Fachhochschule Köln
Cologne University of Applied Sciences

Softwaretechnik

Fomuso Ekellem

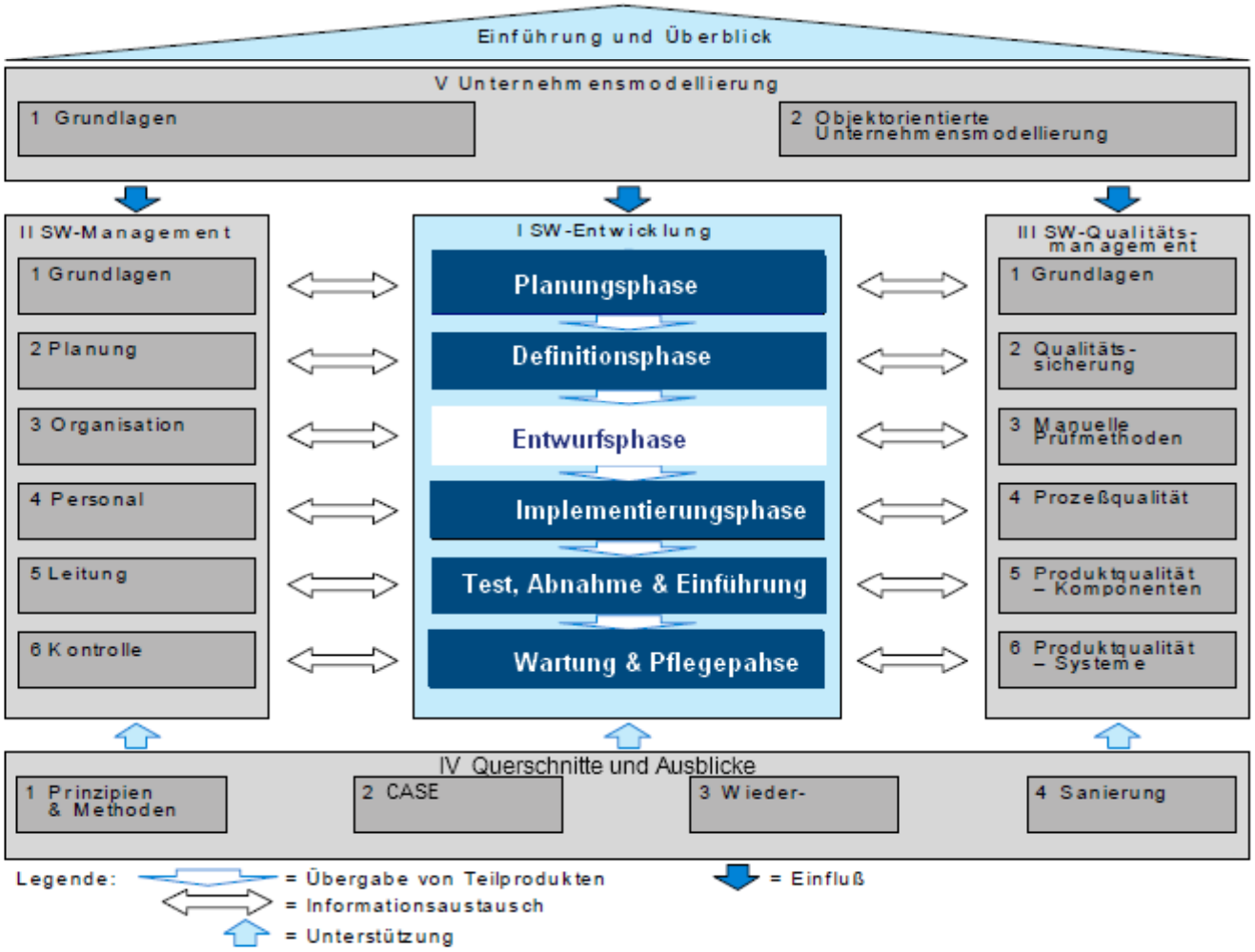
WS 2011/12

Inhalt

- **Entwurfsphase**
  - Systementwurf
  - Software Architektur Entwurf
  - Software Komponenten Entwurf
    - Struktur
    - Verhalten
  - OO Entwurf (OOD)

Entwurfsphase

In der Entwurfsphase werden die Spezifikationen der zu entwickelnden Software auf mögliche Software-Architekturen abgebildet.



