

# Software Engineering

## SS 2005

**Prof. Dr. Barbara Paech, Jürgen Rückert**



Institut für Informatik  
Im Neuenheimer Feld 326  
69120 Heidelberg  
<http://www-swe.informatik.uni-heidelberg.de>  
[paech@informatik.uni-heidelberg.de](mailto:paech@informatik.uni-heidelberg.de)

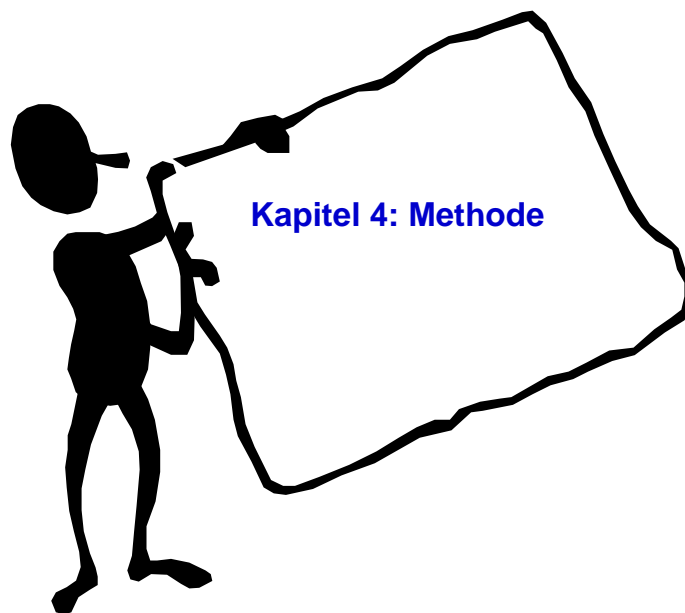


RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG



#### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test



4. Methode

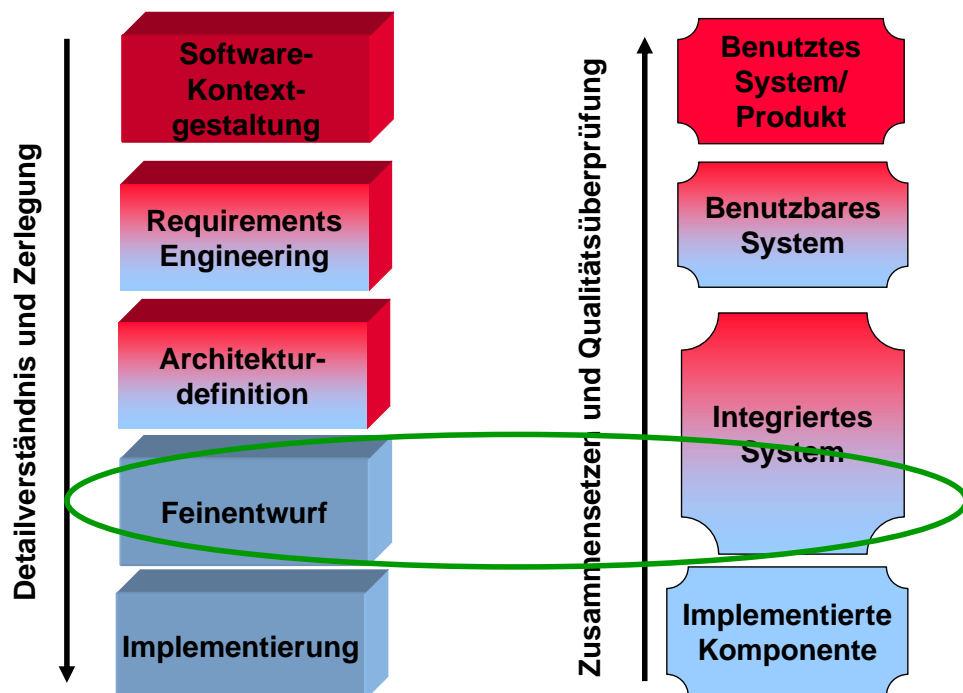
- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

- ◆ 4.1. Allgemeines
- ◆ 4.2. Vorgehen Requirements Engineering
- ◆ 4.3. Vorgehen Architektur
- ◆ **4.4. Vorgehen Entwurf**
- ◆ 4.5. Vorgehen Testen

