

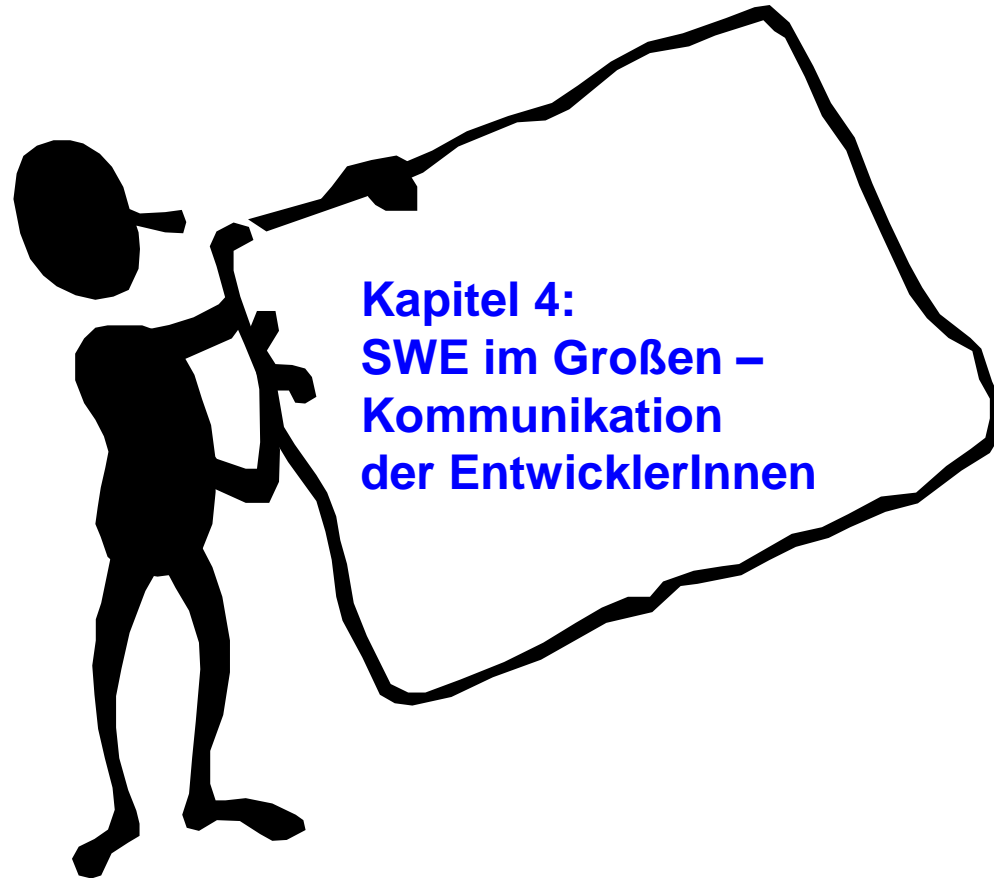
Einführung in Software Engineering

Barbara Paech, Marcus Seiler

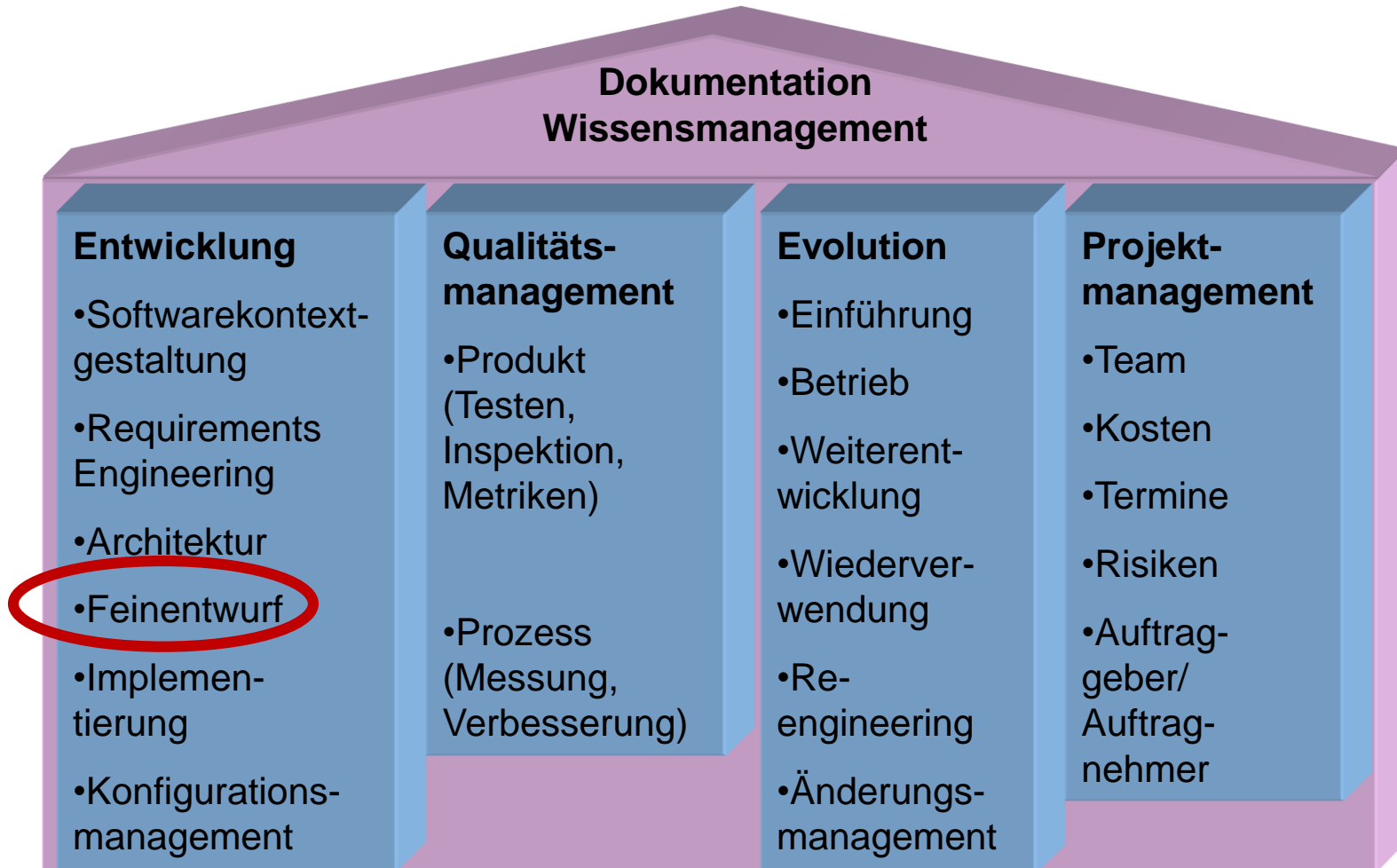
Institute of Computer Science
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
<http://se.ifi.uni-heidelberg.de>
paech@informatik.uni-heidelberg.de



