Software Engineering

Entwurf

Prof. Dr. Bodo Kraft

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Agenda und Quellen
Anforderungsanalyse mit UML

Motivation und Einordnung

Klassendiagramme im strukturellen Entwurfsmodell

Vom Analysemodell zum Entwurfsmodell

Pakete im strukturellen Entwurfsmodell

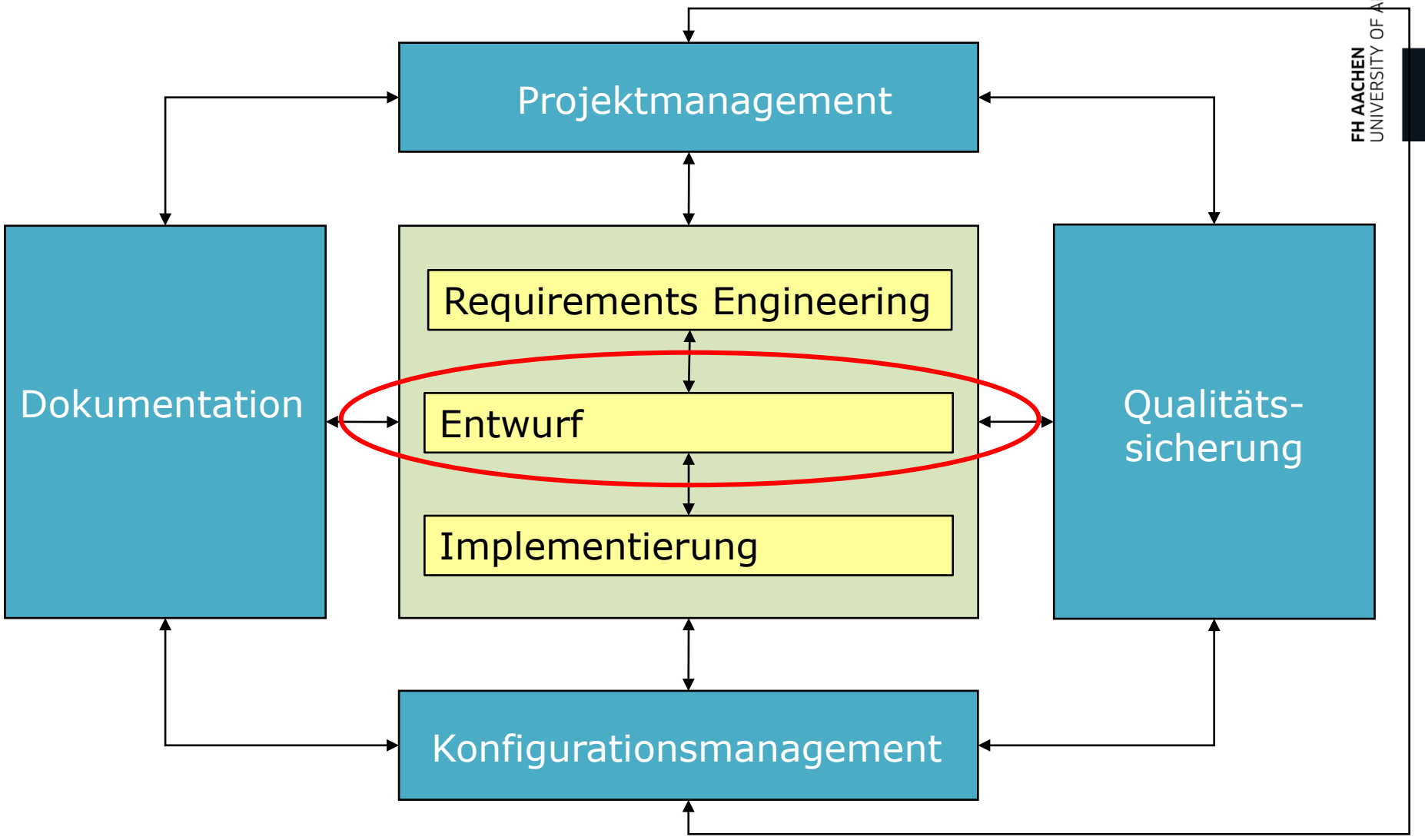
Verhaltensdiagramm im Entwurfsmodell

Quellen

Vorlesung von
Prof. Westfechtel
Uni Beireuth

Arbeitsbereiche und Disziplinen

Motivation und Einordnung



Der Begriff „Programmieren im Großen“
Motivation und Einordnung

Charakterisierung

- *We argue that structuring a large collection of modules to form a “system” is an essentially different intellectual activity from that of constructing the individual modules.*

  [DeRemer 1976]

Definition

- *Alle Aktivitäten oberhalb der Realisierung einzelner Module, insbesondere die Definition und Modifikation der Gesamtstruktur (Gesamtarchitektur) eines Softwaresystems entsprechend der Anforderungsspezifikation*

