Netz Bestückungsroboter
UML-Klasse	Objekt
Metamodell	Modell

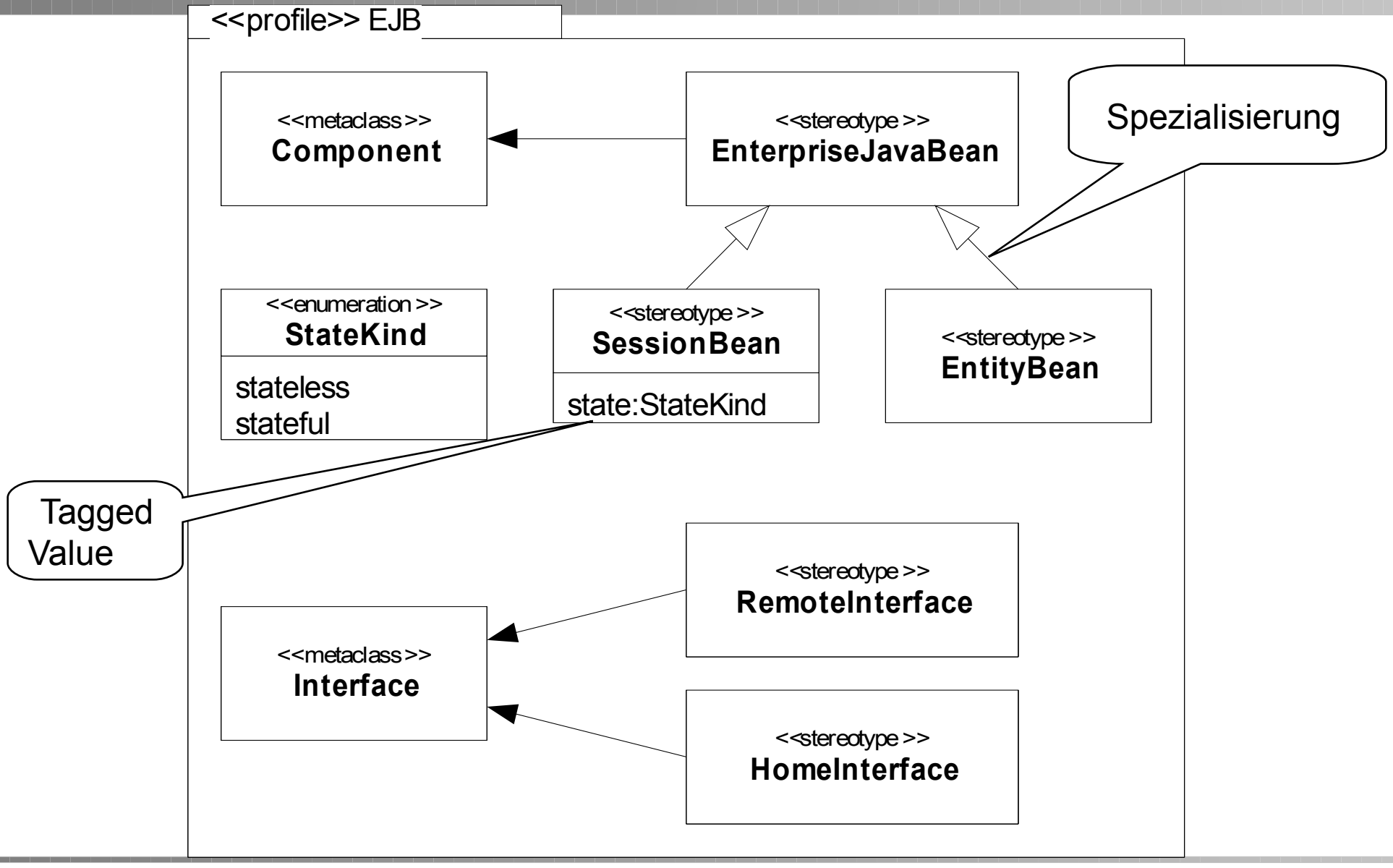
UML-Profile

- UML-Profil
  - Spezialisierung von Standard UML-Elementen zu konkreten Metatypen
  - Ein Profil kann zu einem Modell hinzugefügt werden und steht dann im gesamten Modell zur Verfügung
  - Verschiedene Profile für verschiedene Anwendungsdomänen
  - Einige Profile sind bereits vordefiniert und bei der OMG verfügbar