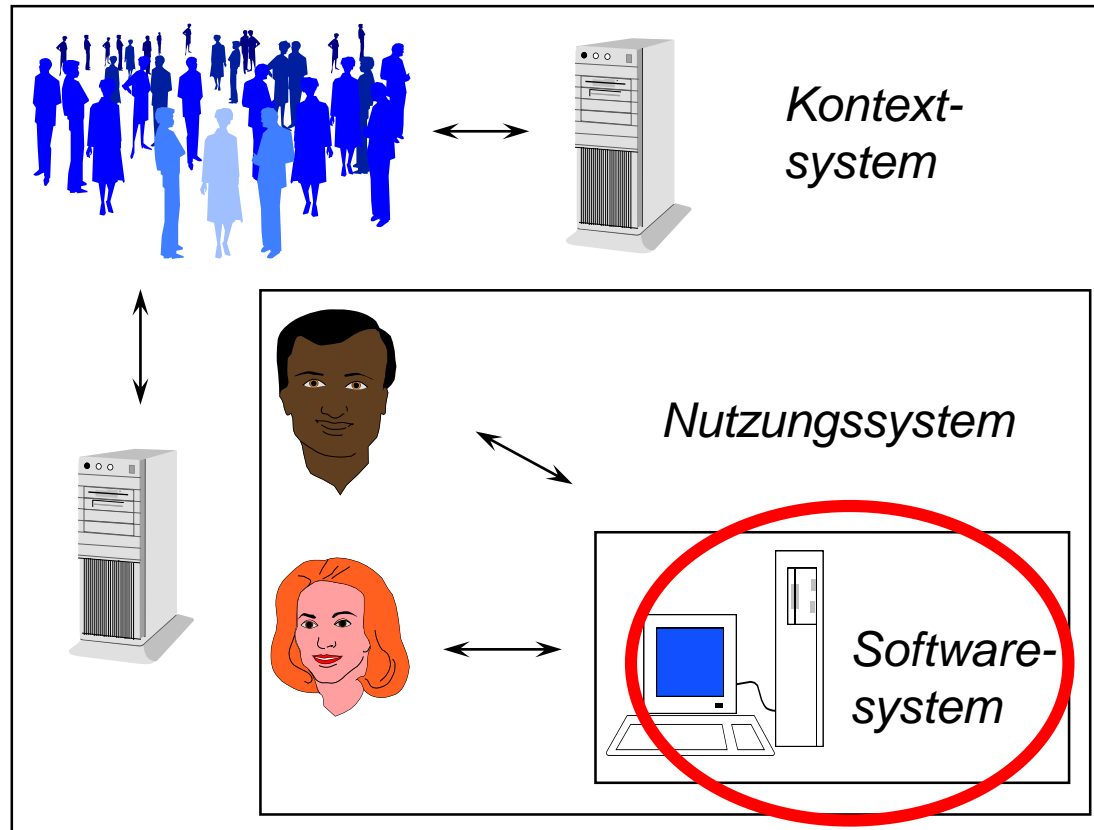
RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG



Aufgabenbereiche des Software Engineering



Wdh. Gestaltungsbereiche der SW-Entwicklung



[Paech2000]

4. Kommunikation der EntwicklerInnen (1. Teil)

- 4.1. Einführung Modellierung
- 4.2. Klassendiagramme
- 4.3. Interaktionsdiagramme
- 4.4. Zustandsdiagramme
- 4.5. Klassenentwurf mit OOAD
- 4.6. Kommunikation von Erfahrungswissen (Entwurfsmuster)
- 4.7. Kommunikation von Entscheidungen (Rationale)
- 4.8. Zusammenfassung Kommunikation der EntwicklerInnen

4.1.Einführung Modellierung

Wdh (Kapitel 2.7.) Teamorganisation

- Software-Entwicklung ist ein **arbeitsteiliger Prozess**.
- Die am Projekt beteiligten Personen – das **Projektteam** – sind zu **organisieren**:
 - Festlegung von **Weisungsbefugnis**
 - Festlegung von **Kommunikationswegen**
 - Je nach Größe des Projekts kann das Projektteam in mehrere Teams zerfallen. Ein Team sollte einen **definierten Bereich** bearbeiten und aus **höchstens fünf bis sieben Personen** bestehen.
- Im Folgenden werden einige typische Organisationsformen für Teams vorgestellt. Diese Übersicht ist natürlich **holzschnittartig** vereinfacht, in der Praxis gibt es alle möglichen **Mischformen**.



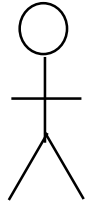
- Auf Basis der Anforderungen entwickelt das Team ein Verständnis der Struktur des Softwaresystems (den Entwurf)
- Codestrukturen sind zu kleinteilig (siehe UI-Struktur vs. Bildschirmbeschreibung)
- Es ist wichtig, eine geeignete Abstraktionsebene zur Beschreibung des Systems zu nutzen.
- => Modellierungssprachen

Was ist ein Modell (1)?