  [Nagl 1990]

Definition des Begriffs „Architektur“
Motivation und Einordnung

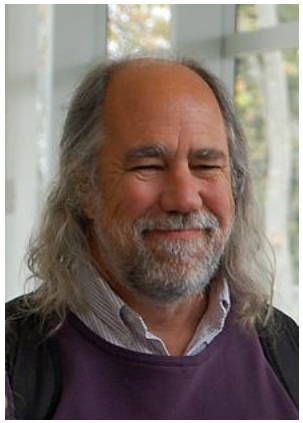
*An architecture is the set of **significant decisions** about the organization of a software system,*

*the selection of the **structural elements** and their interfaces by which the system is composed, together with*

*their **behavior** as specified in the collaborations among those elements,*

*the **composition of these structural** and behavioral elements*

*into progressively **larger subsystems** and the architectural style that guides this organization.*



[Grady Booch 1999]

Typische Bestandteile einer Softwarearchitektur
Motivation und Einordnung

- Anwendungsspezifische Funktionen (Applikationslogik)
- Details der Benutzerschnittstelle (GUI-Bibliotheken)

- Ablaufsteuerung (Transaktionen, Workflowmanagement)
- Datenhaltung (in Dateien, Datenbanken)
- Infrastrukturdienste für
  - Objektverwaltung (Freispeichersammlung, verteilte Objekte)
  - Prozesskommunikation
- Sicherheitsfunktionen (Verschlüsselung, Passwortschutz)
- Zuverlässigkeitsfunktionen (Fehlererkennung und –behebung)
- Systemadministration (Statistiken, Installation, Sicherung)
- Weitere Basisbibliotheken (arithmetische Funktionen)