- Definition eines Profils
  - Paket von Stereotypen und Tagged Values
  - Klassendiagramm definiert Beziehungen zwischen neuem Stereotyp und dem zu beschreibenden Element definiert
  - Für die Definition neuer Stereotypen ist eine Grundkenntnis des Metamodells erforderlich

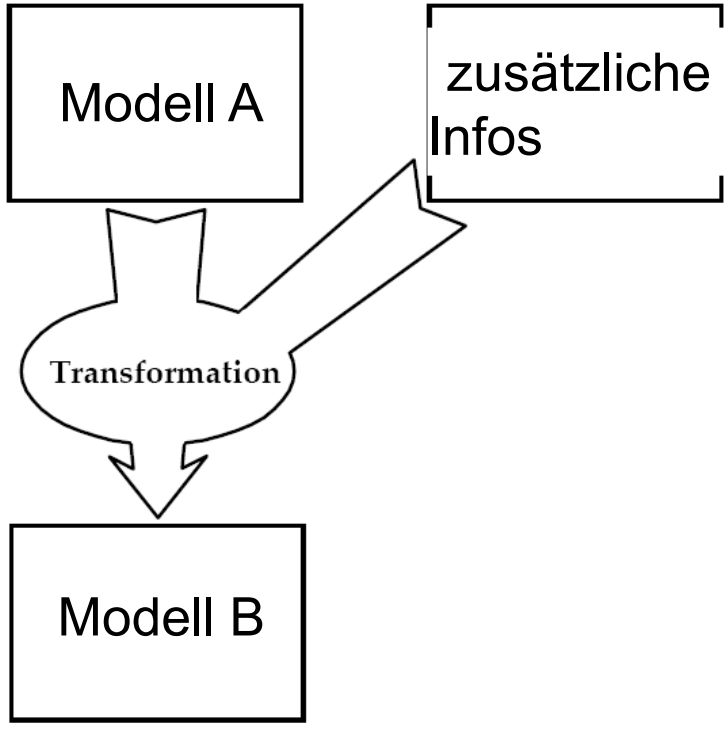
UML-Profile - Beispiel

LEHRSTUHL 14
SOFTWARE ENGINEERING



