
Vorlesung „Methoden des Software Engineering“

Block C „Formale Methoden“

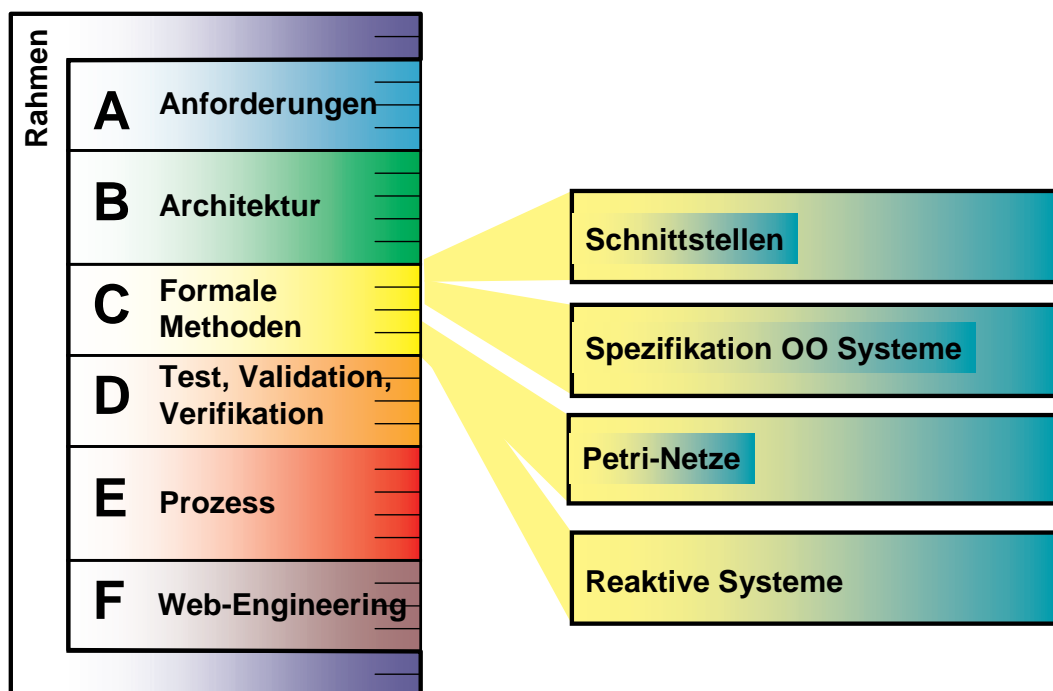
Schnittstellen und deren Interpretation

Martin Wirsing

Einheit C.1, 2.12.2004

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Gliederung Block C



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Ziele

- Die Rolle formaler Methoden in der Software-Entwicklung verstehen lernen
- (Die Syntax von) Schnittstellen durch Signaturen beschreiben lernen
- Die Interpretation von Schnittstellen durch
Algebren und Transitionssysteme
kennen lernen

Formale Methoden

Verwendung von mathematischen Notationen zur
Beschreibung von
Anforderungs- und Entwurfsspezifikationen
zusammen mit
Validierungs- und Verifikationstechniken

Geschichtlicher Überblick

- ab ca. 1960: **formale Sprachen und Automaten**; Petri-Netze
- ab ca. 1970: **Semantik von Programmiersprachen** (Scott, Strachey):
- ab ca. 1969: **Beweise von Programmen, Zusicherungskalküle**
(Dijkstra, Hoare, Floyd)
- ab ca. 1970: **Formale Programmentwicklung** (CIP, Burstall/ Darlington, Hoare)
- ab ca. 1980: **Temporale Logik** (Kröger, Manna, Pnueli)
zur Beschreibung des dynamischen Verhaltens von Systemen.
- ab ca. 1990: **Integration diagrammatischer Notationen
mit formalen Techniken,**
- ab ca. 1995: große Erfolge der Temporallogik durch Einsatz von **Modelchecking**

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Signaturen zur Schnittstellenbeschreibung

Zur Angabe der Schnittstelle eines Systems benötigt man die Angabe der

- Namen der nach außen sichtbaren Datentypen („Sorten“, „Typen“)
- Namen und Typ der nach außen sichtbaren Funktionen („Funktionssymbole“)

Definition

Eine (algebraische) *Signatur* Σ ist ein Paar (S, F) mit

S eine Menge von Sorten, d.h. Namen für Mengen.