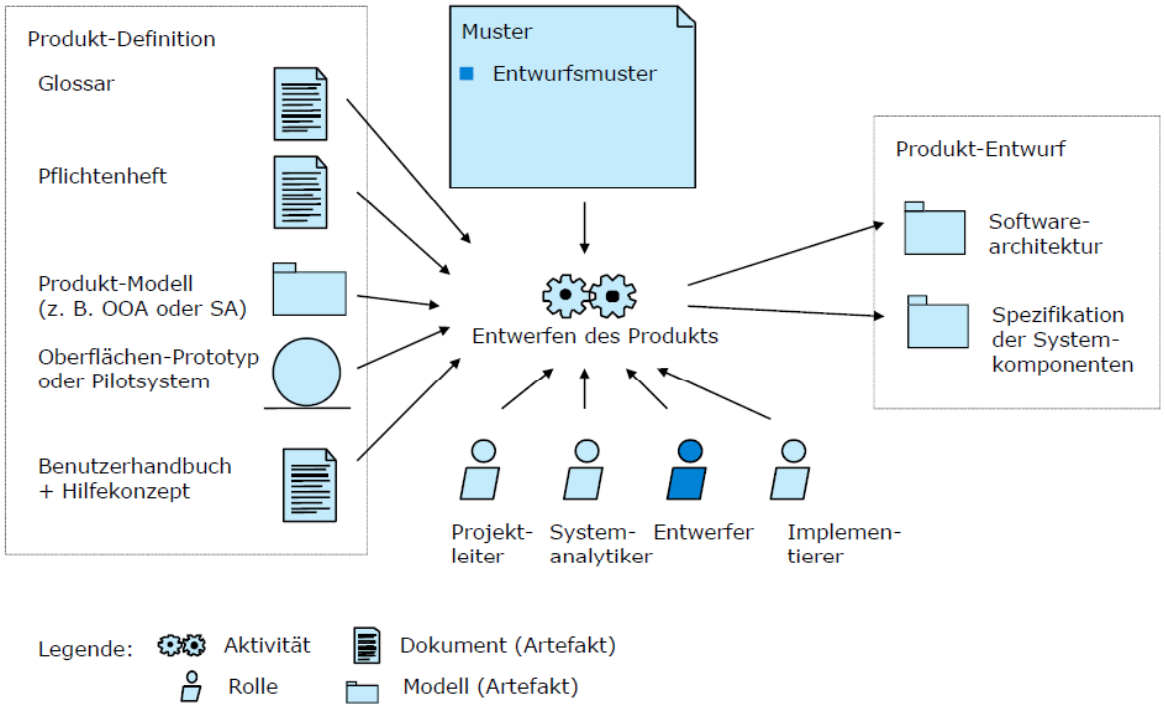
Entwurfsphase

Lernziele

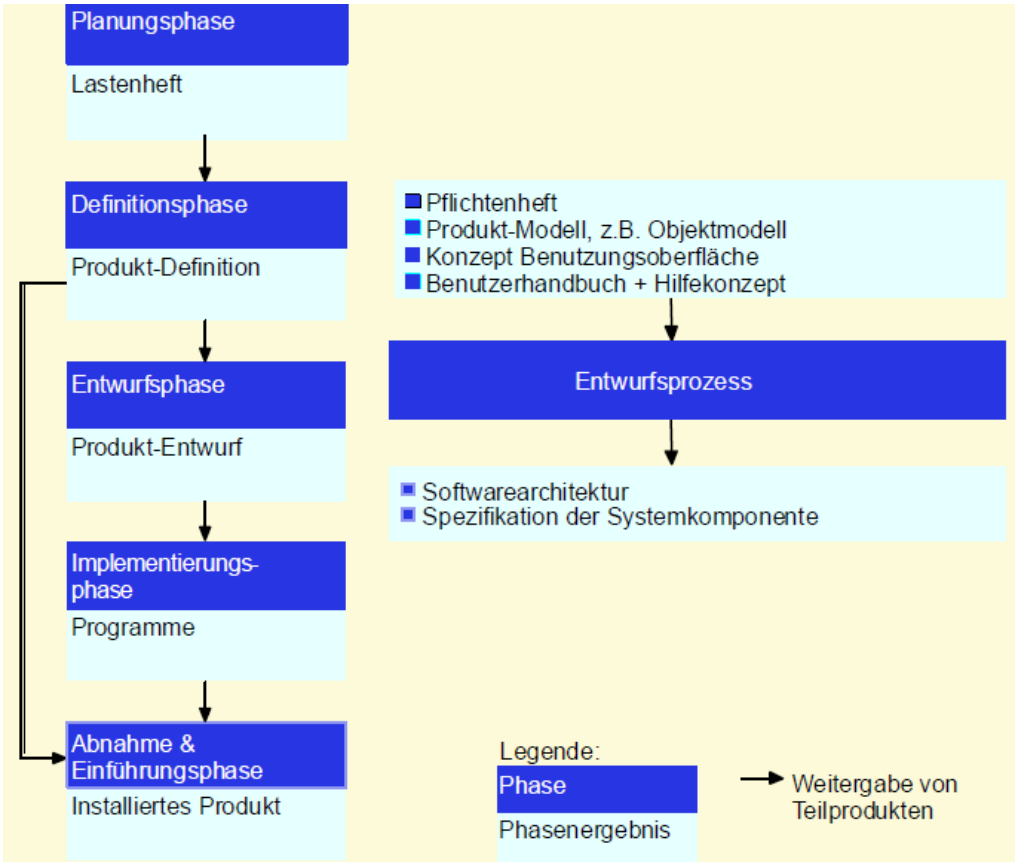
- Aufgaben der Entwurfsphase und notwendige Grundsatzentscheidungen
- Die wichtigsten Einflussfaktoren auf die Software-Architektur
- Alternativen bei den notwendigen Grundsatzentscheidungen
- Zusammenhänge zwischen Definitions-, Entwurfs-, und Implementierungsphase
- Aufbau und Charakteristika einer Schichtenarchitektur
- Client/Server Architekturen
- Anwendung der vorgestellten Entwurfskonzepte auf eine Problemstellung**