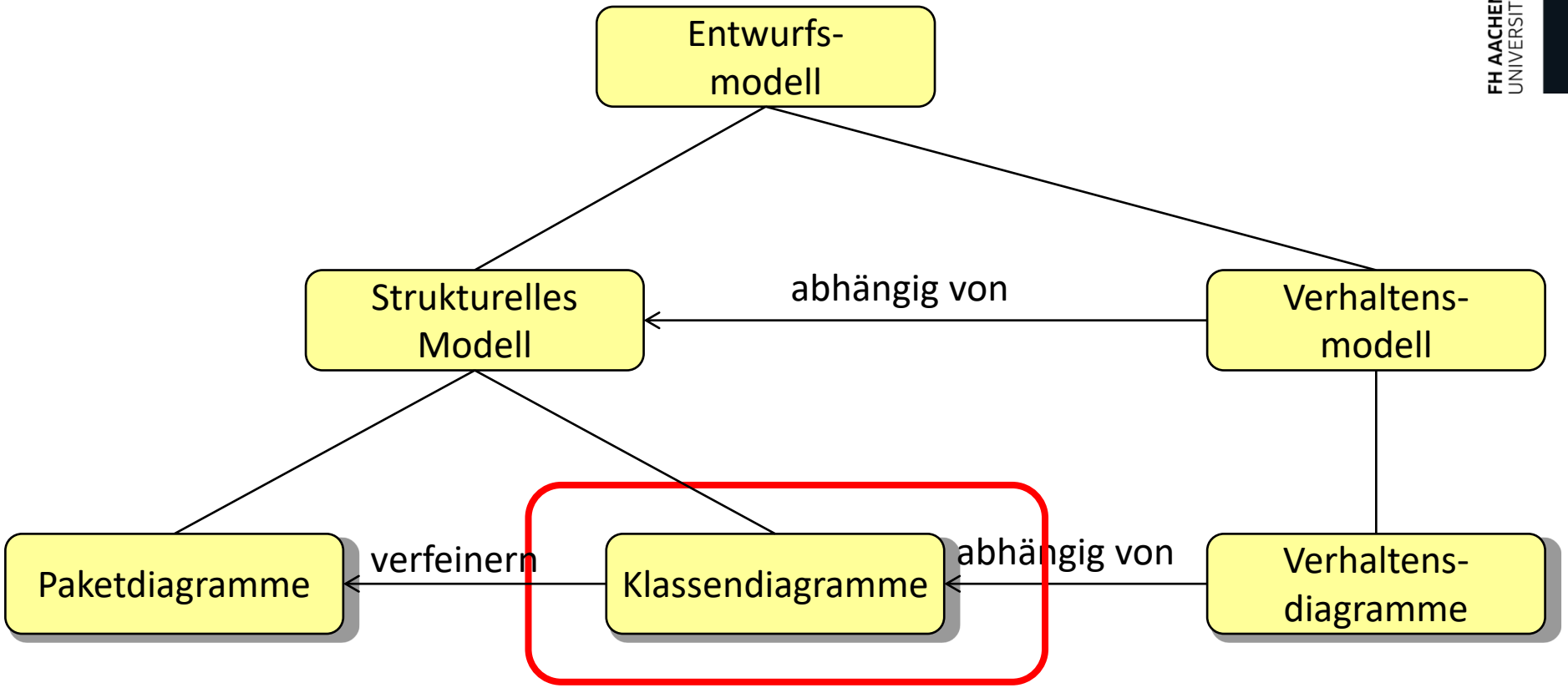
…

Funktion der Architektur im Software-Lebenszyklus

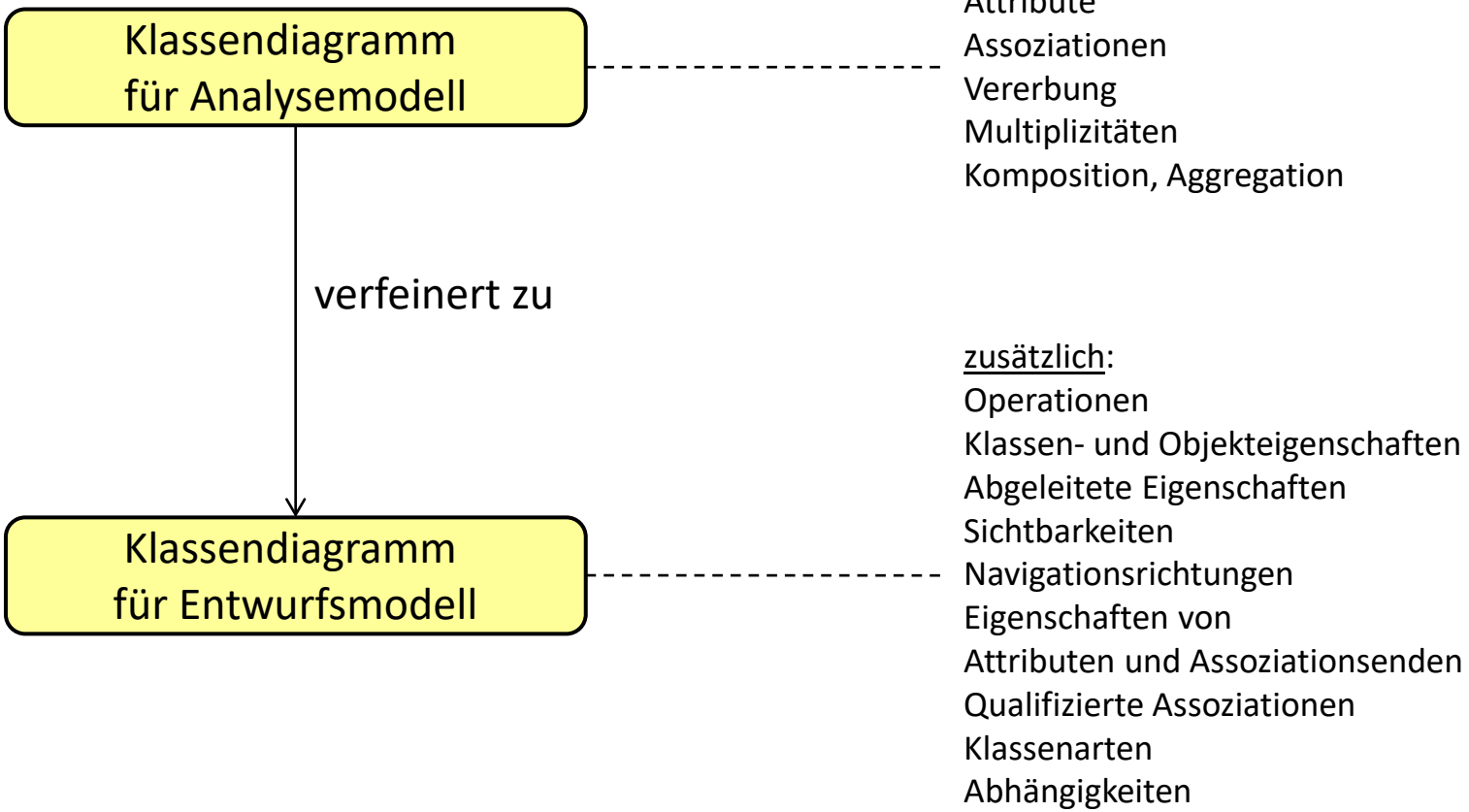
Motivation und Einordnung

Bauplan	Spezifikation des zu implementierenden Systems sowohl auf grober als auch auf feiner Ebene
Projektplanung	Definition von Arbeitspaketen zum Implementieren und Testen Definition von Meilensteinen Aufteilung des Projektteams nach Architekturkomponenten Fortschrittskontrolle
Testen	Festlegung von Teststrategien Ableitung von Testfällen
Nachvollziehbarkeit	Management von Beziehungen zu Anwendungsfällen bzw. Funktionen der Anforderungsspezifikation
Wartung	Verstehen des Systems Analyse der Auswirkung von Änderungen Planung von Änderungen