- Ein Modell ist seinem Wesen nach **eine in Maßstab, Detailliertheit und/oder Funktionalität verkürzte beziehungsweise abstrahierte Darstellung** des originalen Systems [Stachowiak 73]
- Ein Modell ist eine **Abstraktion eines Systems mit der Zielsetzung das Nachdenken** über ein System zu vereinfachen, indem irrelevante Details ausgelassen werden [Brügge 00].
- Modelle sind also charakterisiert durch
 - **Abbildungsmerkmal** (es gibt ein Original)
 - **Verkürzungsmerkmal** (Original nicht vollständig dargestellt)
 - **Pragmatisches Merkmal** (Modell steht für Original in einem bestimmten Zusammenhang, Zweck)

Beispiel Modellierung

Modell 1

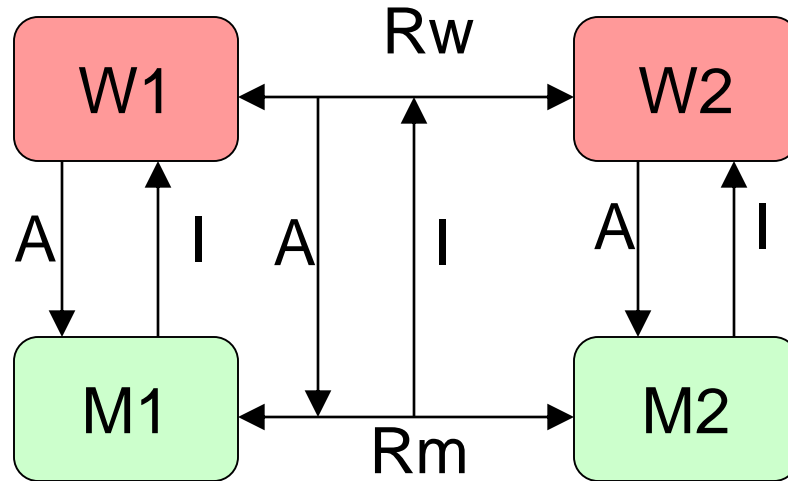


Modell 2



Modell 3





- W1, W2: Elemente der realen Welt
- R_w : Relation in der realen Welt
- M1, M2: Elemente im Modell
- R_m : Relation im Modell
- A: **Abstraktion** (Abbildung reale Welt \rightarrow Modell)
- I: **Interpretation** (Abbildung Modell \rightarrow reale Welt)

- **Beispiele** für Modelle:
 - Spielzeug („Verkürzung“ der Größe und des Innenlebens, Zweck: für Kinder handhabbar)
 - Landkarten („Verkürzung“ der Landschaft, Zweck: Orientierung)

- Zur Darstellung von Modellen ist eine **Notation** notwendig.
 - In SWE typisch: Text, Diagramm, Tabelle, Formeln, Programmtext, Strukturierter Text (z.B. eingeschränkte Satzform), Pseudocode

■ Syntax

- Konkretes Aussehen
- Abstrakte Syntax
- Kontextbedingungen zur Wohlgeformtheit

Analog zu Code

■ Semantik

- Bedeutungsbeschreibung
- Interpretation in der realen Welt (bzw. der beschriebenen Domäne)

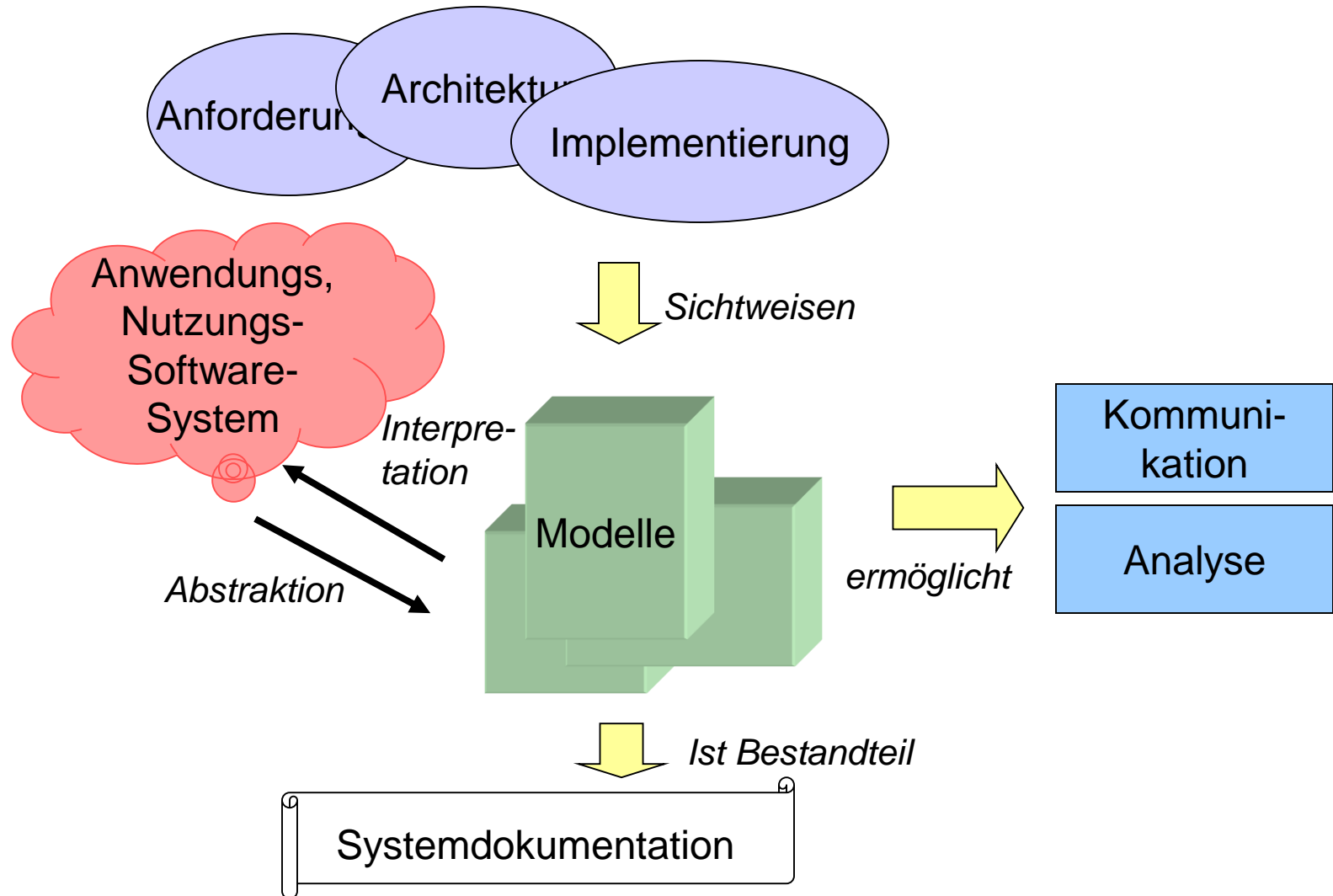
■ Pragmatik (Methodik des Gebrauchs)