## 1.3.2. Aktivitäten und Ergebnisse der Entwicklung

4. Methode

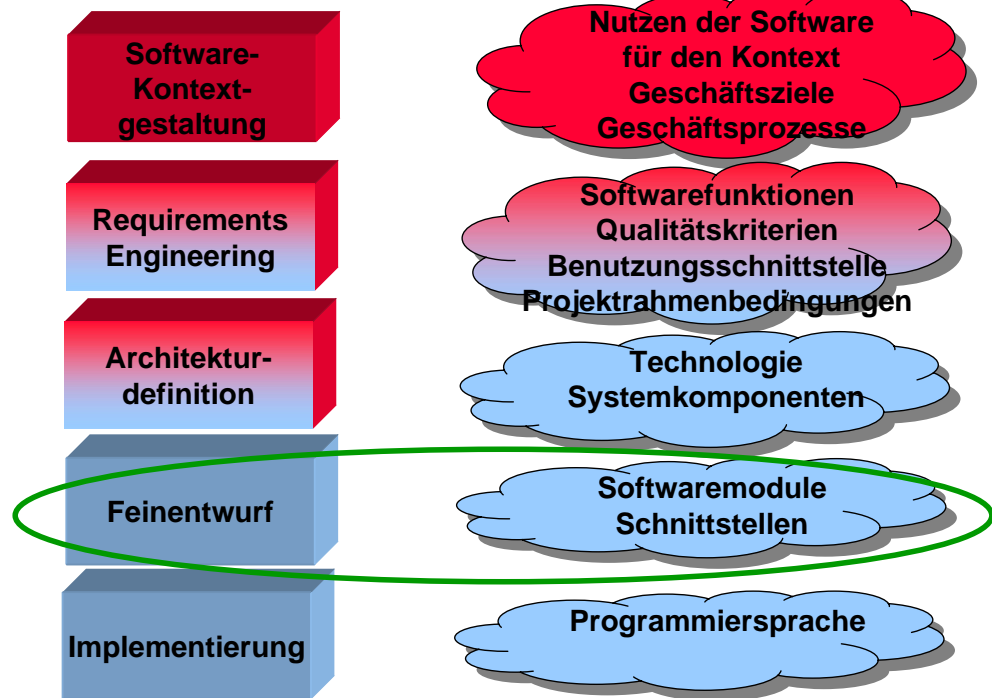
- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test



## 1.3.2. Gestaltungsentscheidungen

### 4. Methode

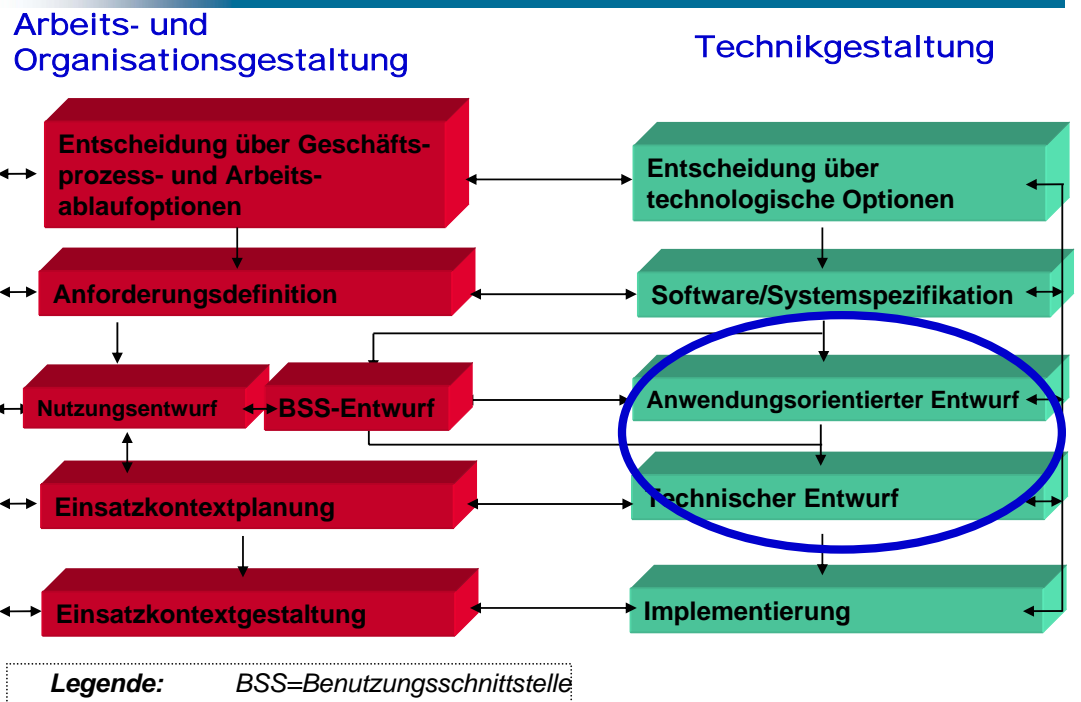
- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test