F eine $S^* \times S$ –sortierte Familie von Funktionssymbolen,

wobei $\langle s_1, \dots, s_n \rangle \in S^*$ den Definitionsbereich,

$s \in S$ den Wertebereich der Funktionen aus

$F_{\langle \langle s_1, \dots, s_n \rangle, s \rangle}$ bezeichnet.

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Beispiel: Boolesche Werte

sig BOOL0 =

sort Bool;

ops true : Bool;

false : Bool;

not_ : Bool \rightarrow Bool;

and, _or_ , _implies_: Bool \times Bool \rightarrow Bool;

end

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Beispiel: Keller

sig STACK0 =

sorts Stack , Nat

ops empty : Stack

push : Nat \times Stack \rightarrow Stack

top : Stack \rightarrow Nat

pop : Stack \rightarrow Stack

Bemerkung: In der funktionalen Sichtweise werden Zustände explizit durch Terme dargestellt.

Beispiel:

Der Zustand z eines Kellers wird explizit dargestellt mit Hilfe von push:

Der Term $\text{push}(2, \text{push}(1, \text{empty}))$

repräsentiert einen zwei-elementigen Keller.

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Zustandsbasierte Signaturen

Bei einem zustandsbasierten (prozeduralen) System ist der Zustand implizit, d.h. der Typ für den Zustand wird durch den trivialen Typ Void ersetzt und erscheint nicht mehr in der Funktionalität der Operationen.

Beispiel Keller:

Der Typ Stack wird durch Void ersetzt.

Die Operationen erhalten folgende Typen:

```
ops    empty : Void
       push  : Nat → Void
       top   : Void → Nat
       pop   : Void → Void
```

Objektorientierte Signaturen

Bei objektorientierten Signaturen kommt der Objektbezeichner als erstes Argument hinzu. (Void bezeichne den trivialen Typ.)

Beispiel Keller:

sig STACKOO =

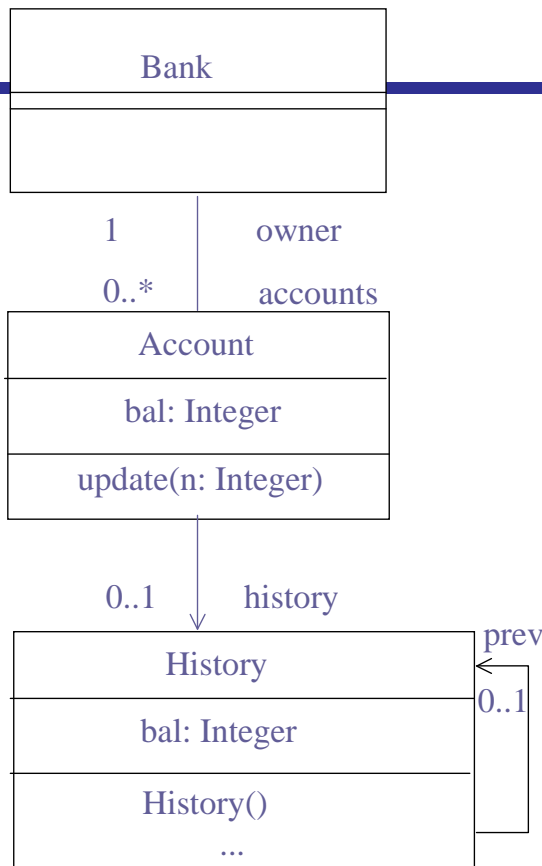
```
sorts  Stack , Nat
ops    empty : Stack → Void
       push  : Stack × Nat → Void
       top   : Stack → Nat
       pop   : Stack → Void
```

Stack
empty() push(n: Nat) top(): Nat pop()

Klassendiagramm

- Ein (einfaches) **Klassendiagramm** in UML besteht aus Klassen und Assoziationen.
Für weitere Beziehungen (Aggregation, Vererbung) siehe FOOSE.
- Die **Signatur** eines Klassendiagramms besteht aus Basistypen, allen Klassen des Diagramms und Typen für endliche Kollektionen (Mengen, Sequenzen, Multimengen) sowie aus Operationen für die Attribute, Assoziationen, Methoden und Konstruktoren.