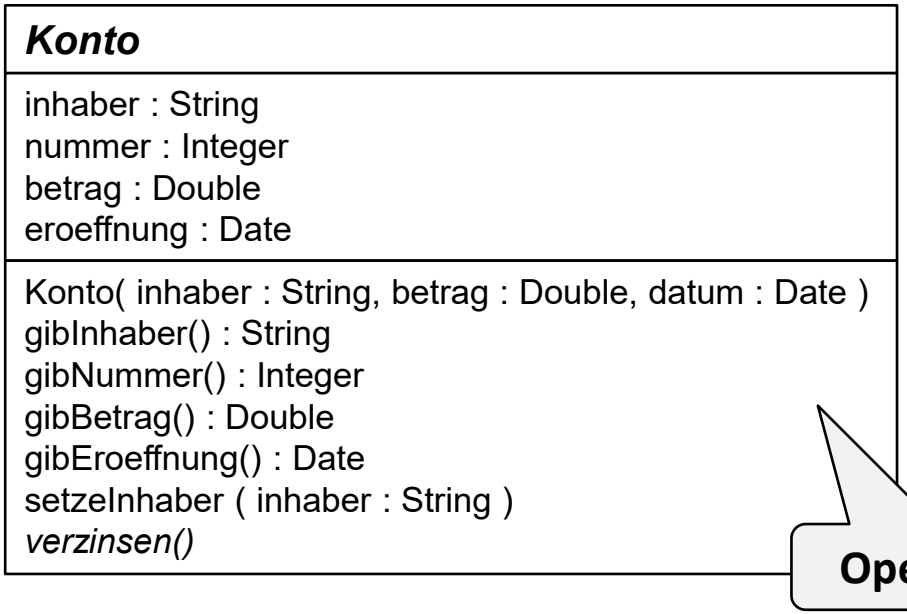
Aufbau des Entwurfsmodells
Motivation und Einordnung



Detaillierung von Klassendiagrammen
Klassendiagramme im strukturellen Entwurfsmodell



Modellierung des Verhaltens im Klassendiagramm
Klassendiagramme im strukturellen Entwurfsmodell



- Konstruktoren werden in der UML wie gewöhnliche Operationen behandelt
- Abstrakte Operationen werden gekennzeichnet durch
  - *Kursivschrift*
  - oder durch `{abstract}`

Syntax:

```
Operationsdeklaration = Operationsname [Parameterliste] [":" Rückgabetyp]
Parameterliste = "(" [Parameter {"," Parameter}] ")"
Parameter = [Richtung] Parametername ":" Typname [Multiplizität][Standardwert]
Richtung = ["in" | "inout" | "out"]
Standardwert = "=" Ausdruck
```

Syntaktische und semantische Unterschiede zu Java

Faustregeln für die Festlegung von Operationen
Klassendiagramme im strukturellen Entwurfsmodell

- Operationen für lesenden/schreibenden Zugriff auf Attribute dürfen fehlen (lassen sich aber später automatisch generieren)

- Konstruktoren dürfen ebenfalls fehlen, falls es sich nur um parameterlose Standard-Konstruktoren handelt (lassen sich auch automatisch generieren)

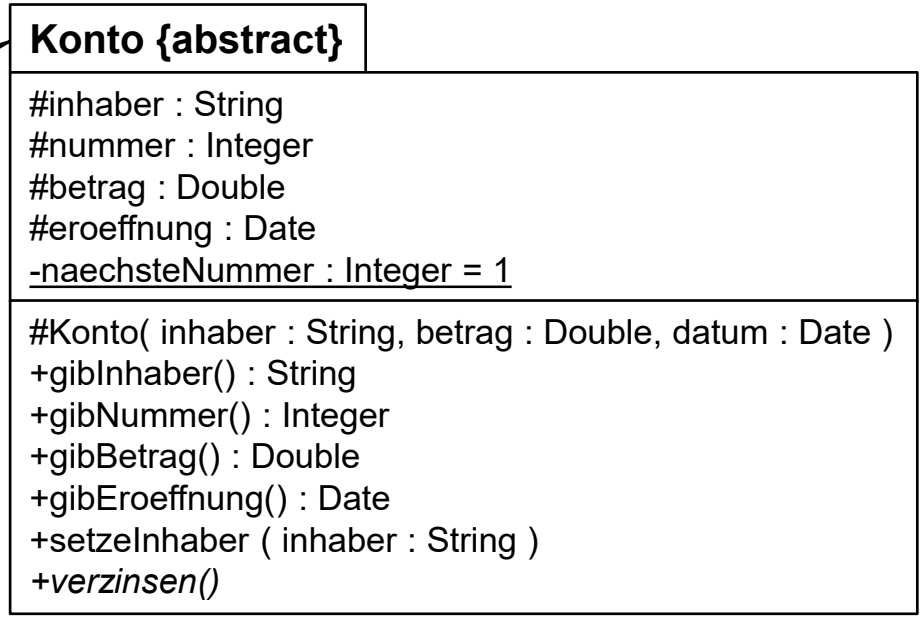
- Aktivitäten aus Anwendungsfallbeschreibungen oder Aktivitäts-diagrammen, die auf Instanzen genau einer Klasse operieren, werden dieser Klasse zugeordnet