## 4.2.4. SW-Entwicklung als Arbeits- und Technikgestaltung

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

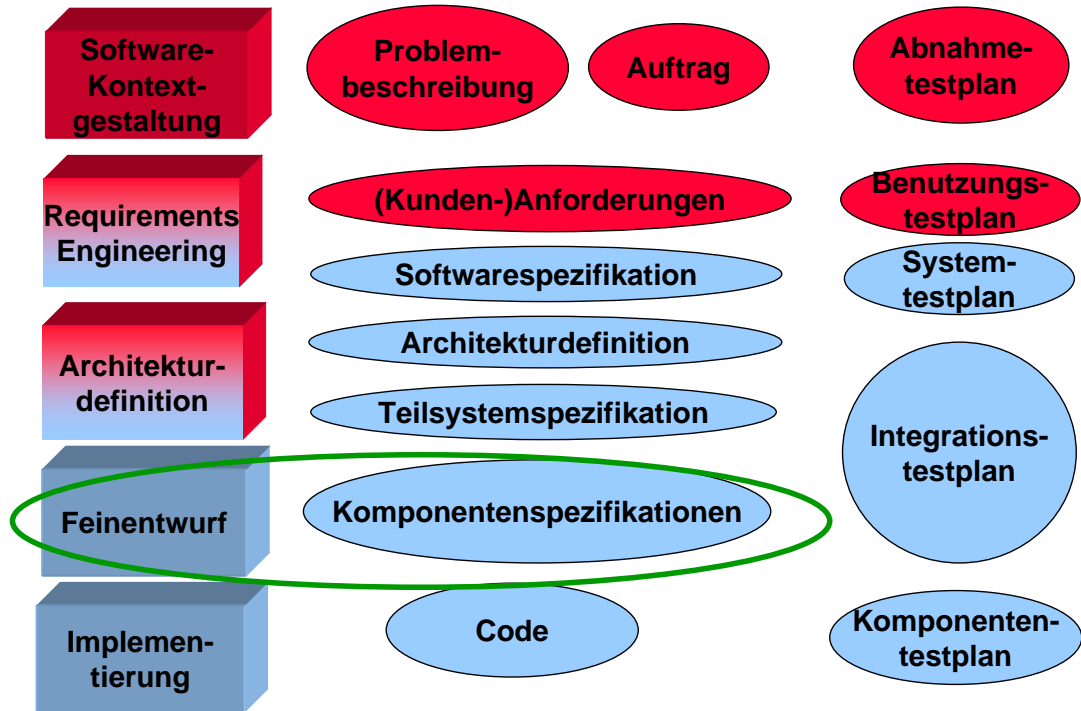


[Paech 2000]

## 1.3.2. Entwicklungsdokumente / Abstraktionsebenen

4. Methode

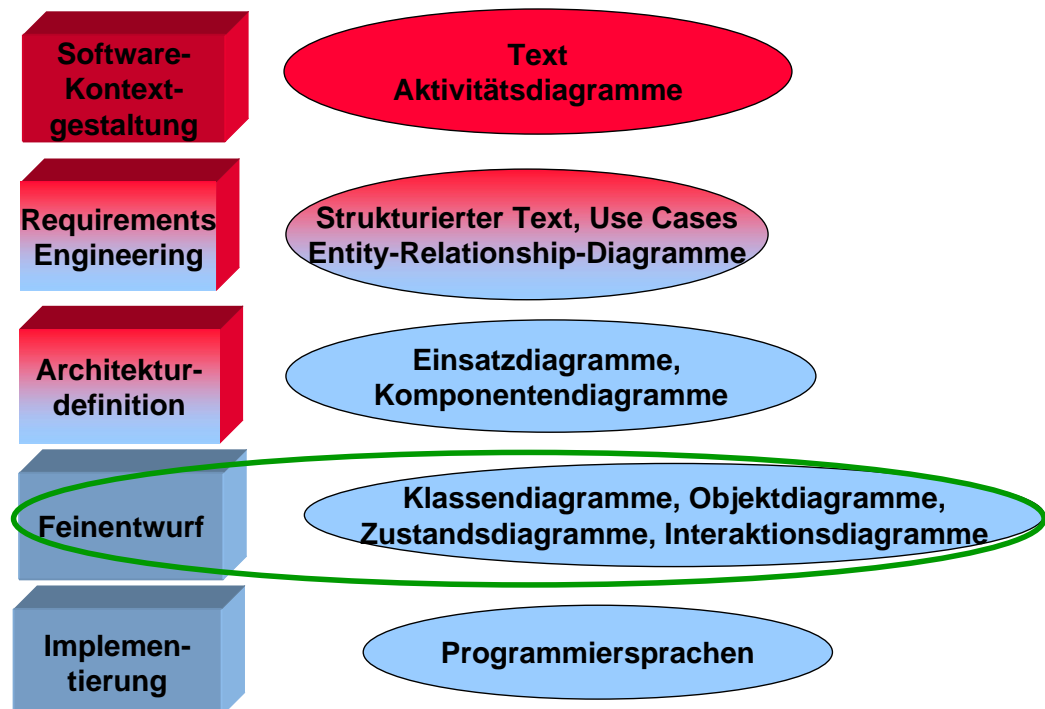
- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test