Beispiel: Bank



Signatur (Ausschnitt)

sorts Bank, Account, History, Integer, Set(Account)
ops

- `..owner: Account → Bank`
- `..accounts: Bank → Set(Account)`
- `..bal: Account → Integer`
- `update: Account × Integer → Void`
- `..history: Account → History`
- `..bal: History → Integer`
- `..prev: History → History`
- `History: Void → History`
- ...

Klassensignatur eines Klassendiagramms Δ

Sorten sind alle Basistypen (Void, Integer, Boolean, ...),
 alle in Δ auftretenden Klassen, und
 $\text{Set}(C)$, $\text{Sequ}(C)$, $\text{Bag}(C)$ für alle Klassen C aus Δ

Operationen

- $_a: C \rightarrow D$ für jedes Attribut von C und
jede von C nach D gerichtete Assoziation mit Multiplizität 1 oder 0..1
- $_a: C \rightarrow \text{Set}(D)$ für jede von C nach D gerichtete Assoziation
mit Multiplizität *
- $m: C \times T_1 \times \dots \times T_n \rightarrow \text{Void}$
für jede Methode $m(x_1:T_1, \dots, x_n:T_n)$ der Klasse C
- $m: C \times T_1 \times \dots \times T_n \rightarrow T$
für jede Methode $m(x_1:T_1, \dots, x_n:T_n): T$ der Klasse C
- $C: T_1 \times \dots \times T_n \rightarrow C$
für jeden Konstruktor $C(x_1:T_1, \dots, x_n:T_n)$ der Klasse C

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Schnittstelleninterpretation durch Strukturen und Transitionssysteme

Eine Σ – Algebra besitzt zu

- jeder Sorte eine Trägermenge,
- jedem Funktionssymbol eine Funktion.

Definition

Eine Σ –Algebra A besteht aus

- einer Familie $(A_s)_{s \in S}$ von nichtleeren Trägermengen, und
- (totalen) Funktionen $f^A: A_{s_1} \times \dots \times A_{s_n} \rightarrow A_s \cup \{\perp\}$, für alle $f \in F_{\langle\langle s_1, \dots, s_n \rangle, s \rangle}$

Bemerkung

Enthält die Signatur Σ auch Prädikatensymbole, so spricht man von Σ -Strukturen.

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Beispiel: Keller funktional

Interpretation von Kellern als Sequenzen natürlicher Zahlen:

Trägermengen: $S_{\text{Stack}} = \mathbb{N}^*$ $S_{\text{Nat}} = \mathbb{N}$

Funktionen: $\text{empty}^S = \langle \rangle$

$$\text{push}^S(\langle k_1, \dots, k_i \rangle, n) = \langle k_1, \dots, k_i, n \rangle,$$

$$\text{top}^S(\langle k_1, \dots, k_i \rangle) = \begin{cases} k_i, & \text{falls } i > 0 \\ \perp, & \text{falls } i = 0 \end{cases}$$

$$\text{pop}^S(\langle k_1, \dots, k_i \rangle) = \begin{cases} \langle k_1, \dots, k_{i-1} \rangle, & \text{falls } i > 0 \\ \perp, & \text{falls } i = 0 \end{cases}$$

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Zustand

Zur Interpretation prozeduraler und objektorientierter Signaturen benötigt man den Begriff des Zustands.

Definition

- 1) Ein **Zustand eines prozeduralen (zustandsbasierten) Systems** ist gegeben durch die Interpretation einer (festen) Menge von Variablen.
- 2) Ein **Zustand eines objektorientierten Systems** ist gegeben durch ein Objektdiagramm.

Zustand (Prozedural)

Gegeben sei eine Menge von Sorten S und zugehörige Trägermengen A_s für alle $s \in S$.

1) Eine **Zustandssignatur** ist eine

Familie von (System-) Variablen $(X_s)_{s \in S}$

mit X_s abzählbar für alle $s \in S$.

2) Ein **Zustand** ist eine Belegung

$\beta: X_s \rightarrow A_s$ für alle $s \in S$.