- Andere Aktivitäten werden
  - in Teilaktivitäten zerschlagen oder
  - bei „umfassenden" Objekten deklariert oder
  - eigenen Transaktionsobjekten zugeordnet

Sichtbarkeiten in der UML
Klassendiagramme im strukturellen Entwurfsmodell

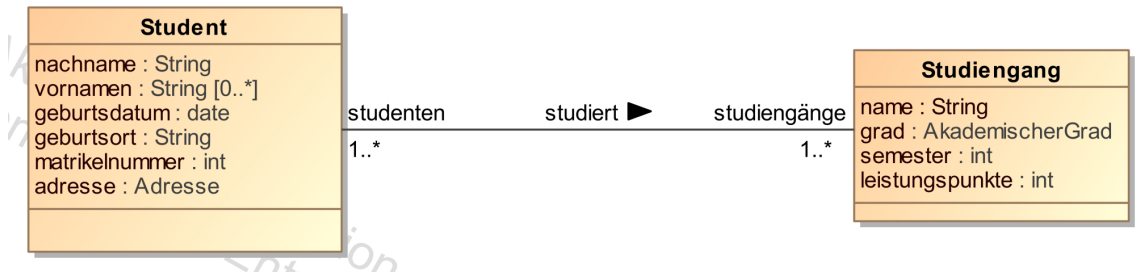
Sichtbarkeit	*Symbol*	*Erläuterung*
öffentlich	**+**	Element ist überall sichtbar
privat	**-**	Element ist nur in der gleichen Klasse sichtbar
geschützt	**#**	Element ist in der Klasse und ihren Unterklassen sichtbar
Paket	**~**	Element ist überall im gleichen Paket sichtbar

Abstrakter Datentyp



Navigierbarkeit
Klassendiagramme im strukturellen Entwurfsmodell

- Eine Assoziation ist navigierbar, wenn Objekte der adjazenten Klasse erreichbar sind



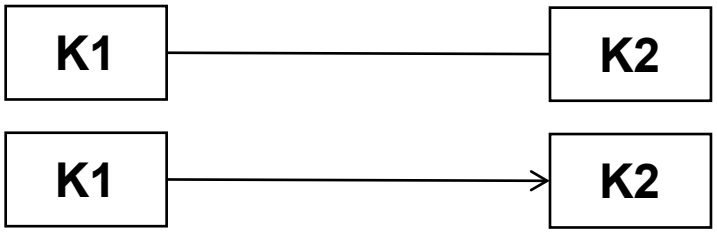
- Dazu muss man in der Implementierung eine Leseoperation zur Verfügung stellen

- Man unterscheide Assoziationen wie folgt:
  - Unidirektionale Assoziation: nur in einer Richtung navigierbar
  - Bidirektionale Assoziation: in beiden Richtungen navigierbar

Navigierbarkeit in UML1 und UML2
Klassendiagramme im strukturellen Entwurfsmodell

UML1:

- Es gibt genau zwei Fälle der Navigierbarkeit:

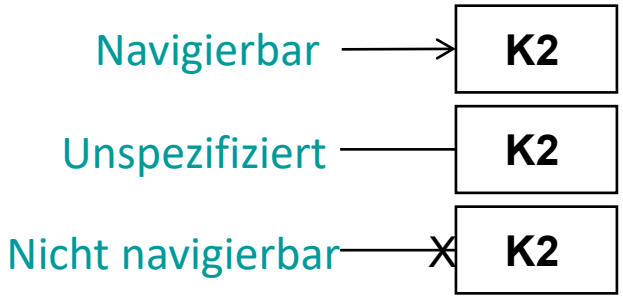


Bidirektionale Assoziation

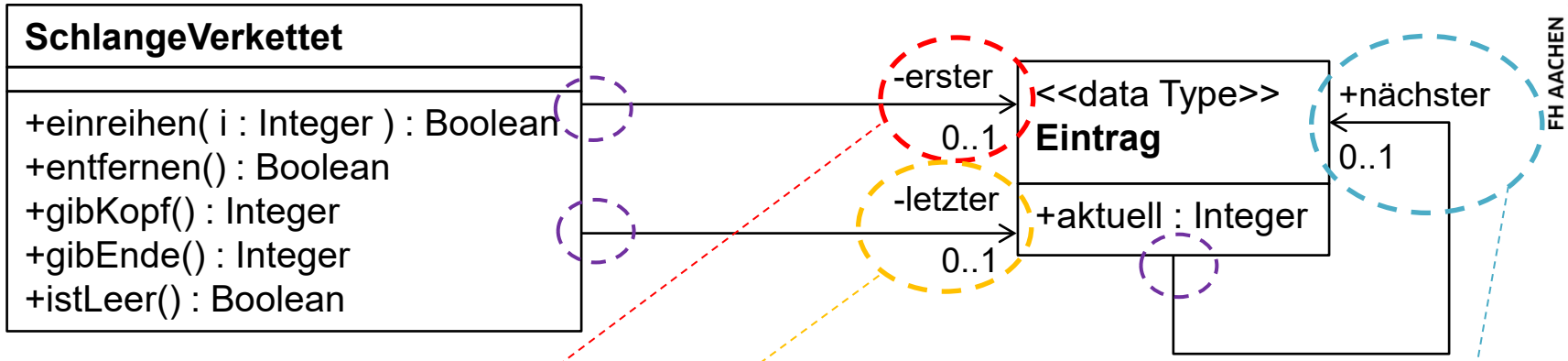
Unidirektionale Assoziation
(von K1 zu K2)

UML2:

- Jedes Assoziationsende hat einen von drei möglichen Navigationszuständen:



Implementierung Navigierbarkeit
Klassendiagramme im strukturellen Entwurfsmodell




```
public class SchlangeVerkettet {
   /* navigable 0..1 */
   private Eintrag erster;
   /* navigable 0..1 */
   private Eintrag letzter;
    public boolean einreihen(int i) {
        return false;
    }
    public boolean entfernen() {
        return false;
    }
}
```

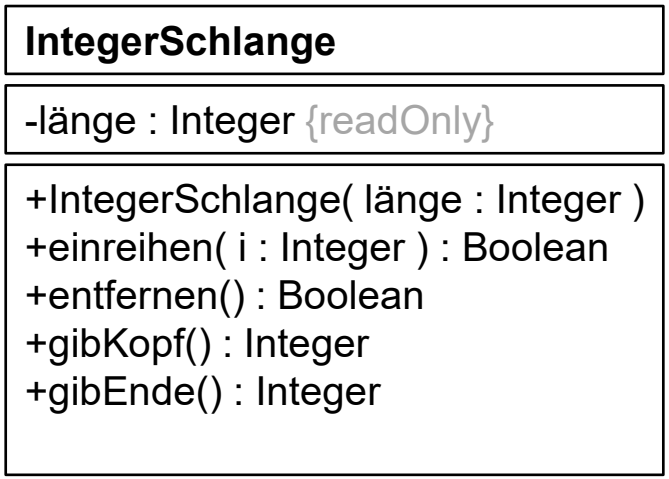
```
public int gibKopf() {
    return -1;
}
public int gibEnde(int i) {
    return -1;
}
public boolean istLeer() {
    return false;
}
```

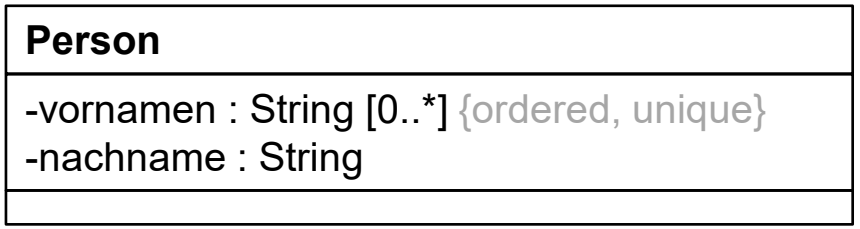
```
public class Eintrag {
   /* navigable 0..1 */
   public Eintrag nächster;
    public int aktuell;
}
```

Die unspezifizierte Gegenrichtung wird als "nicht navigierbar" interpretiert.