## 1.3.5. Beschreibungstechniken

4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test



### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- ▶ 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

- ◆ 4.4.1. Überblick
- ◆ 4.4.2. Von Use Cases zu Analyseklassen
- ◆ 4.4.3. Verfeinerung der Analyseklassen

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- ▶ 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

- ◆ Ziel des Feinentwurfs ist die Definition eines **Entwurfsklassenmodells**
- ◆ Das geschieht in 2 Schritten:
  - Definition eines **Analyseklassenmodells** aus dem E/R-Diagramm, Use Cases und Systemfunktionen
  - Verfeinerung des Analyseklassenmodells in Bezug auf Implementierung
- ◆ Qualitätsziele und Entwurfsprinzipien wie bei Architektur

## 4.4.2. Von Use Cases zum Analyseklassendiagramm

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

#### ◆ E/R-Diagramm in der Spezifikation:

Modellierung der wichtigen Konzepte

#### ◆ Analyseklassendiagramm:

Modellierung des Systemverhaltens auf abstraktem Niveau.

- Allgemeine *schematische* Beschreibung des Verhaltens
- Interne Struktur des Systems ("von innen")
- Struktur aber noch nicht für Implementierung optimiert

## 4.4.2. Umsetzung von Use Cases (nach Ivar Jacobson, OOSE)

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

◆ Jeder Use Case und jede Systemfunktion wird durch Kommunikation zwischen den System-Objekten realisiert werden.

◆ Hauptschwierigkeit ist

- Identifikation der richtigen Klassen
- Verteilung des Verhaltens als Operationen auf die Klassen

## 4.2.4. Vorgehen: Datenmodellierung

### 4. Methode

#### 4.1. Allgemeines

#### 4.2. Vorgehen RE

#### 4.3. Vorgehen Architektur

#### 4.4. Vorgehen Entwurf

#### 4.5. Vorgehen Test

- ◆ Sammle **Begriffe** (z.B. aus Aufgabenbeschreibung)
- ◆ Unterscheide **Klassen** und **Attribute**
  - Klassen haben eigene Identität, mehrere Attribute, z.B. Buch
  - Attribute spiegeln einen (evtl. veränderlichen) Wert wieder, z.B. Titel, Lesername
- ◆ Ordne Attribute den Klassen zu
- ◆ Definiere **Beziehungen**
  - spiegeln Abhängigkeiten wieder (oft Verben), z.B. hat\_ausgeliehen (Leser,Buch)
- ◆ Lege **Vielfachheiten** fest
- ◆ **Achtung:** "System" nicht als Klasse (zentralisiert sonst Verhalten)
- ◆ **Achtung:** Benutzerrollen nur als Klassen, wenn System darüber Daten speichert

Folie 13

## 4.4.2. Stereotypen von Analyseklassen

### 4. Methode

#### 4.1. Allgemeines

#### 4.2. Vorgehen RE

#### 4.3. Vorgehen Architektur

#### 4.4. Vorgehen Entwurf

#### 4.5. Vorgehen Test

- ◆ Verschiedene Stereotypen von Klassenrollen (bei Identifikation aus Use Cases):
  - **Dialog-Klasse (boundary class):** Erwähnung von Benutzerinteraktion und deren Inhalt
  - **Steuerungs-Klasse (control class):** Beschreibung von Vorgängen und Reihenfolgen
  - **Entitäts-Klasse (entity class):** Beschreibung von Gegenständen mit dauerhafter Existenz
- ◆ **Beispiel:**

Use Case Text:

  1. Das System prüft anhand der Kundennummer, ob der Kunde dem System bekannt ist.
  2. Das System überprüft, ob der Kunde schon für eine Veranstaltung des angegebenen Seminartyps gebucht ist.