Entwurfsphase

- **Entwurfsprozess nach Balzert**

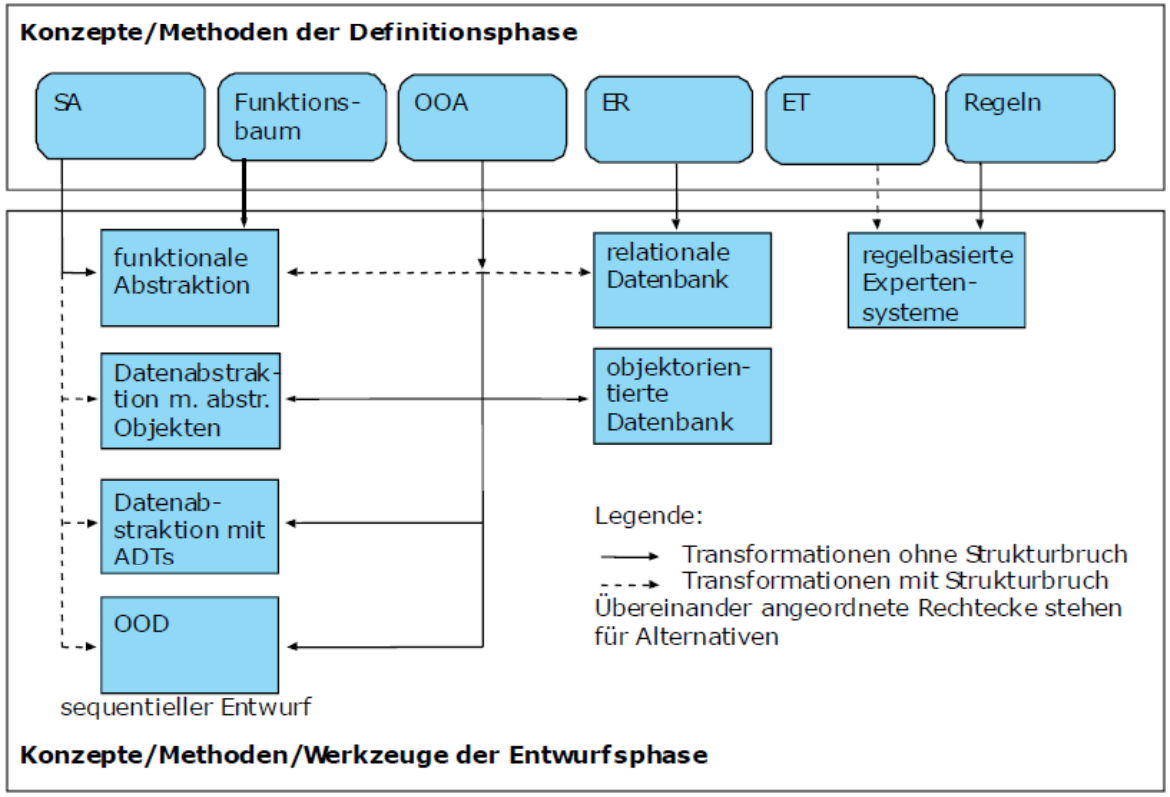


Entwurfsphase



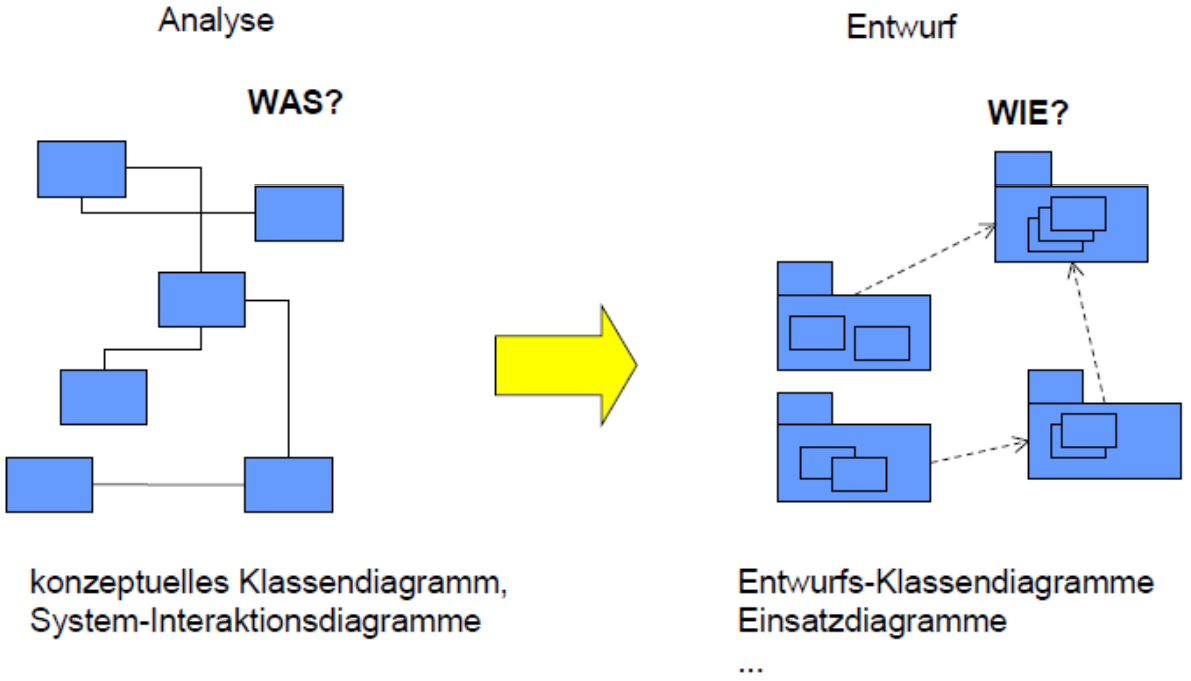
Entwurfsphase

- **Übergang von der Definition zum Entwurfsphase**



Entwurfsphase

- **Entwurf**



Aufgabe der Entwurfsphase

- Ziel des Entwurfs (Programmieren im Großen, design) ist es, ausgehend von der Produkt-Definition einen Produkt-Entwurf zu erstellen, der die Software-Architektur beschreibt und die Spezifikationen der Systemkomponenten enthält.
- Entwerfen einer Software-Architektur
  - Zerlegung des definierten Systems in Systemkomponenten
  - Strukturierung des Systems durch geeignete Anordnung der Systemkomponenten
  - Beschreibung der Beziehungen zwischen den Systemkomponenten.
- Spezifikation des Funktions- und Leistungsumfangs sowie des Verhaltens der Systemkomponenten.
  - informal, semiformal oder formal
- Festlegung der Schnittstellen, über die die Systemkomponenten miteinander kommunizieren.

Ergebnis der Entwurfsphase

- Ergebnis: (Produkt-Entwurf)
  - **Softwarearchitektur (Systemarchitektur):** Strukturierte oder hierarchische Anordnung der Systemkomponenten und ihre Beziehungen untereinander.
  - **Spezifikation der Systemkomponenten** : Festlegung von Schnittstelle Funktions- und Leistungsumfang

Entwurfsentscheidung

- Bevor mit dem eigentlichen Entwurf begonnen werden kann, müssen die Einsatzbedingungen und in Abhängigkeit vom Produkt selbst dann folgende Themenbereiche entschieden werden:

Grundsatzentscheidungen bzgl.

  - Betriebssystem
  - Hardwareplattform(en)
  - Programmiersprache
  - Datenhaltung (Persistenz)
  - Netzwerkverteilung
  - Benutzungsoberfläche (GUI)
  - Hilfesystem ... usw.

- ➡ Architekturentwurf

UML als Beschreibungsmittel

In UML vorhandene Beschreibungsmittel

- für Architekturentwurf:
  - logische Strukturen (Pakete, Paketdiagramme, Subsysteme, Schnittstellen)
  - physische Strukturen (Komponenten, Komponentendiagramme, *Knoten*, Einsatzdiagramme)
- für den Strukturentwurf (Software Komponente):
  - Klassendiagramme, Klassen
- für den Verhaltensentwurf (Software Komponente):
  - Interaktionsdiagramme, Zustandsdiagramme, Zustandsautomaten

Systemarchitektur

- Ein Architekturmodell (Strukturmodell) ist ein Model eines Datenverarbeitungssystems, das die statische Struktur der Komponenten eines Systems beschreibt.

Beispiele:

- Netzwerktopologie (Hardware)
- Blockschaltbild (Hardware)
- Funktionsbaum (Software)
- Einsatzdiagramm, Paketdiagramm (Software)
- Moduldiagramm (Hard/Software)
- Organigramm in einem Unternehmensmodell

Systemarchitektur

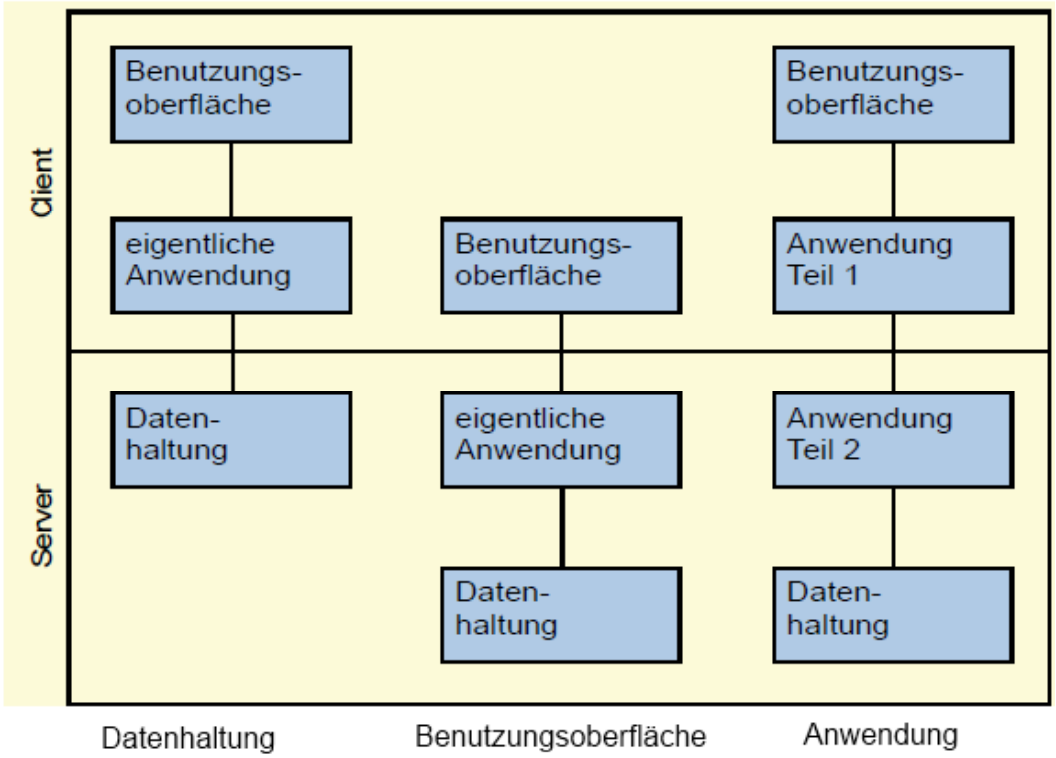
- Generelles Ziel muss es sein, vorhandene Systeme auf hohem Abstraktionsniveau einzusetzen und möglichst viele Dienstleistungen der jeweiligen Plattform in Anspruch zu nehmen. Oft werden die Systemkomponenten in einer logischen Schichtenarchitektur strukturiert. Viele Anwendungen bestehen dann aus einer 3-Schichten-Architektur:
  - Datenhaltung, Benutzungsoberfläche und Eigentliche Anwendung