- Analysetechniken (Typprüfung, Konsistenzchecks)
- Simulationstechniken
- Transformationstechniken (Z.B. Refactoring)
- Generierungsmöglichkeiten

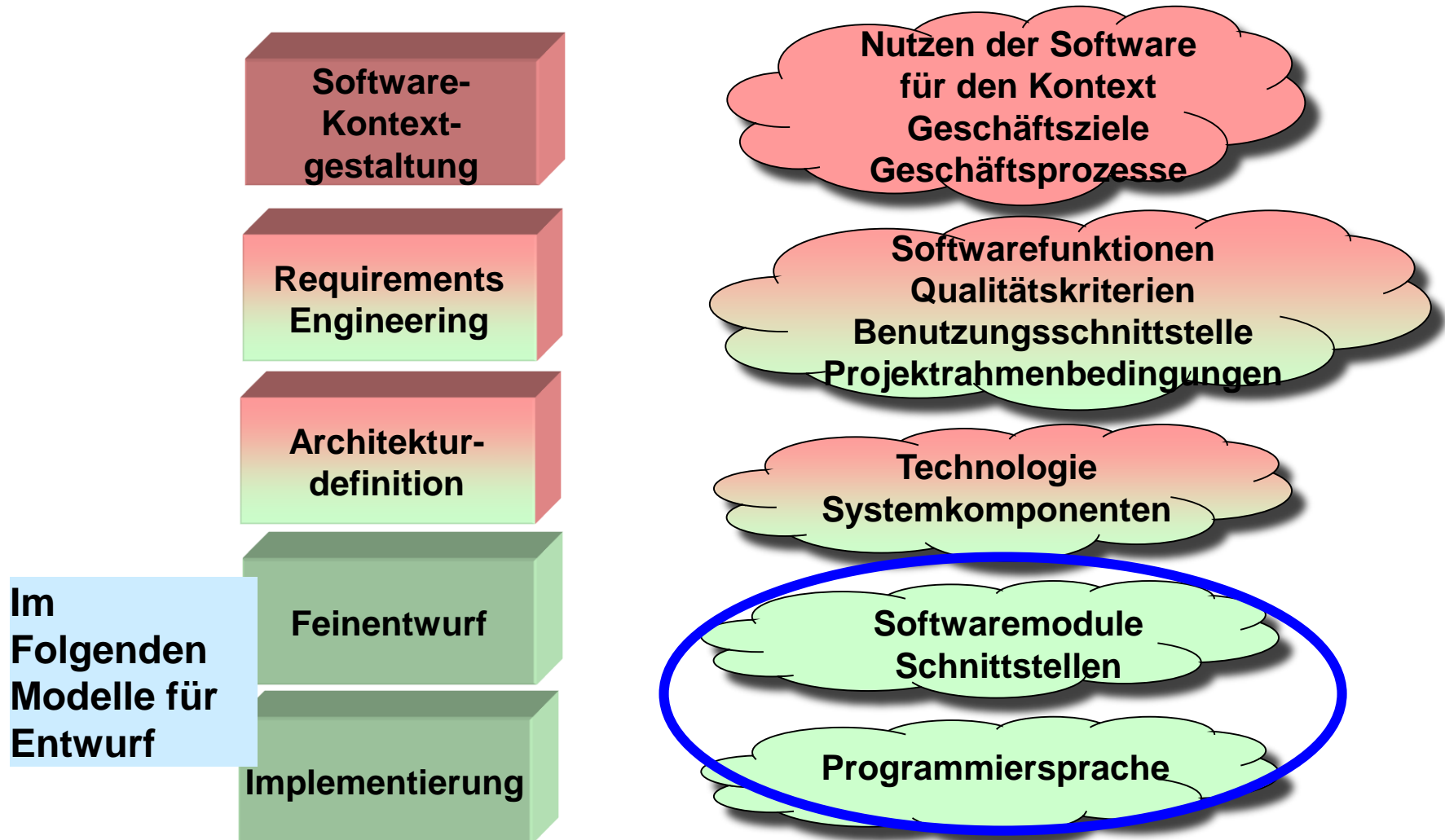
- Bilden eine „**Theorie**“ zur Bearbeitung der in einer Notation getroffenen Aussagen.

- Modelle helfen **komplexe Zusammenhänge zu verstehen**
 - **Bildhafte Darstellung und Abstraktion** unterstützen **Kommunikation** zwischen verschiedenen Beteiligten
 - **Skizze** zur Diskussion
 - **Vorgabe (Blaupause)** zur Entwicklung (**VORBILD, präskriptiv**)
 - **Exakte Notation und Abstraktion** ermöglicht Verwendung von Werkzeugen zur **Analyse** von Eigenschaften (des Originals) (**ABBILD, deskriptiv**)
 - Größe
 - Komplexität
 - Erreichbarkeit von Graphen
 -

Modelle in der Systementwicklung



Wdh. Aktivitäten und Gestaltungsentscheidungen



- Seit 1997 De-Facto-Industrie-Standard, der die bisherigen Einzelnotationen integriert (aktuell Version 2.5, www.uml.org)
- Vor allem ausgerichtet auf **Objektorientierte Entwicklungsmethoden**
- definiert
 - **Strukturdiagramme** (Statik des Systems)
 - **Verhaltensdiagramme** (Dynamik des Systems)
 - Interaktionen
 - Abläufe
 - Zustandsübergänge

■ Entwurf

- Klassendiagramm (Analyse und Entwurf) +
- Objektdiagramm (spez. Situationen) 0
- Paketdiagramm (Bündel von Klassen) -

+ = ausführlich besprochen
0 = kurz besprochen
- = nicht besprochen

■ Architektur

- Kompositionsstrukturdiagramm(interne Struktur im Detail) 0
- (logische) Komponentendiagramm (interne und externe Sicht) 0
- Verteilungsdiagramm (auf physische Komponenten) 0