**UML-Icon:**  
(Stereotypen)



Buchungswunsch    Prüfungsvorgang    Kunde    Buchung    Veranstaltung

Folie 14

## 4.4.2. Die Erstellung eines Analyseklassendiagramms

- ◆ Die Schritte 3-5 können in beliebiger Reihenfolge oder auch parallel durchgeführt werden.
  - **Schritt 1: Klassen, Attribute und Assoziationen bestimmen**  
Klassenkandidaten aus den Use Cases (und evtl. aus vorhandenem ER-Diagramm) identifizieren. Zuweisung passender Namen und Attribute
  - **Schritt 2: Grundoperationen definieren**
  - **Schritt 3: Komplexe Operationen definieren**  
Die in den Use Cases beschriebenen Systemfunktionen in Form von Operationen auf die Klassen verteilen
  - **Schritt 4: Vererbung nutzen und komplexe Assoziationen wie Aggregationen beschreiben**
  - **Schritt 5: Klassendiagramm konsolidieren**

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- ▶ 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

## 4.4.2. Schritt 1: Regeln für Klassen

- ◆ **Regel 1: Entitätsklassen**
  - Entitätsklassen (siehe E/R-Diagramm) spiegeln Dinge oder Sachverhalte aus dem Anwendungsbereich wieder (z.B. „Kunde“ oder „Veranstaltung“). Typischerweise sind das:
    - Informationen, die verwaltet oder ausgetauscht werden
    - Materialien, die ausgetauscht werden
- ◆ **Regel 2: Steuerungsklassen**
  - Systemfunktionen werden im Analyseklassendiagramm zuerst durch Steuerungsklassen modelliert. Dies sind Platzhalter für die Modellierung komplexer Operationen im Schritt 3

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- ▶ 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

## 4.4.2. Schritt 1: Regeln für Attribute

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

- ◆ Die zwei Grundregeln zur Unterscheidung von Klassen und Attributen lauten:
  - Klassen haben eine **eigene Identität und mehrere Attribute**. Diese Attribute werden für komplexere Berechnungen oder Überprüfungen gebraucht, die später als Teile komplexer Operationen modelliert werden.
  - Attribute modellieren einen (**evtl. veränderlichen**) Wert, der unabhängig von der Klasse nicht sinnvoll ist.
- ◆ Beispiel Datum
  - Ein **Attribut** Datum ist sinnvoll, wenn das Datum nur als zusammenhängender Wert abgespeichert, verändert und abgefragt wird und nur einfache Operationen damit durchgeführt werden.
  - Eine **Klasse** Datum ist angemessen, wenn damit komplexere Operationen durchgeführt werden (z.B. Schaltjahrprüfung, Veränderungen einzelner Tage, Monate u.ä.).

## 4.4.2. Schritt 1: Regeln für Assoziationen

### 4. Methode

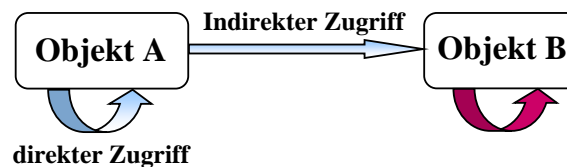
- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

- ◆ **Regel 1: Adjektive und Verben deuten auf Assoziationen hin**
  - Assoziationen in Analysenklassendiagrammen spiegeln Zusammenhänge aus dem Anwendungsbereich wider. Oft entsprechen diesen Zusammenhängen Adjektive oder Verben im Use Case Text. Der Zusammenhang zwischen Buchung und Veranstaltung könnte sich z.B. ergeben aus den Textteilen „die Veranstaltung ist gebucht“ oder „gebuchte Veranstaltung“.
- ◆ **Regel 2: Möglichst wenig redundante Assoziationen**
  - Gibt es zwischen einer Gruppe von Klassen nur zweistellige Assoziationen (d.h. Assoziationen zwischen jeweils 2 Klassen dieser Gruppe), ist zu überlegen, ob man einige davon nicht aus anderen ableiten kann (z.B. Ableitung von Kunde-Veranstaltung aus Kunde-Buchung und Buchung-Veranstaltung).

4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

- ◆ Ein Objekt kann Attributwerte nur durch den Aufruf von Methoden dieser Objekte lesen oder verändern.
- ◆ Es sind zwei Arten von Attributzugriffen durch Methoden zu unterscheiden:
  - Die Methode eines Objekts greift auf dessen eigene Attribute zu (**direkter Zugriff**).
  - Die Methode eines Objekts greift auf die Attributwerte anderer Objekte durch Aufruf von Methoden dieser Objekte zu (**indirekter Zugriff**).