Verteilungsmuster

- Die logischen Schichten werden auf physische Schichten nach einem Verteilungsmuster verteilt. Die physischen Schichten hängen davon ab, welche Architektur verwendet wird:
  - **Client/Server-Architektur**
  - Web-Architektur

Systemarchitektur

- **Typische Client/Server Konzept**



Software-Architekturen

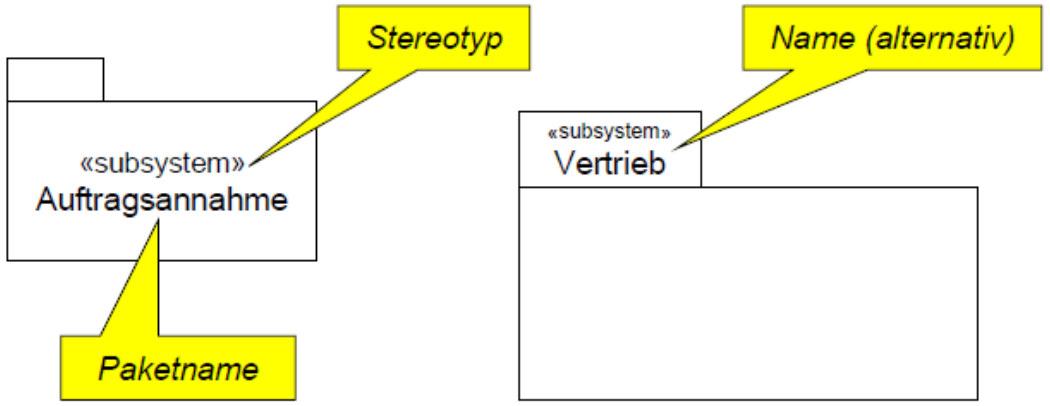
Logische Schicht

- Um eine Anwendung verteilen zu können, muss sie in Schichten strukturiert sein.
- Die Schichten einer Anwendung bezeichnet man auch als logische Software-Schichten.
- Beispiele:
  - Pakete
  - Paketdiagramme
  - Subsysteme
  - Schnittstellen

Software-Architekturen

Pakete

- Ein Paket ist ein genereller Mechanismus zum Gruppieren von Modell-Elementen (z.B. Diagrammen, Klassen, ...). Pakete können (rekursiv) in Pakete geschachtelt sein. Jedes Element ist in genau einem Paket enthalten (➜ Baum). Ein Paket definiert einen Namensraum für die in ihm enthaltenen Elemente.
- Das Gesamtsystem kann als ein Paket betrachtet werden.
- Spezielle Arten von Paketen

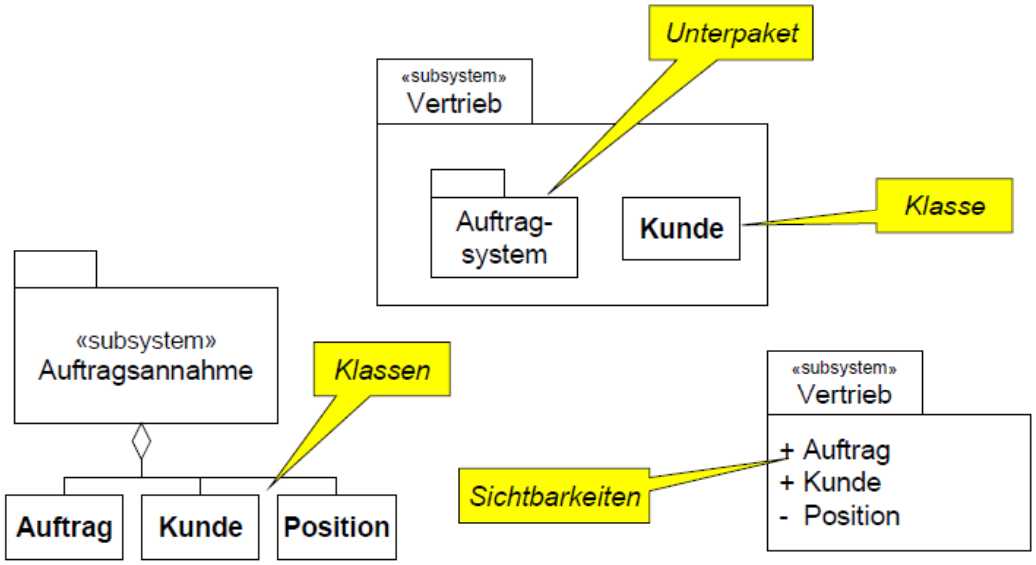


Software-Architekturen

- **Pakete:**

Möglichkeiten zur Darstellung des Inhalts von Paketen sind:

- Aggregation, Schachtelung und Namenslisten
- In C++ werden Pakete mit Namensräume unterstutzt.



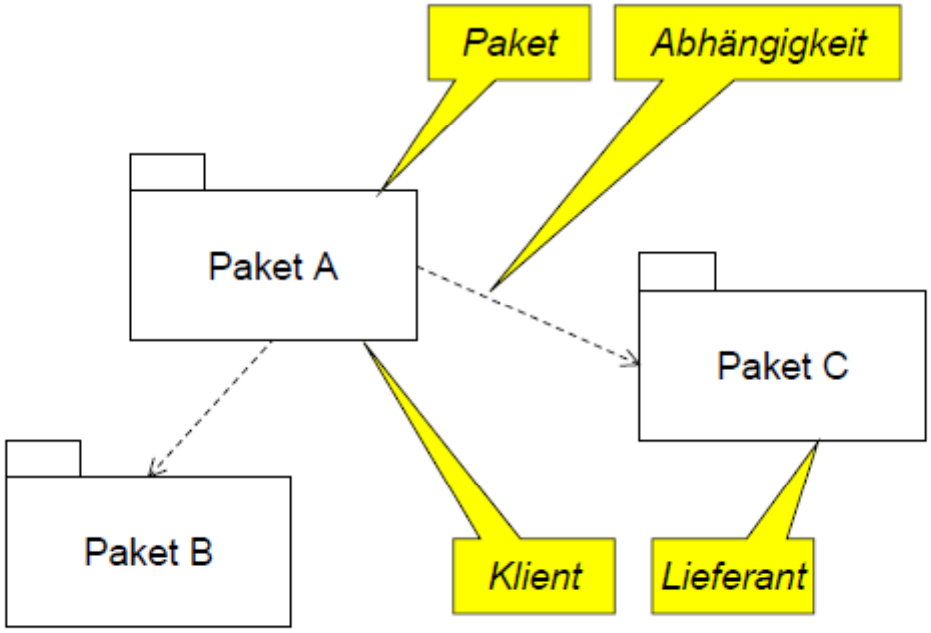
Software- Architekturen

Pakete erlauben:

- Kontrolle der Komplexität:
  - Ein Klassendiagramm sollte auf eine A4 Seite passen.
  - Ein Entwickler kann etwa 7 (+/– 2) Klassen auf einmal erfassen.
- Einschränkung der Namensräume
- Kontrolle der Propagierung von Änderungen
- Bau von geschachtelten Klassenhierarchien
- Bau von geschichteten Architekturen
- Sichtbarmachen von Abhängigkeiten auf höheren Abstraktionsniveau
- Aufteilung der Entwicklungsarbeit im Team
- Spezifikation von Schnittstellen zwischen Gruppen von Klassen (Subsystemen)

Software-Architekturen

- Paketdiagramme : Ein Paketdiagramm zeigt den Zusammenhang (*Abhängigkeiten*) *zwischen* verschiedenen *Paketen des Systems.*



Software-Architekturen

Subsysteme

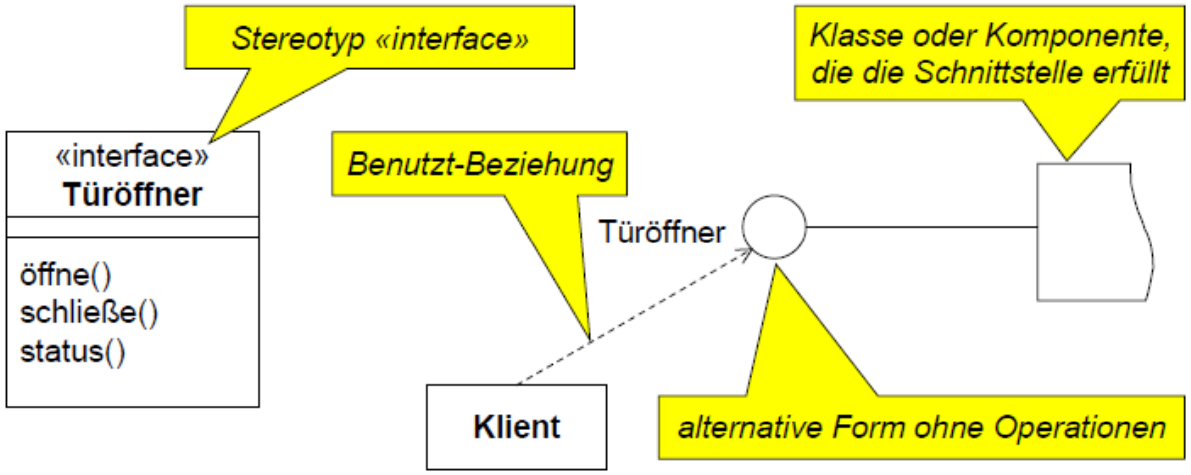
- **Wie breche ich ein großes System in kleinere Systeme herunter?**

funktionale Dekomposition	OO Pakete
Abbildung des Gesamtsystems auf Funktionen und Unterfunktionen, beginnend beim Anwendungsfall	• Zusammenfassung von Klassen zu Subsystemen • Schichtung von Subsystemen • **Abstraktion:** Konzentration auf das Wesentliche, Verdrängen des Unwesentlichen. • **Lokalität:** Physische Zusammenfassung von Zusammengehörigem (Daten und Algorithmen). • **Verdecken:** Beschränkung der Sichtbarkeit von Details auf diejenigen Teile eines Systems, die benötigt werden.

Software-Architekturen

Schnittstellen

- Eine Schnittstelle ist eine benannte Menge von Operationen, die das nach außen sichtbare Verhalten z.B. einer Klasse, Komponente oder eines Pakets definiert.
- Schnittstellen haben keine Implementierung, keine Attribute und keine Beziehungen.
- Schnittstellen können in Generalisierungsbeziehungen zueinander stehen.



Software-Architekturen

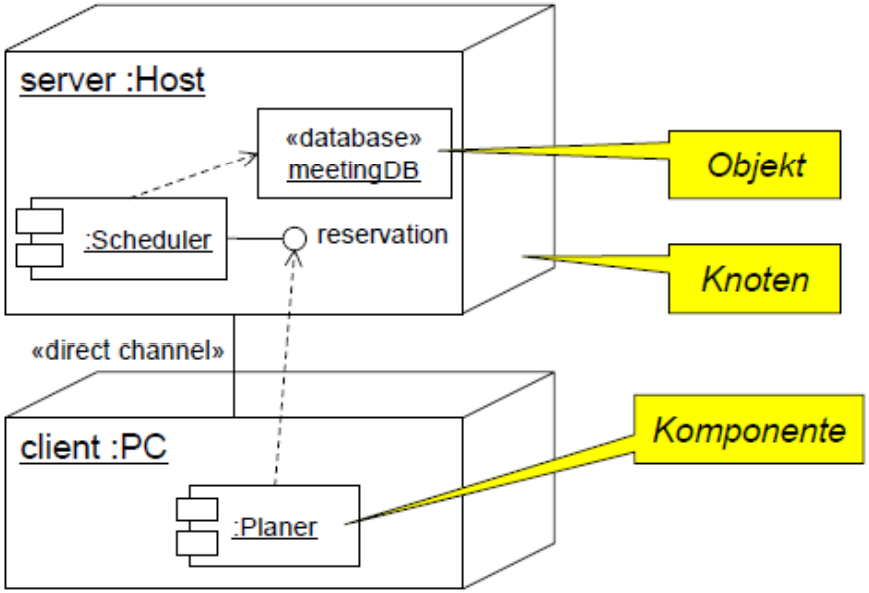
Physische Schicht

- Logische Schichten stellen die Modularisierungseinheiten einer Anwendung dar.
- Physische Schichten repräsentieren die Einheiten, die auf unterschiedliche Computerklassen (Arbeitsplatzcomputer, Abteilungsserver, Datenbankserver, Zentralcomputer) verteilt werden.
- Die 3-Schichten-Architektur kann zu einer n-Schichten-Struktur verfeinert werden.
- Beipiele:
  - Einsatzdiagramme
  - Komponenten
  - Komponentendiagramme
  - Knoten