Eigenschaften von Attributen und Assoziationsenden
Klassendiagramme im strukturellen Entwurfsmodell

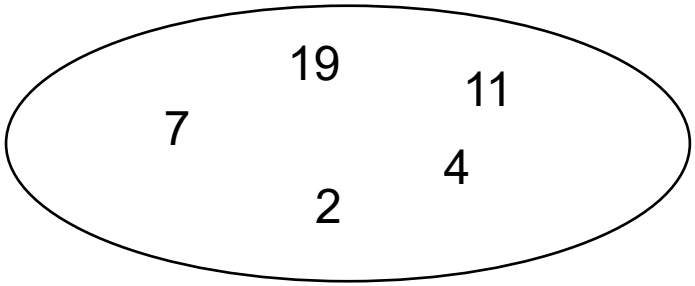
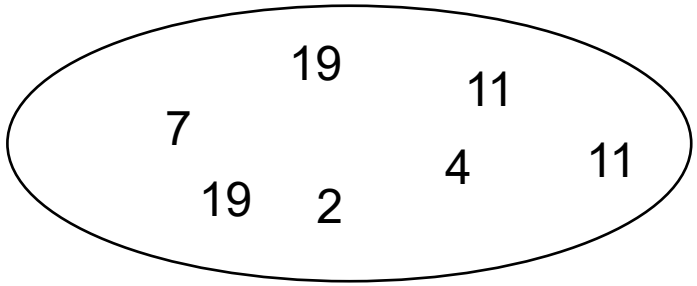
Eigenschaft	*Default*	*Bedeutung*
ordered	-	Geordnete Kollektion (Reihenfolge, keine Sortierung)
unordered	+	Ungeordnete Kollektion
unique	+	Eindeutige Kollektion (Element höchstens einmal enthalten)
nonunique	-	Mehrdeutige Kollektion
readonly	-	Wert darf nach Initialisierung nicht mehr verändert werden





Kollektionen im Klassendiagramm

Klassendiagramme im strukturellen Entwurfsmodell

	unique	nonunique
unordered	Menge 	Bag 
ordered	<7, 2, 11,19, 4> Geordnete Menge	<19, 7, 2, 11, 11,19, 4> Sequenz
	unique	nonunique

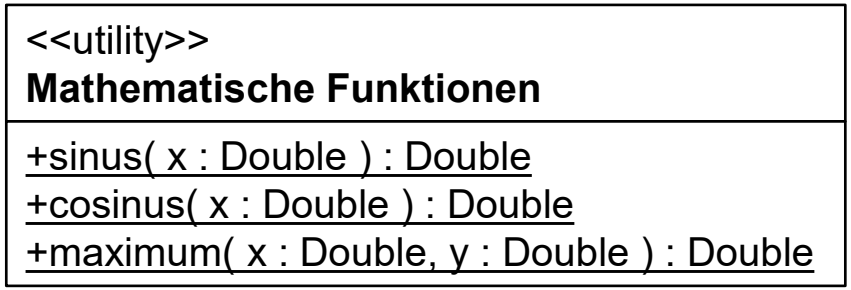
Hilfsmittelklassen
Klassendiagramme im strukturellen Entwurfsmodell

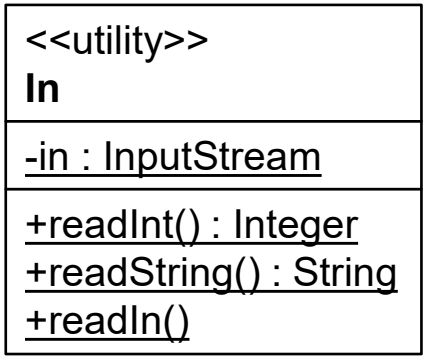
Hilfsmittelklassen dienen zur Gruppierung statischer (struktureller und Verhaltens-) Eigenschaften

- Sie sind (wie abstrakte Klassen) nicht instantiierbar
- Mit statischen Attributen lassen sich globale Variablen darstellen
- Statische Operationen stellen Sammlungen von Funktionen bzw. Prozeduren dar

Spezialfälle

- Operationsklassen (ohne Gedächtnis)
- Datenobjektklassen (ein einziges, nicht instantiierbares Datenobjekt)