4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

- ◆ Die Unterscheidung von direktem und indirektem Zugriff ermöglicht die Unterscheidung von Grundoperationen und komplexen Operationen.
  - Eine **Grundoperation** liest und verändert Attribute nur direkt.
  - Eine **komplexen Operation** liest und verändert Attribute auch indirekt.
- ◆ Um hohe Kohäsion und niedrige Kopplung zu erreichen, sollte es möglichst wenige komplexe Operationen geben.

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- ▶ 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

### ◆ Regel 1: Verben deuten auf Grundoperationen hin

- Grundoperationen spiegeln Überprüfungen oder Berechnungen von und mit Attributen eines Objekts wider. Oft sind diese Überprüfungen oder Berechnungen mit entsprechenden Verben im Use Case Text genannt. Oft müssen sie aber auch ergänzt werden aus der Kenntnis der Attributbedeutung.

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- ▶ 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

### ◆ Regel 1: Verteile Steuerungsklassen auf Entitäts- und Dialogklassen

- **Kanonische Lösung**, falls Steuerungsklasse nur auf Attributen (und Beziehungen) einer anderen Klasse arbeitet (z.B. Stornierungsvorgang und Buchung)
- **Einfache Lösung**, falls als Eingabe nur Attribute einer Klasse (z.B. Prüfungsvorgang und Buchungswunsch) oder als Ausgabe nur Attribute einer Klasse verwendet werden
- Keine „beste Lösung“, wenn Attribute mehrerer Klassen als Eingaben oder als Ausgaben verwendet werden

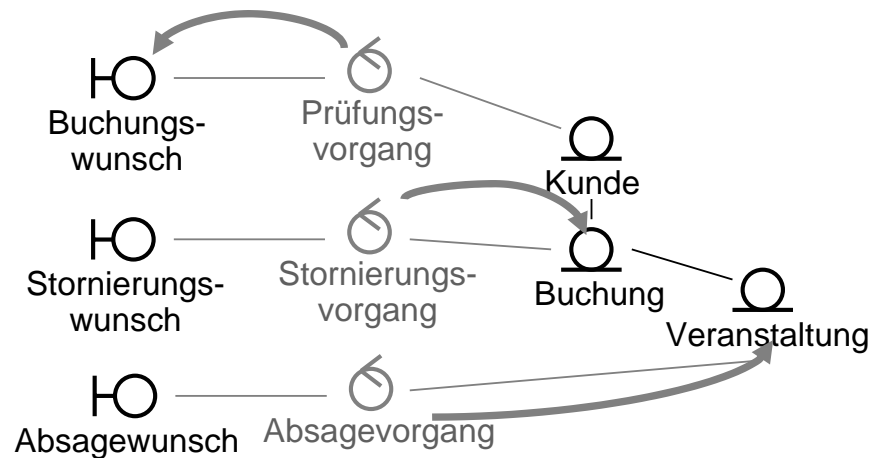
### ◆ Regel 2: Fasse evtl. Operationen zusammen oder teile sie weiter auf

- Operationen können mehrere UC-Schritte umfassen (Achtung: nur bei kanonischer Lösung)
- UC-Schritte evtl. noch in weitere Operationen aufspalten (falls auch eigenständig sinnvoll)

## 4.4.2. Schritt 3: Beispiel: Verhalten zu Klassen zuordnen

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- ▶ 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test



Folie 23

## 4.4.2. Schritt 3: Prüfung durch Sequenzdiagramm

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- ▶ 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

- ◆ Die vorgenommene Verteilung der Operationen sollte durch Erstellung von Sequenzdiagrammen für jeden Use Case und für jede Systemfunktion überprüft werden:
  - **Zu hohe Kopplung:** Sind für die Umsetzung sehr viele Objekte und Nachrichten notwendig?
  - Falls ja: kann die Anzahl durch Umverteilung von Operationen verringert werden?
  - **Zu niedrige Kohäsion:** Gibt es Operationen einer Klasse, die auf getrennten Attributbereichen arbeiten?
  - Falls ja: wird die Komplexität der Sequenzdiagramme durch eine Aufteilung dieser Operationen auf 2 Klassen erhöht?