Software-Architekturen

Einsatzdiagramme

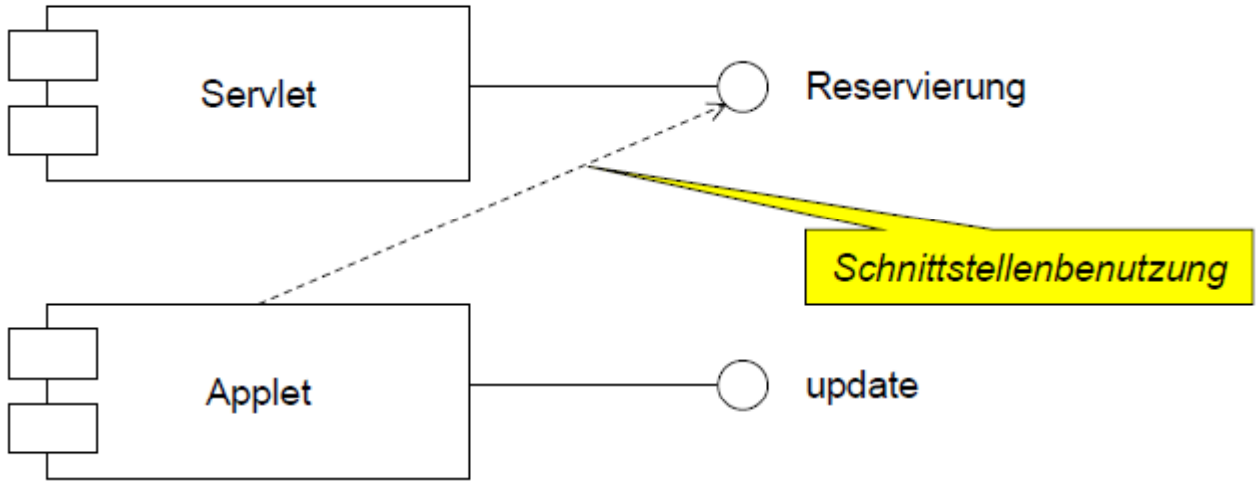
- Ein Einsatzdiagramm zeigt die Konfiguration von *Knoten zur Laufzeit sowie die Komponenten (Instanzen) und Objekte, die sich darauf befinden.*
- Komponenten, die nicht als Laufzeitobjekt (Instanz) existieren, sollen nur im Komponentendiagramm erscheinen.



Software-Architekturen

Komponentendiagramme

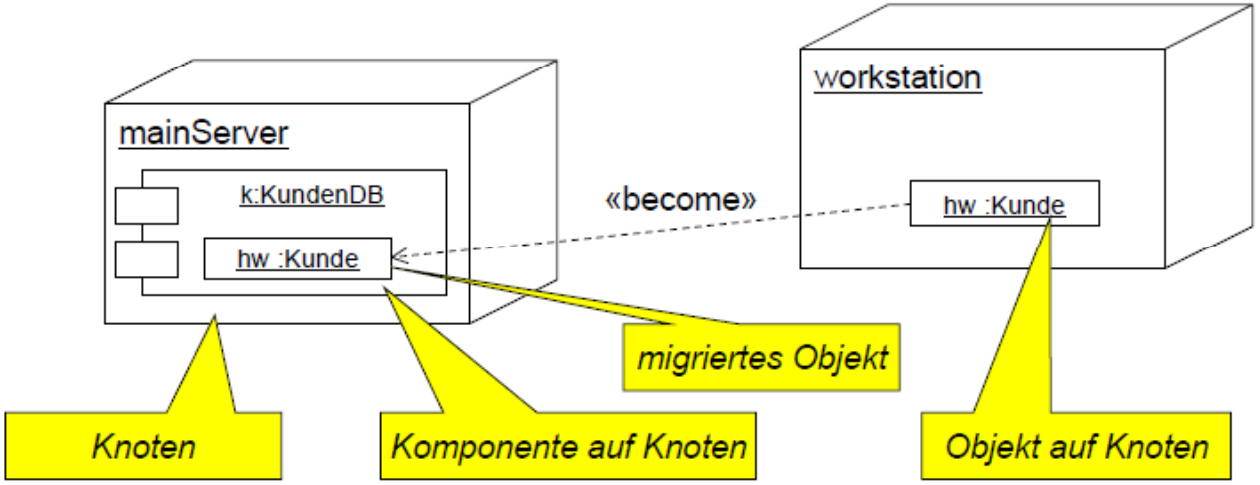
- Ein Komponentendiagramm beschreibt die Organisation und die Abhängigkeiten zwischen Komponenten eines Systems.



Software-Architekturen

Knoten

- Ein Knoten (*node) ist physikalisches Objekt, das zur Laufzeit existiert und* Bearbeitungsressourcen repräsentiert, die Speicher und erechnungskapazität haben. Auf einem Knoten können *Objekte und Komponenten angesiedelt sein.*
- Knoten können Rechner, aber auch Menschen oder (mechanische) Geräte oder Maschinen sein´(wichtig für Geschäftsmodelle).



Software-Entwurf

- Identifiziere und benenne Subsysteme.
- Lege Sichtbarkeiten für Klassen innerhalb jedes Subsystems fest.
- Definiere und überprüfe Abhängigkeiten zwischen Paketen und den Klassen- und Interaktionsdiagrammen.
- Falls erforderlich, definiere Schnittstellen, die in separaten, gemeinsam genutzten Paketen gespeichert werden.
- **Tips**
  - Sichten sind kein Bestandteil von UML, aber des *Unified Process*.
  - Sichten sind bei der Modellierung und Modellrevision wichtig.
  - Jedes Diagramme muß eindeutig genau einer Sicht zugeordnet sein.
  - Beim Lesen von Diagrammen muß man die Sicht des Zeichnenden kennen.
  - Die Implementierungssicht wird häufig verwendet, aber die anderen sind mindestens genauso wichtig!

Sichten der OO Modellierung

- **Konzeptuelle Sicht (frühe Analyse)**
  - Klassen repräsentieren die fachlichen Konzepte
  - sprach- und systemunabhängig
  - wenige Klassen, passt auf wenige Diagramme
- **Entwurfssicht (späte Analyse, früher Entwurf)**
  - Klassen repräsentieren SW-Schnittstellen
  - beschreibt die Lösung schwieriger Probleme der Implementierung (Risikominimierung)
  - Gliederung des Systems in Schichten, Subsysteme und Pakete
- **Implementierungssicht (Entwurf und Programmierung)**
  - zeigt die tatsächlich in der Programmiersprache verwendeten Klassen
  - wird direkt auf die Implementierung abgebildet
  - Beispiel: Abbildung von Assoziationen

OO Entwurf (OOD)