Beispiel Keller:

$S = \{ \text{Stack}, \text{Nat} \}$ $A_{\text{Stack}} = \mathbb{N}^*$, $A_{\text{Nat}} = \mathbb{N}$

Systemvariable: $X_{\text{stack}} = \{\text{stack}\}$, $X_{\text{Nat}} = \{\text{result}\}$

Ein Zustand ist eine Belegung $\beta: X_{\text{Stack}} \rightarrow A_{\text{stack}}$, $\beta: X_{\text{Nat}} \rightarrow A_{\text{Nat}}$

Z.B. $\beta_1(\text{stack}) = \langle 17, 12, 3 \rangle$ oder nur $\beta_2(\text{stack}) = \langle 7, 10, 3, 11 \rangle$

$\beta_1(\text{result}) = 3$ wenn result unwichtig

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

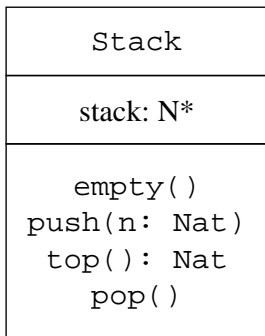
Zustand (Objektorientiert)

Ein (objektorientierter) **Systemzustand** besteht aus

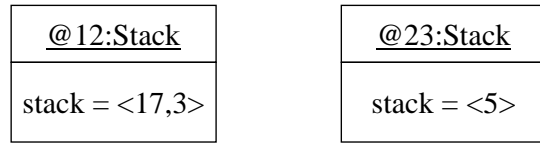
1. einer Menge von aktuell existierenden Objekten (für jede Klasse des Systems) und
2. einer Belegung aller Instanzvariablen, d.h. einer Funktion, die jeder Instanzvariable o.a (für jedes Objekt o und Attribut a) einen Wert zuweist.
3. (eventuell zusätzlich) einer Belegung der lokalen/freien Variablen des aktuell betrachteten Ausdrucks oder Programms.

Beispiel Keller mit zwei Stack-Objekten

Klassendiagramm:



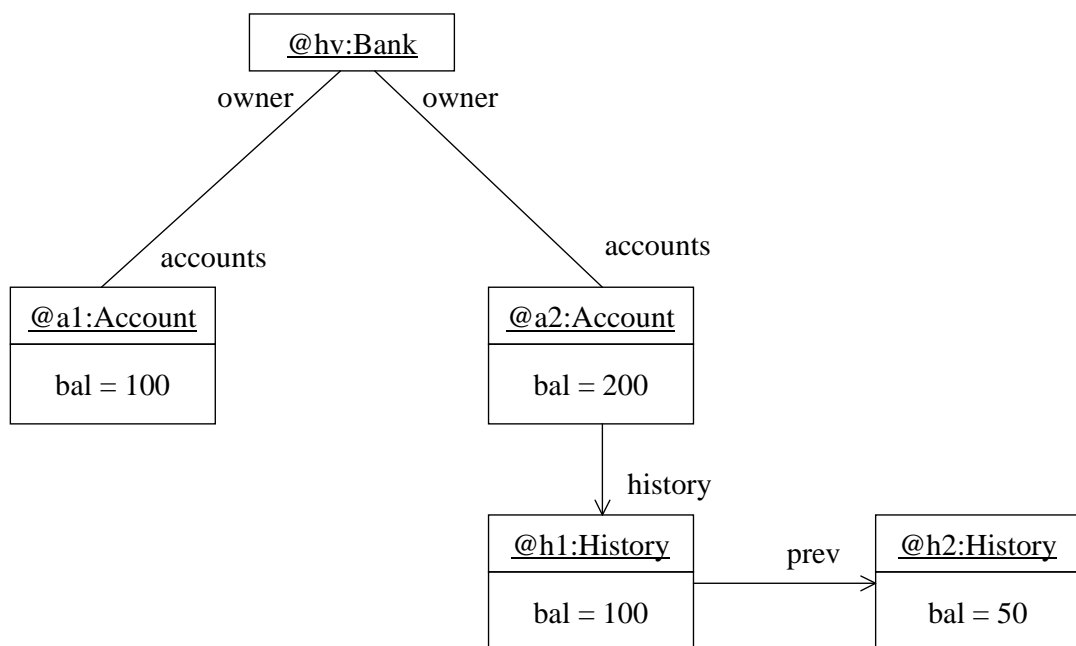
Objektdiagramm



Formale Beschreibung des Objektdiagramms:

Existierende Objekte @12, @23
 $\sigma(@12.stack) = \langle 17, 3 \rangle$ $\sigma(@23.stack) = \langle 5 \rangle$

Beispiel: Zustand des Banksystems



Transitionssystem für Keller (prozedural)

Ein markiertes **Transitionssystem** gibt die Beziehung zwischen dem Zustand vor und dem Zustand nach Ausführung einer Operation an; jede **Transition** wird mit dem passenden Methodenaufruf markiert.

Markierungen: $\text{pop}()$, $\text{top}()$, $\text{push}(n)$ für $n \in \mathbb{N}$

Transitionen:

$\text{stack} = s \xrightarrow{\text{push}(n)} \text{stack} = s^\circ \langle n \rangle$ für alle $n \in \mathbb{N}$
 $\text{stack} = s^\circ \langle n \rangle \xrightarrow{\text{pop}()} \text{stack} = s$
 $\text{stack} = s^\circ \langle n \rangle \xrightarrow{\text{top}()} \text{stack} = s^\circ \langle n \rangle, \text{result} = n$

Rückgabewert

wobei wir kurz $\text{stack} = s$ für den Zustand $\beta + [\beta(\text{stack}) = s]$ schreiben

und

$$\beta + [x = v](y) = \begin{cases} v & \text{falls } x=y \\ \beta(y) & \text{sonst} \end{cases}$$

Transitionssystem

Gegeben sei eine Signatur $\Sigma = (S, F)$.

1) Ein Transitionssystem

$\Gamma = (Z, T)$

ist gegeben durch

- * eine Menge Z von Zuständen und
- * eine Transitionsrelation $T \subseteq Z \times Z$.

2) Ein markiertes Transitionssystem

$\Gamma = (Z, A, T)$

ist gegeben durch

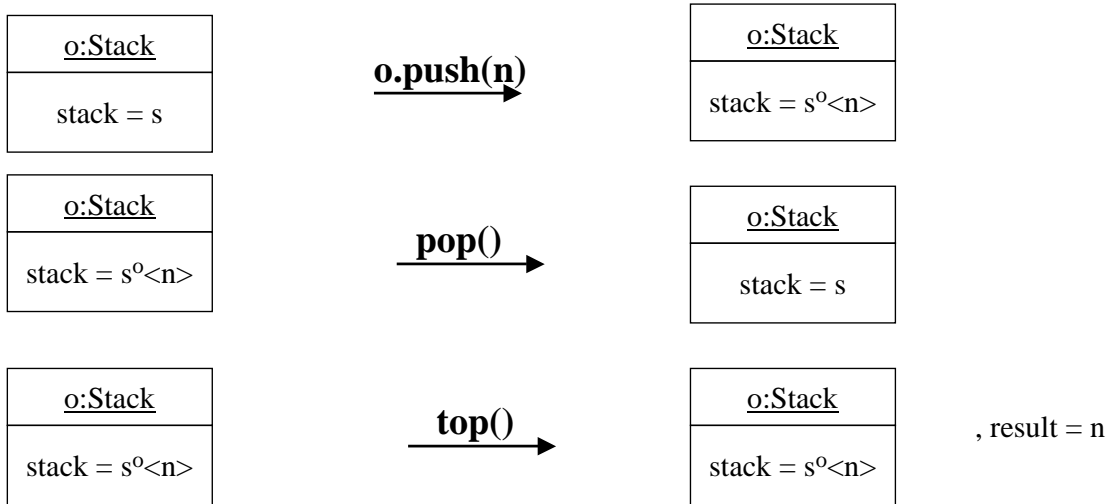
- * eine Menge Z von Zuständen
- * eine Menge A von Aktionen und
- * eine Transitionsrelation $T \subseteq Z \times A \times Z$.