Folie 24

## 4.4.2. Schritt 3: „Objektifizierung“ von Verhalten

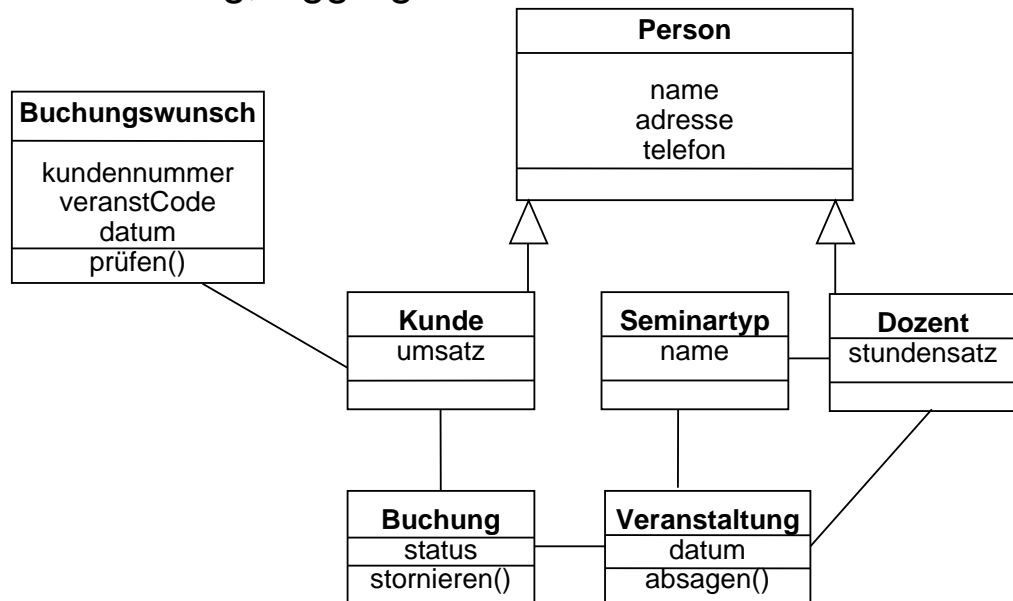
### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

- ◆ Manchmal ist es sinnvoll, Steuerungsklassen in der Implementierung beizubehalten
  - Vermeidung zu großer Implementierungsklassen
  - Austauschbarkeit von alternativen Vorgängen durch eine Vererbungshierarchie von Steuervorgängen
  - Speicherbarkeit von Zwischenzuständen lang andauernder Vorgänge
  - Wiederverwendbarkeit derselben Vorgangsklasse für unterschiedliche Implementierungen
  
- ◆ Objektifizierung von Verhalten bringt
  - Flexibilität in der Implementierung
  - Entkopplung
  
- ◆ **Aber: Organisatorischer Zusatzaufwand und sollte daher nur eingesetzt werden, wenn sinnvoll begründbar!**

## 4.4.2. Schritt 4: Klassendiagramm vervollständigen

- ◆ Vererbung, Aggregation



### 4. Methode

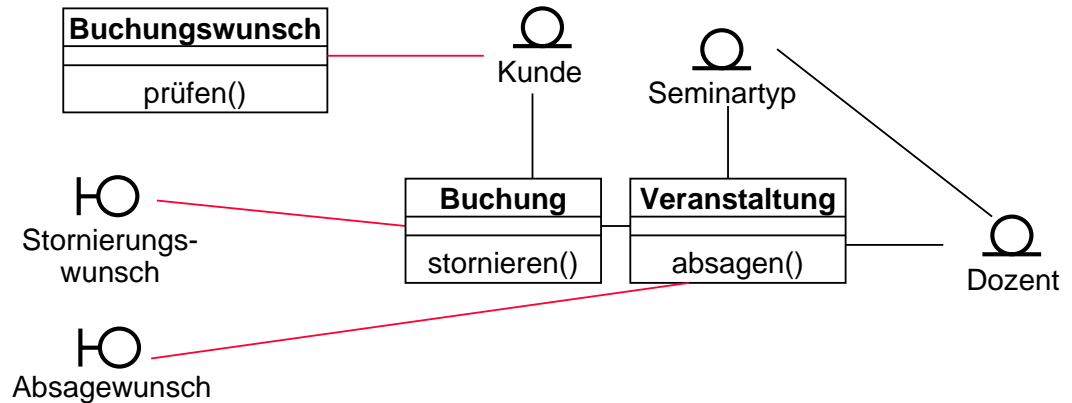
- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

## 4.4.2. Schritt 5: Regeln für Umgang mit Dialogklassen

- Regel 1: Dialogklassen beibehalten, wenn eigene Operationen oder wichtige Attribute
- Regel 2: Andernfalls Dialogklassen einer Klasse zuordnen
- Regel 3: Dialogklassen in eine separate Einheit legen (Dialogschnittstelle)

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test



## 4.4.2. Schritt 5: Regeln Assoziationen konsolidieren

- ◆ **Regel 1: Alle notwendigen Kommunikationswege abdecken**
  - Jede Klasse mit einer komplexen Operation Op muss eine Assoziation zu allen Klassen haben, deren (Grund-) Operationen bei der Ausführung von Op benötigt werden.
- ◆ **Regel 2: Keine unnötigen Kommunikationswege**
  - Es ist zu überprüfen, ob Assoziationen, die in keinem Use Case als Kommunikationsweg genutzt werden, wirklich sinnvoll sind.