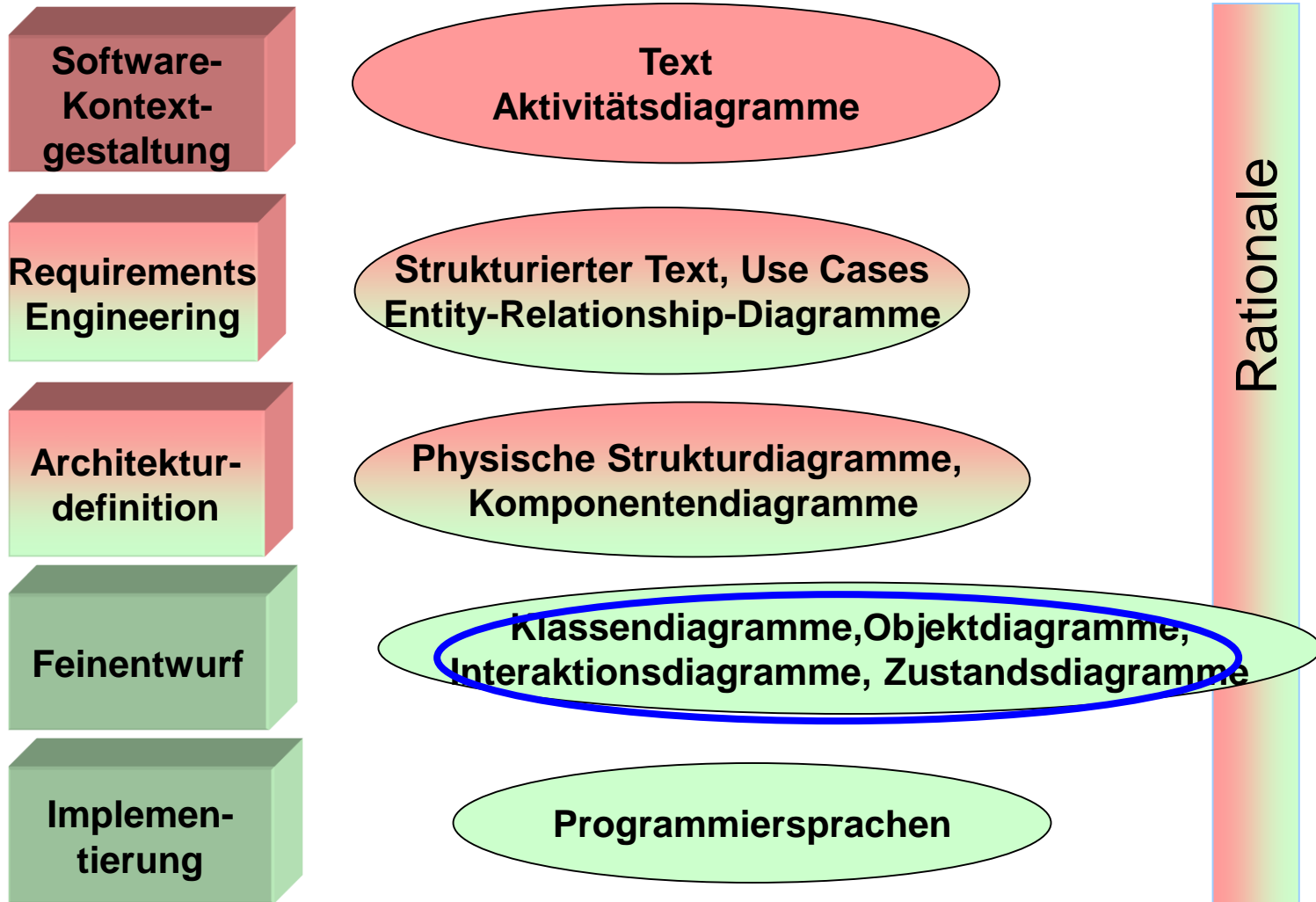
■ Abläufe

- Use Case Diagramm (Übersicht der UC) +
- Aktivitätsdiagramm (Folgen von Aktivitäten) 0
- Zustandsdiagramm (Folgen von Zuständen) +

+ = ausführlich besprochen
0 = kurz besprochen
- = nicht besprochen

■ Interaktion

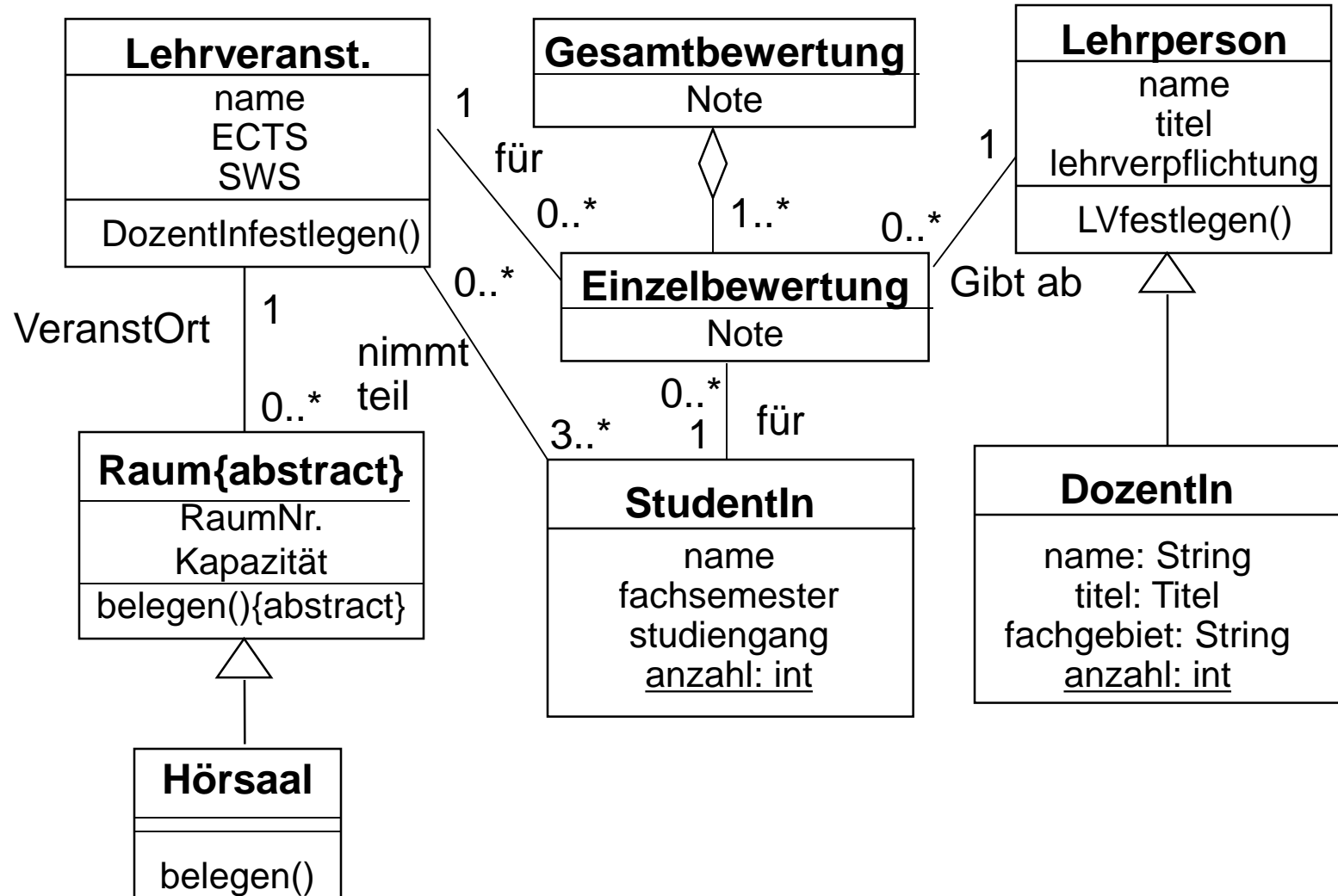
- Sequenzdiagramm (Nachrichtenfolgen) +
- Kommunikationsdiagramm (Fokus auf eine Komponente) 0
- Zeitdiagramm (Kommunikation zwischen Automaten) -
- Interaktionsübersichtsdiagramm (Interaktion mehrerer Interaktionen) -