Modelltransformation

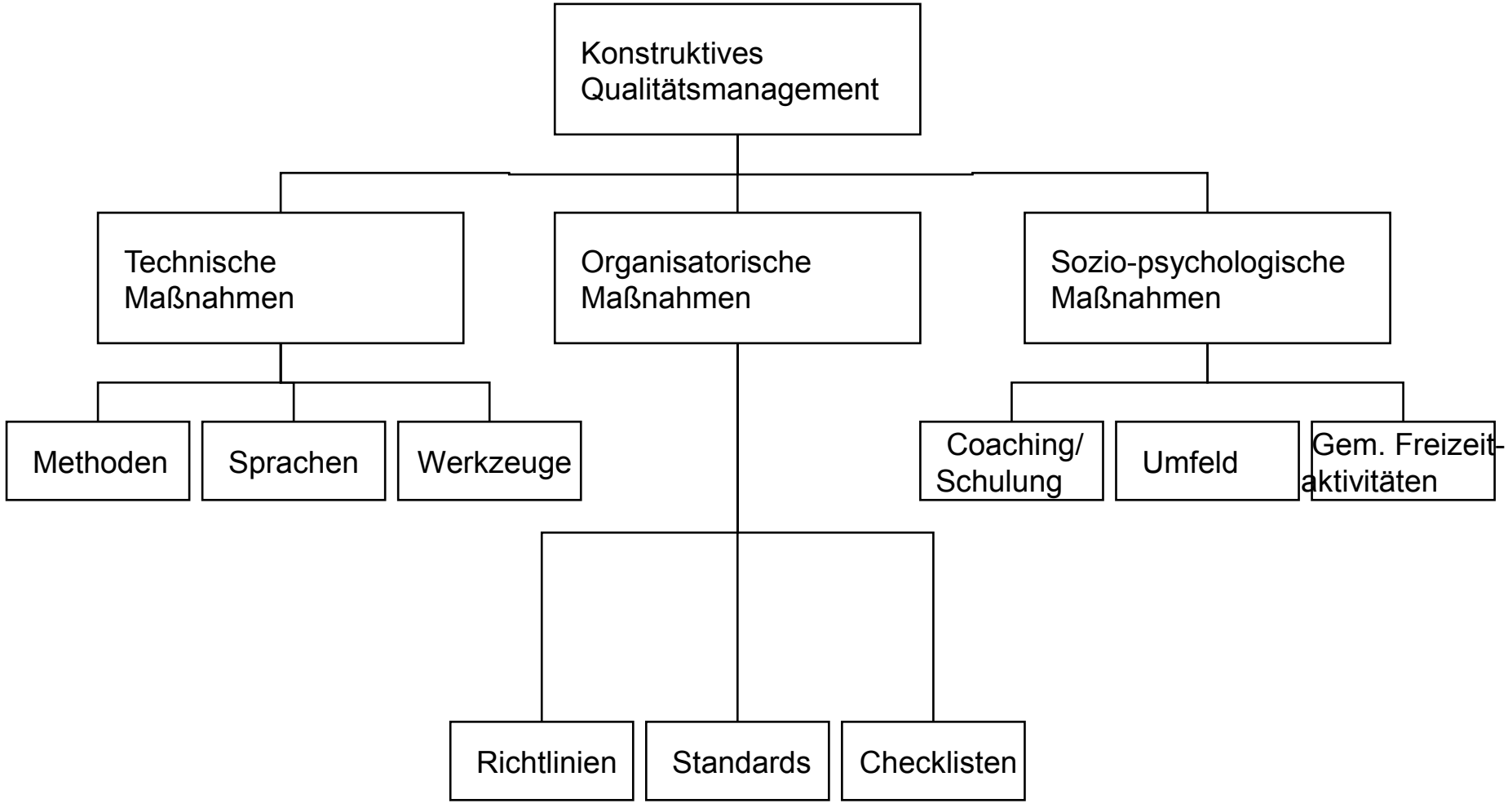
- Model A und zusätzliche Information werden durch die Anwendung einer Transformation in das Zielmodell B überführt
- Hintereinanderausführung mehrerer Transformationen sind möglich
- Prinzipiell könne alle Modelle die man definieren kann Quell- oder/und Zielmodell sein.
- Viel Know-How steckt im Generator