### 4. Methode

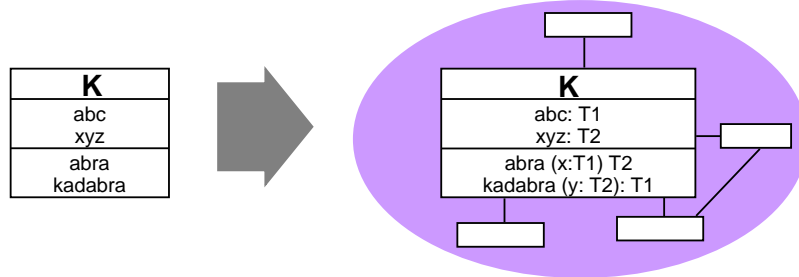
- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

## 4.4.3. Erstellung von Entwurfsmodellen

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

- ◆ Ziel: Vorbereitung der Codierung durch Berücksichtigung der Implementierungsziele
- ◆ Verfeinerung der Klassenmodelle durch Infrastrukturklassen und Vervollständigung



- ◆ Verfeinerung der Statecharts (evtl. direkt in Code)

## 4.4.3. Entwurfs- vs. Analysemodell

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

Analyse-Modell	Entwurfs-Modell
Objekte: <b>Fachgegenstände</b>	Objekte: <b>Softwareeinheiten</b>
Klassen: <b>Fachbegriffe</b>	Klassen: <b>Schemata</b>
Vererbung: <b>Begriffsstruktur</b>	Vererbung: <b>Programmableitung</b>
<b>Annahme perfekter Technologie</b>	<b>Erfüllung konkreter Implementierungsziele</b>
<b>Funktionale Essenz</b>	<b>Gesamtstruktur des Systems</b>
Meist projektspezifisch	Ähnlichkeiten zwischen verwandten Projekten
Grobe Strukturskizze	Genaue Strukturdefinition
	<b>Mehr Struktur &amp; mehr Details</b>

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

- ◆ Erweiterung des fachlichen Kerns: Mehr Details als im Analysemodell
  - Vollständige Listen der Attribute und Operationen
  - Festlegung von Datentypen, Sichtbarkeit
  - Spezifikation der Operationen (z.B. Vor- und Nachbedingungen)
  - Assoziationen/Aggregationen: Detaillierung, z.B. ordered
  - Keine Mehrfachvererbung
  
- ◆ Zusätzliche Klassen/Pakete:
  - Einbindung in Infrastruktur, Altsysteme etc.
  - Anpassungs- und Entkopplungsschichten für gewählte Technologien (z.B. Datenzugriffsschicht, CORBA-Schnittstellen, XML-Anschluss...)

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

- ◆ Definition: Die Spezifikation einer Operation legt das Verhalten der Operation fest, ohne einen Algorithmus festzuschreiben.
  
- ◆ Häufigste Formen von Spezifikationen:
  - Text in natürlicher Sprache (oft mit speziellen Konventionen)
    - Oft in Programmcode eingebettet (Kommentare)
    - Werkzeugunterstützung zur Dokumentationsgenerierung, z.B. "javadoc"
  - Vor- und Nachbedingungen
  - Tabellen, spezielle Notationen
  - "Pseudocode" (Programmiersprachenartiger Text)
    - nur mit Vorsicht zu verwenden - oft zu viele Details festgelegt !

## 4.4. Zusammenfassung Feinentwurf

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- ▶ 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

- ◆ Analyseklassenmodell als **Zwischenschritt** ermöglicht Konzentration auf das Wesentliche:
  - Analyseklassenmodell: grobe Strukturentscheidungen
  - Entwurfsklassenmodell: Implementierungsvorgabe

## 4.4. Literatur

### 4. Methode

- 4.1. Allgemeines
- 4.2. Vorgehen RE
- 4.3. Vorgehen Architektur
- ▶ 4.4. Vorgehen Entwurf
- 4.5. Vorgehen Test

- ◆ B. Bruegge, A. Dutoit, Object-oriented Software engineering, Pearson, 2004