- J Ludewig, H Lichter (2010) *Software Engineering*, dpunkt
- M Seidl , M Brandsteidl, C Huemer, G Kappel (2012).
UML@ Classroom: Eine Einführung in die objektorientierte Modellierung. dpunkt Verlag.
- H Störrle (2005) *UML 2 für Studenten*, Pearson Studium
- M Fowler (2004) *UML Distilled*. Addison-Wesley:
<http://www.ub.uni-heidelberg.de/cgi-bin/edok?dok=http%3A%2F%2Fproquest.safaribooksonline.com%2F0321193687>
- R Miles (2006) *Learning UML 2.0*. O'Reilly:
<http://www.ub.uni-heidelberg.de/cgi-bin/edok?dok=http%3A%2F%2Fproquest.safaribooksonline.com%2F0596009828>

4.2. Klassendiagramme



Beispiel: Klassendiagramm







- Klassen/Objektdiagramm
 - Klassen (Objekte)
 - Assoziationen (Aggregation, Komposition)
 - Attribute
 - Operationen
 - Generalisierungsbeziehungen (Vererbung)
 - Schnittstellen

- Siehe Folien TU Wien
- <http://www.uml.ac.at/de/lernen>

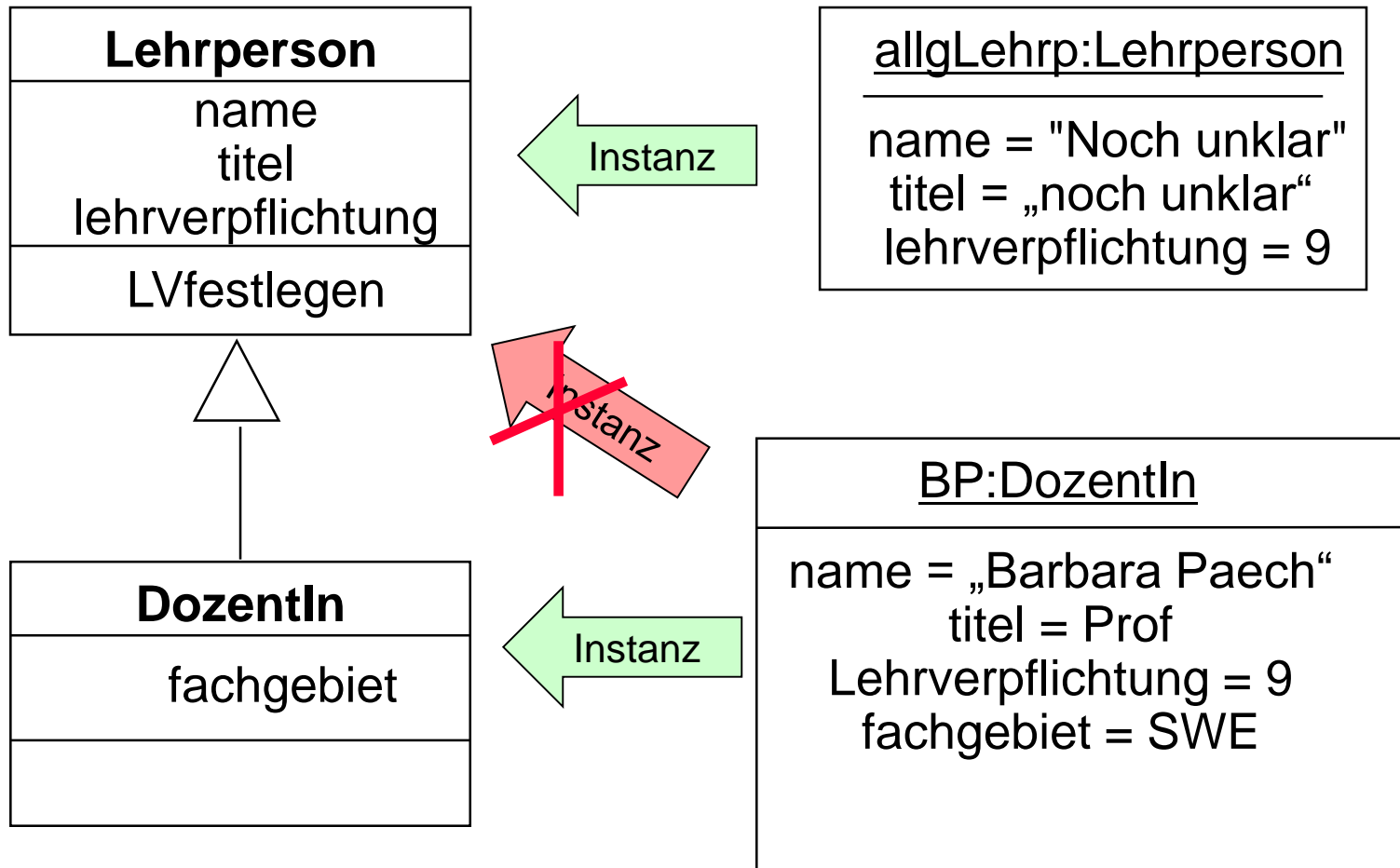
■ Falsch oder richtig??