Transitionssystem für Keller (objektorientiert)

Markierungen: $o.pop()$, $o.top()$, $o.push(n)$

für Stackobjekte o , $n \in \mathbb{N}$

Transitionen:



Transitionssystem für Keller (objektorientiert)

Bemerkung:

- Die Regeln der vorhergehenden Seite sind Graphersetzungsgesetze.
- Die Transitionsrelationen gelten für alle Objektdiagramme, die die linke Seite einer Regel enthalten.
- Die rechte Seite des Gesamtdiagramms entsteht durch Ersetzung der linken Seite der Regel durch die rechte Seite der Regel.

Transitionssystem für Keller (objektorientiert)

Genauer formuliert man die Transitionsrelation mathematisch:

Für alle Zustände σ gelte:

$$\sigma+[o.\text{stack} = s] \xrightarrow{o.\text{push}(n)} \sigma+[o.\text{stack} = s^\circ\langle n \rangle]$$

$$\sigma+[o.\text{stack} = s^\circ\langle n \rangle] \xrightarrow{o.\text{pop}()} \sigma+[o.\text{stack} = s]$$

$$\sigma+[o.\text{stack} = s^\circ\langle n \rangle] \xrightarrow{o.\text{top}()} \sigma+[o.\text{stack} = s^\circ\langle n \rangle], \text{ result} = n$$

Dabei bedeutet $\sigma+[o.a = v]$, dass in σ an der Stelle $o.a$ geändert wurde und der Wert von $o.a$ gleich v ist:

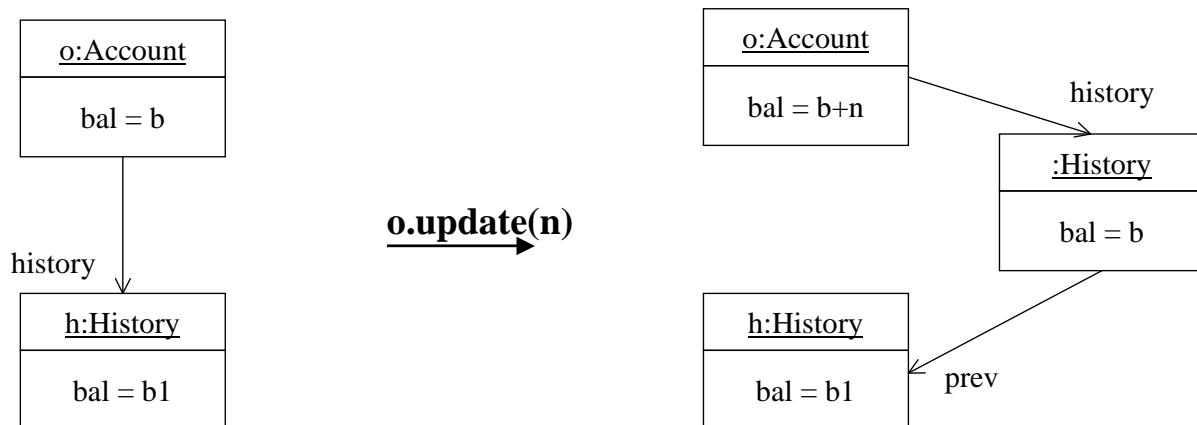
$$\sigma+[o.a = v](p.b) = \begin{cases} v & \text{falls } o=p \text{ und } a=b \\ \sigma(p.b) & \text{sonst} \end{cases}$$

Transitionssystem für Bank (objektorientiert)

Markierungen: $o.\text{update}(n)$, für Accountobjekte o , $n \in \mathbb{N}$

Transitionen:

Sei $b+n \geq 0$. Dann gelte:



Zusammenfassung

- Eine Signatur dient zur syntaktischen Beschreibung von Schnittstellen.
- Die funktionale Interpretation von Schnittstellen wird durch Algebren und Strukturen beschrieben.
- Die Interpretation prozeduraler und objektorientierter Schnittstellen wird durch Transitionssysteme beschrieben.

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Literatur

- Wirsing: Grundlagen der Systementwicklung, Kap. 2.2 und 3.6
- Hennicker: Formale objekt-orientierte Software-Entwicklung, Kap 2.1, 2.6 und 3.4
- Ehrig, Mahr, Cornelius, Große-Rohde, Zeitz: Mathematisch-strukturelle Grundlagen der Informatik, Springer, 1999, Kap. 7, Signaturen und Algebren

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München