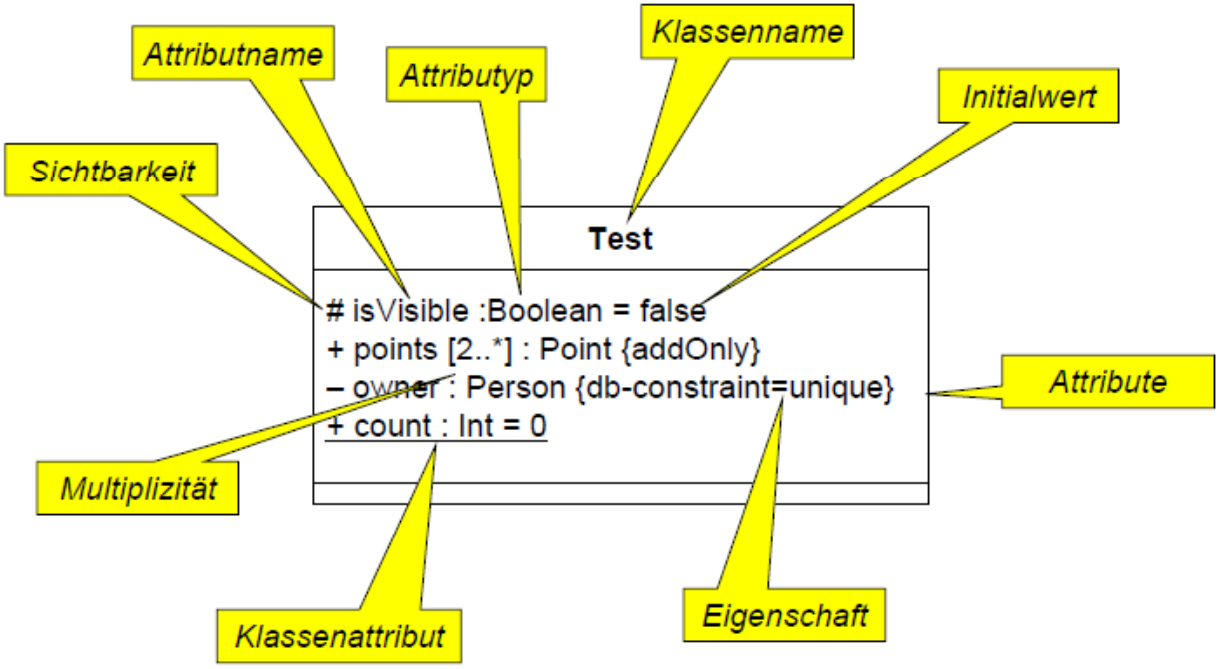
- Grundlage für den objektorientierten Entwurf ist in der Regel das OOA-Modell.
- Das entstehende OOD-Modell wiederum bildet die Grundlage für die Implementierung.
- Die Implementierung erfolgt in einer oder mehreren konkreten Programmiersprachen.
- **Bezeichner-Syntax**
  - Alle Namen des OOD-Modells müssen der Syntax der Ziel-Programmiersprache entsprechen.
  - Bezeichnungen aus dem **OOA-Modell (Produkt-Modell)**, die dieser Syntax nicht entsprechen, müssen daher manuell oder automatisch umgewandelt werden.

OO Entwurf

- Das Konzept der Klasse wird im Entwurf um generische Klassen, Container Klassen und Schnittstellen erweitert.
- Für **Attribute** und **Operationen** wird die Notation um Sichtbarkeit erweitert. Bei Operationen ist außerdem die komplette Signatur anzugeben.
- Die Notation von Assoziationen wird um die Navigation und die Sichtbarkeit erweitert. Assoziationen können auf verschiedene Arten (z. B. mittels Zeigern) realisiert werden.
- Der **Polymorphismus** ermöglicht es, flexible Programme zu entwickeln. Im Gegensatz zur Analyse tritt im Entwurf häufig **Vererbung** auf, wobei außer der Einfachvererbung auch die Mehrfachvererbung vorkommen kann.

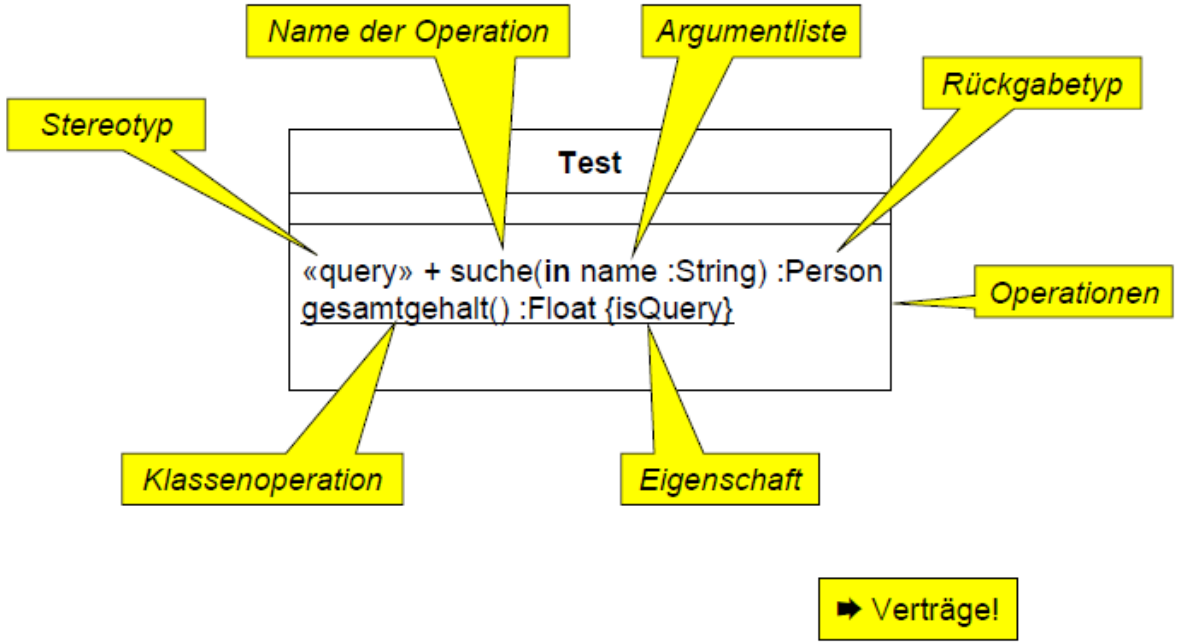
OO Entwurf - Attribute

- **Attribute**



OO Entwurf - Operationen

- **Operationen: Methoden**

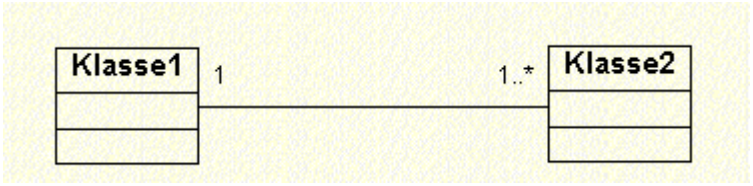


OO Entwurf- Beziehungen

- Eine prinzipielle OO-Sichtweise beschreibt das Verhältnis zwischen zwei aktiven Objekten als eine typische Client-Server Situation. Wenn ein Objekt irgendeine Aktion nicht aus dem eigenen Verhaltensrepertoire erfüllen kann, dann fordert es als **Client** mit einer **Botschaft** von einem anderen **Objekt (Server)** einen Dienst oder eine Leistung an. Realisiert wird die Client-Server Beziehung über eine der Referenzbeziehungen.
- Aus der Fülle der Beziehungssituationen sind folgende drei von besonderer Bedeutung: Vererbung, Aggregation und Assoziation
- Beziehungstypen
  - Vererbungsbeziehungen (Spezialisierung / Generalisierung)
  - Referenz- oder Nutzungsbeziehungen
  - Kardinalitäten : Die Kardinalität (K1, K2) legt fest, wie viele Exemplare aus K2 einem Exemplar aus K1 zugeordnet werden können.