- String **X**
- public adressen : String[1..*] 
- # bruder : Person 
- public / int **X**
- numbers : int[1..*] ordered **X**

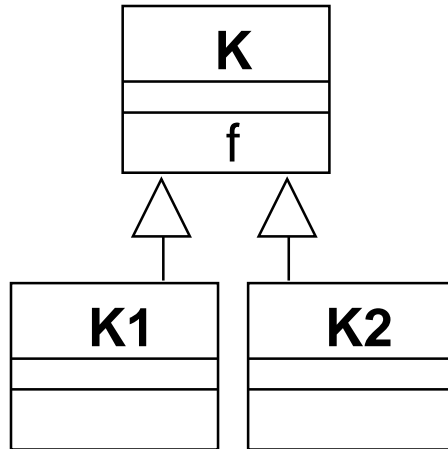
■ Falsch oder richtig??

- `+add (summand1,summand2) : int` 
- `add (int i, int j)` **X**
- `mult (summand : Matrix[2..*]{ordered}) : Matrix` 
- `setLstKI (lohnsteuerklasse : int = 1)` 
- `Clear (foo) : void` **X**
- `sub (in minuend : double, in subtrahend : double, return
resultat : double)` 

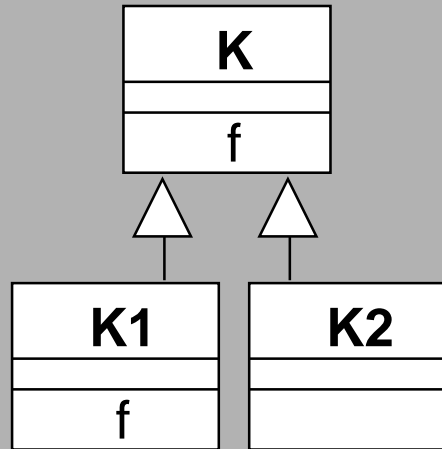
Vererbung und Instanzen



Redefinieren von Operationen

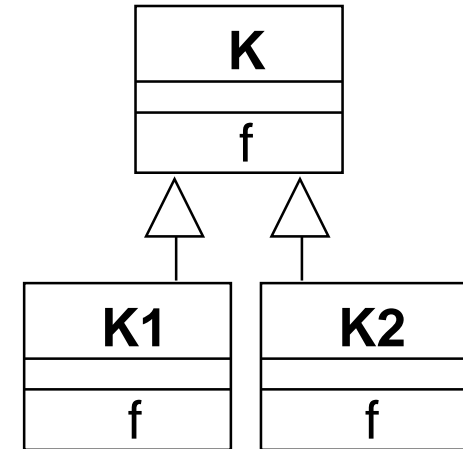


Das in K definierte Verhalten f gilt für alle Objekte der Klassen K, K1 und K2.



Das in K definierte Verhalten f gilt nur für die Objekte der Klassen K und K2.

K1 definiert ein anderes Verhalten (*Redefinition, override*).



Das in K definierte Verhalten f gilt nur für die Objekte der Klasse K
(kann in K1 und K2 bei der Neudefinition von f verwendet werden, sonst nutzlos).

Abstrakte Klasse und Schnittstellenklasse (Interface)

	Abstrakte Klasse	Schnittstellenklasse
Instanzen		
Attribute		
Operationen		
Verhalten der Operationen		

Schnittstellenklasse ist **Sicht** auf eine Klasse

Abstrakte Klasse und Schnittstellenklasse (Interface)

	Abstrakte Klasse	Schnittstellenklasse
Instanzen	nein	nein
Attribute	ja	Nur eingeschränkt (muss Wert haben, final)
Operationen	ja	ja
Verhalten der Operationen	In Abstrakter Klasse ggf. definiert, aber kann in Unterklasse redefiniert werden	In Schnittstellenklasse nicht definiert. Muss in Unterklassen definiert werden.

Schnittstellenklasse ist **Sicht** auf eine Klasse

- **Klassendiagramme abstrahieren vom Code.** Sie beschreiben den Code unter Auslassung von Details.
 - Klassendiagramme werden zusätzlich z.B. in der objektorientierten Analyse oft verwendet, um die Anwendungsdomäne zu modellieren.
 - Diese beiden Verwendungen sollten **klar unterschieden** werden
-
- Was wird durch ein Klassendiagramm **nicht** beschrieben?
 - Code: z.B. welche Operation welche andere Op. wann aufruft
 - Domänenbeschreibung: z.B. viele Eigenschaften der Entitäten, 3er-Beziehungen

Hausaufgabe Klassendiagramm

- Was sind die wichtigsten Elemente der Klassendiagramme?
 - Klassen (Attribute, Operationen), Objekte, Assoziationen (Aggregation, Komposition, Multiplizität), Vererbung, Schnittstelle
- Geben Sie 3 Beispiele für Codedetails an, die nicht im Klassendiagramm beschrieben werden können.
 - Operationsrumpf, Art der Assoziation (Aufruf oder Erzeugung etc), Multiplizitäten können mögliche Assoziationen nicht vollständig beschreiben
- Beschreiben Sie den Unterschied zwischen Klassen, abstrakten Klassen und Schnittstellenklassen.
 - Nur Klassen haben Instanzen
 - Bei abstrakter Klasse kann Operation nicht definiert sein, bei Schnittstellenklassen darf Operation nicht definiert sein
- Wie kann Mehrfachvererbung in Java umgesetzt werden?
 - Mittels (Einfach-)Vererbung und Schnittstellen

- Klassendiagramme beschreiben **komplexe Strukturen von Objekten**
 - **Kapselung** von Attributen und Operationen zu einer konzeptuellen Einheit
 - Ausprägung von **Instanzen** als Objekte
 - **Typisierung** von Objekten
 - **Extension** (Menge aller zu einem Zeitpunkt existierenden Objekte)
 - Charakterisierung der **möglichen Strukturen** eines Systems

- Es gibt noch viel mehr Detailnotation; nachzulesen bei Bedarf in der Literatur

- Erstellung von Klassendiagramm aus Code
- Ergänzung von domänen-nahem Klassendiagramm entsprechend zu Text
- Siehe Arbeitsblatt 7