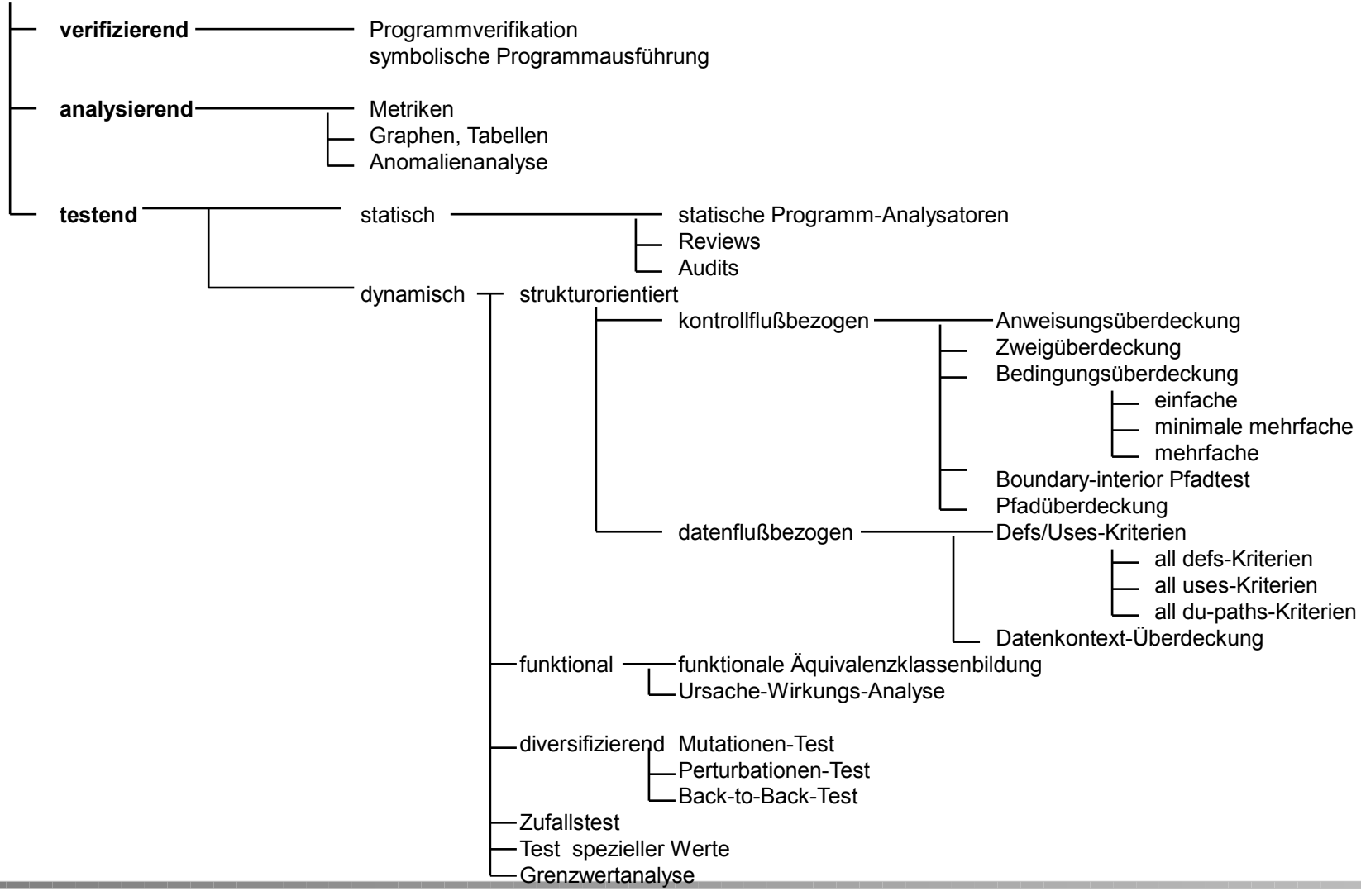
Quelle: MDA Guide Version 1.0.1

Operation	Description
hasReturned() : Boolean	True if type of template parameter is an operation call, and the called operation has returned a value.
result()	Returns the result of the called operation, if type of template parameter is an operation call, and the called operation has returned a value.
isSignalSent() : Boolean	Returns true if the OclMessage represents the sending of a UML Signal.
isOperationCall() : Boolean	Returns true if the OclMessage represents the sending of a UML Operation call.
parameterName	The value of the message parameter.

Übersicht über das konstruktive Qualitätsmanagement:

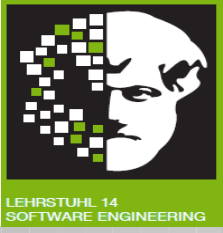


Übersicht Prüfverfahren



Testen

- Kontrollflussbezogene Testverfahren
  - Idee
  - Verschiedene Abdeckungen
- Datenflussbezogene Testverfahren
  - Idee
  - Kriterien



Viel Erfolg !