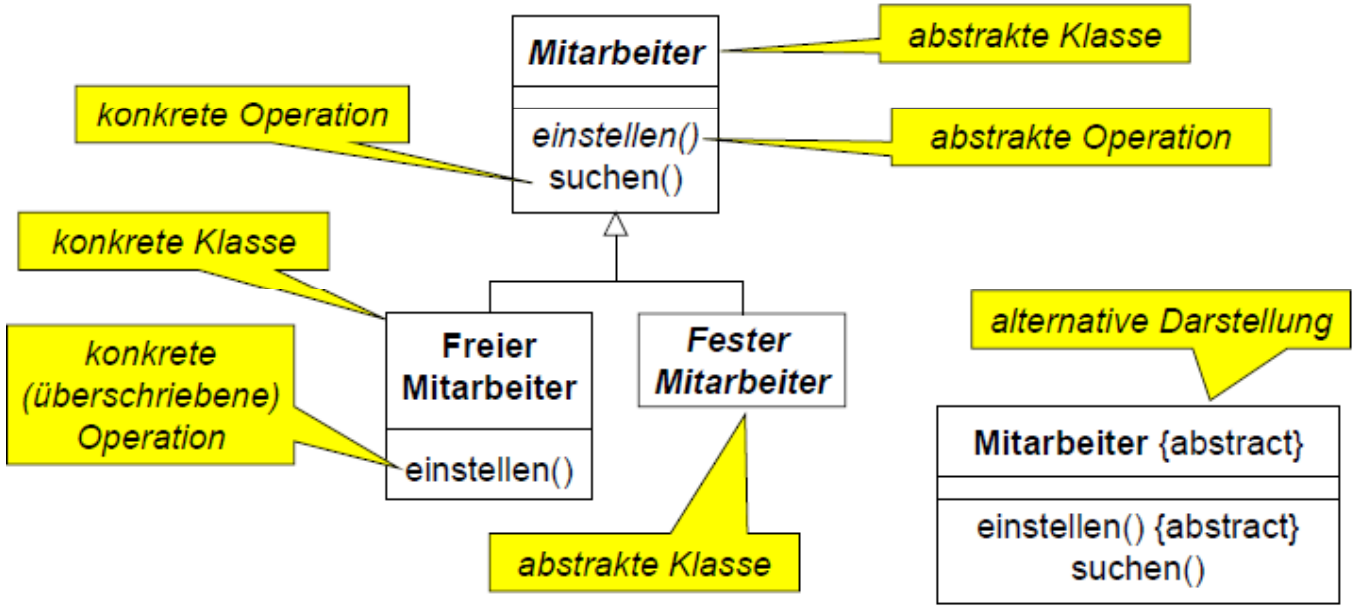
OO Entwurf - Kardinalität

- **Dabei werden drei Grundtypen unterschieden:**
  - 1 : 1
  - 1 : n
  - m : n
- **Andere Notationen (UML) sind:**
  1 : genau eine
  0,1 : konditionell, keine oder eine
  * : multiple, keine Einschränkung
  0..* : multiple
  1..* : mindestens eine



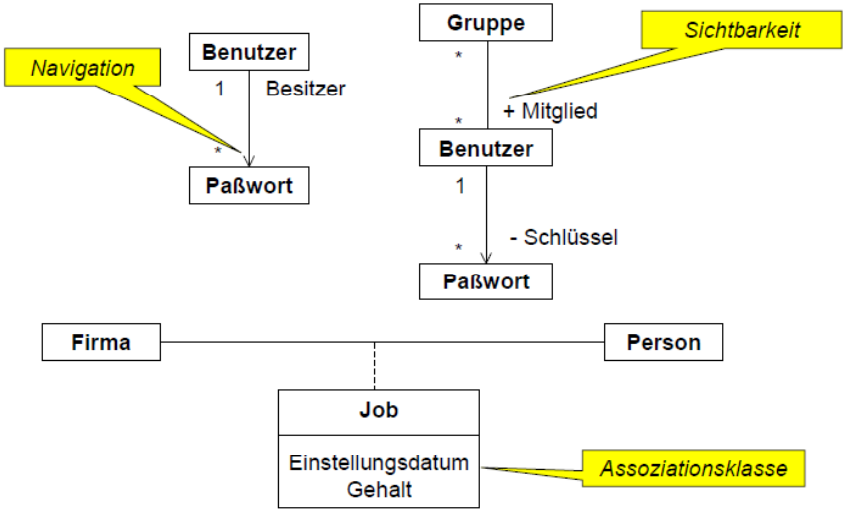
OO Entwurf - Vererbung

- **Vererbung: Eine oder mehrere Kind Klassen erbt von eine Basis Klasse**
- **Darunter auch Mehrfacher-Vererbung(Kind erbt von mehrere)**



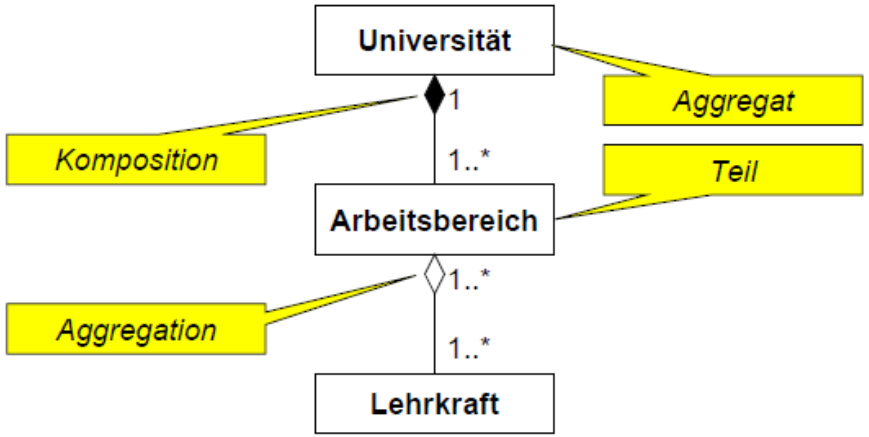
OO Entwurf- Assoziationen

- Assoziationen : drückt das Verhältnis von zwei völlig selbständigen Objekten aus, die auf der gleichen Abstraktionsebene stehen und eigentlich nichts miteinander zu tun haben, aber unter bestimmten Gesichtspunkten in eine lose Kennt-Beziehung ('KNOWS') gebracht werden können.
- Bei einer m : n-Beziehung wird dazu eine neue Klasse gebildet, deren Exemplare (Linkobjekt) die aktuelle Beziehung realisieren.



OO Entwurf – Aggregation und Komposition

- Komposition (echte Aggregation): Komponenten sind vom Aggregat existenzabhängig z.B. Arbeitsbereich ist Teil eine Universität.
- Aggregation (einfache Aggregation):Aggregat und Komponenten sind nicht existenzabhängig
- Häufig lässt sich nur schwierig festlegen, ob eine Aggregation oder Assoziation vorliegt.



OO Entwurf

- **Zustandsautomaten (Beispiel System Verhaltenskomponente)**