4.3. Interaktionsdiagramme

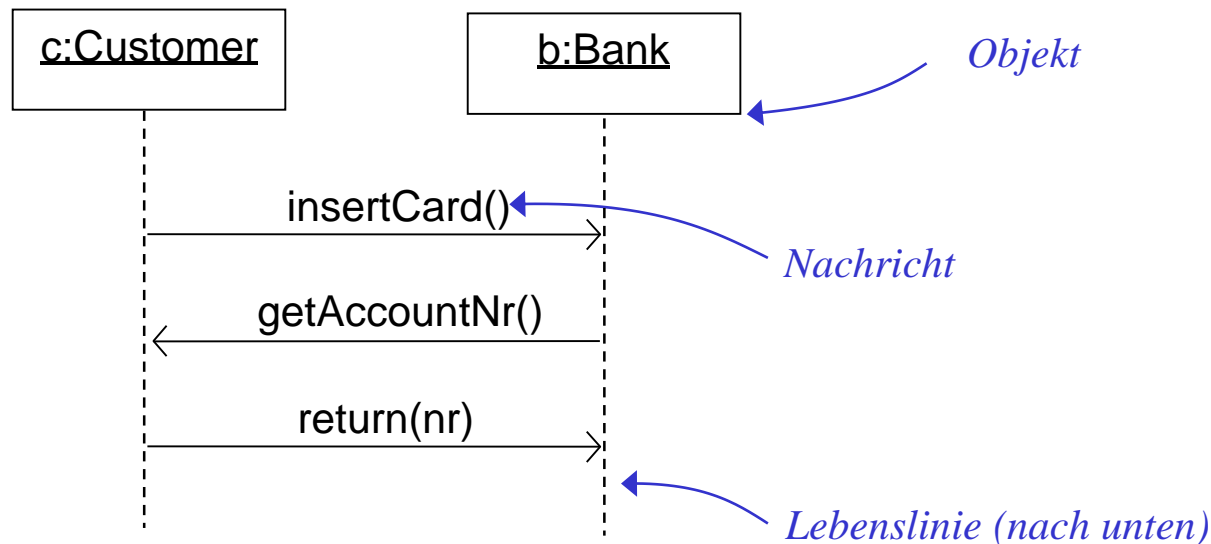
- Beschreibt **Kommunikation zwischen verschiedenen Partnern**

- Wesentliche Diagramme in UML
 - Sequenzdiagramm
 - Kommunikationsdiagramm (in der Vorlesung nur kurz erwähnt)
 - Zeitdiagramm (nicht in der Vorlesung)
 - Interaktionsübersichtsdiagramm (nicht in der Vorlesung)

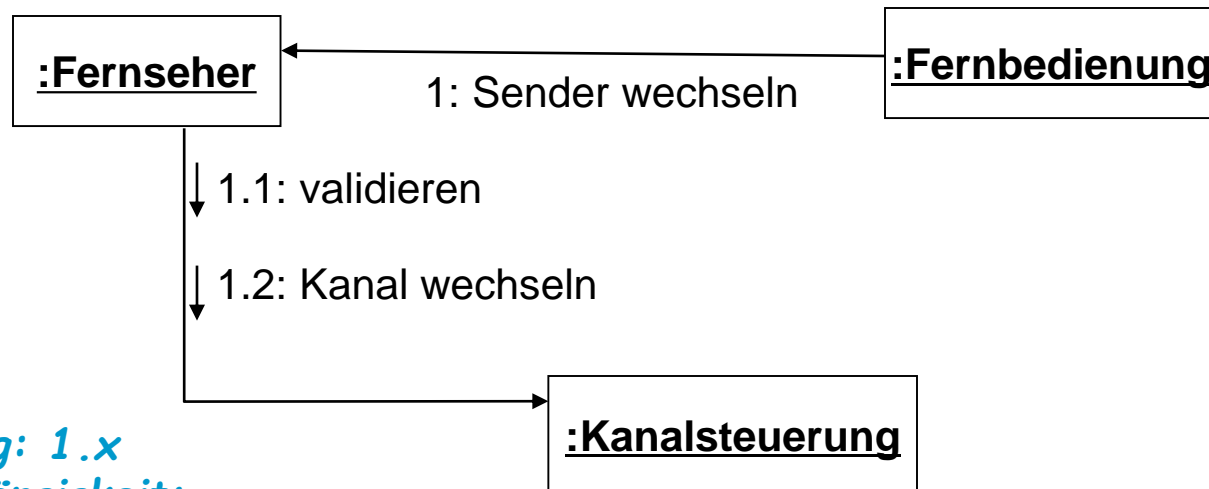
- Siehe Folien TU Wien
- <http://www.uml.ac.at/de/lernen>

Grundidee Sequenzdiagramm

- Objekte werden oben angeordnet
- Die Lebenslinie schreitet für alle Objekte gleich voran
- Ein Sequenzdiagramm zeigt den **Nachrichten-** bzw. **Ereignisfluss** zwischen Objekten



- **Alternative Darstellung** durch Kommunikationsdiagramme
 - zweidimensionale Anordnung der Objekte
 - Nummerierung der Interaktionen statt Lebenslinien



*Nummerierung: 1.x
bedeutet Abhängigkeit:
1 ist erst abgeschlossen,
Wenn 1.x abgeschlossen*

Hausaufgabe Sequenzdiagramme

- Was sind die wichtigsten Elemente der Sequenzdiagramme?
 - Akteure (Objekte), Lebenslinien, Pfeile mit Nachrichten
- Welche Arten der Kommunikation werden unterschieden?
 - Synchron vs. asynchron
- Was ist der Unterschied zwischen einem Sequenzdiagramm und einem Kommunikationsdiagramm?
 - Im wesentlichen gleiche Ausdruckskraft, aber andere Notation
 - Bei Kommunikationsdiagramm Fokus auf einzelne Akteure, zeitlicher Ablauf nur durch Nummerierung
 - Bei Sequenzdiagramm zeitlicher Ablauf im Vordergrund, Zeitfluss von oben nach unten

- Wichtig zur **Detaillierung der Interaktion zwischen verschiedenen Aktoren**
- Vor allem Verwendung von Sequenzdiagrammen
- Kommunikationsdiagramm besser, wenn ein spezieller Akteur (Objekt) im Mittelpunkt steht

- Erstellung von Sequenzdiagramm aus Code
- Siehe Arbeitsblatt 7