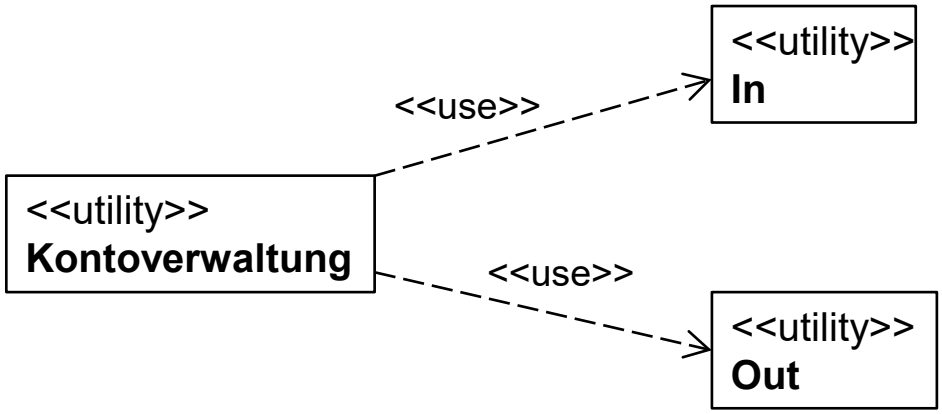


Benutzungsabhängigkeiten
Klassendiagramme im strukturellen Entwurfsmodell

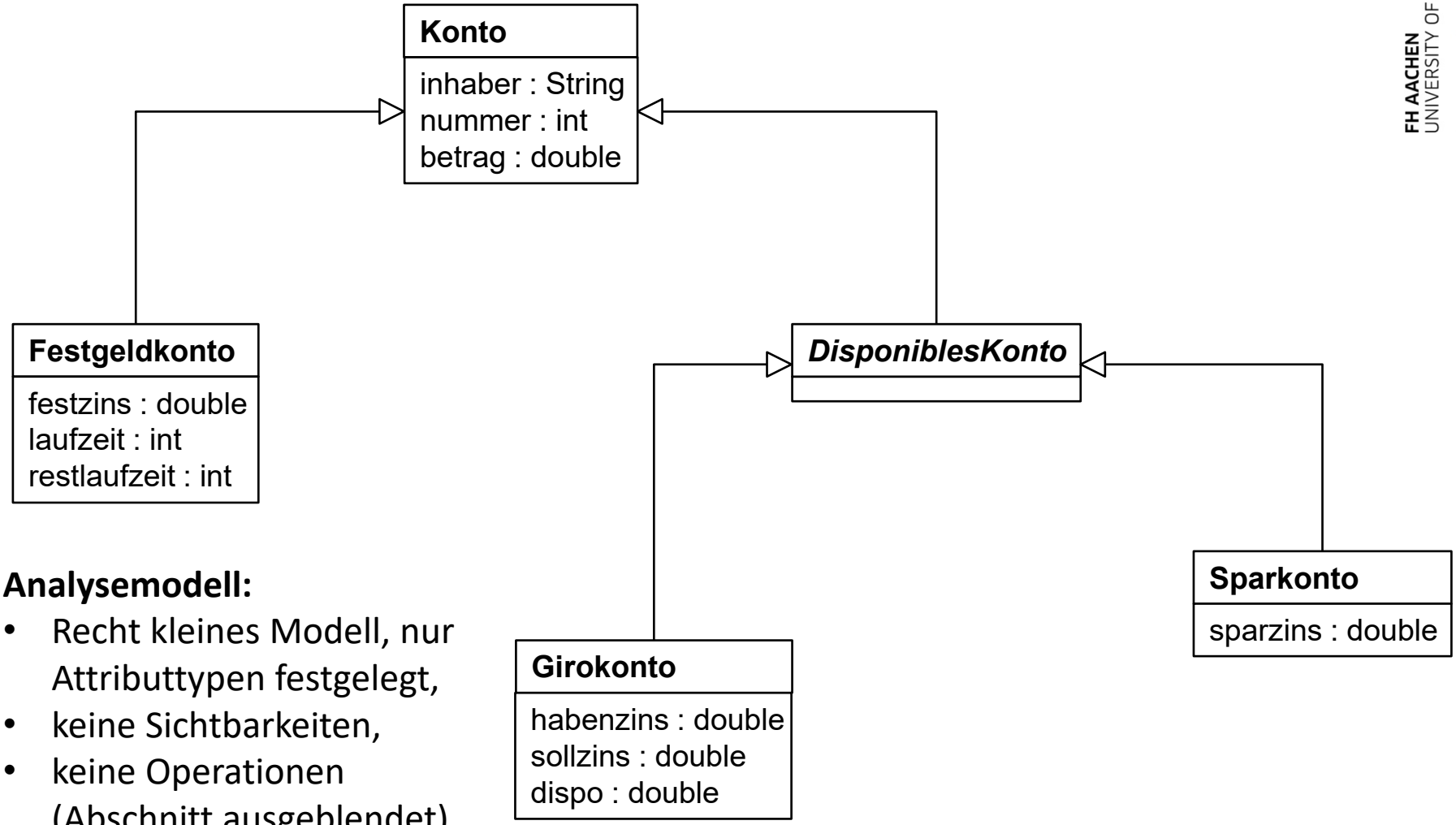
Assoziationen und Vererbungsbeziehungen beschrieben Abhängigkeiten zwischen Klassen nur unvollständig

Eine Quellklasse ist von der Zielklasse benutzungsabhängig, wenn sie Elemente der Zielklasse zu ihrer Realisierung benutzt

Eingetragen werden diese nur, wenn die Benutzung nicht ohnehin aus einer Assoziation oder einer Vererbungsbeziehung hervorgeht



Schritt 1: Analysemodell
Vom Analysemodell zum Entwurfsmodell

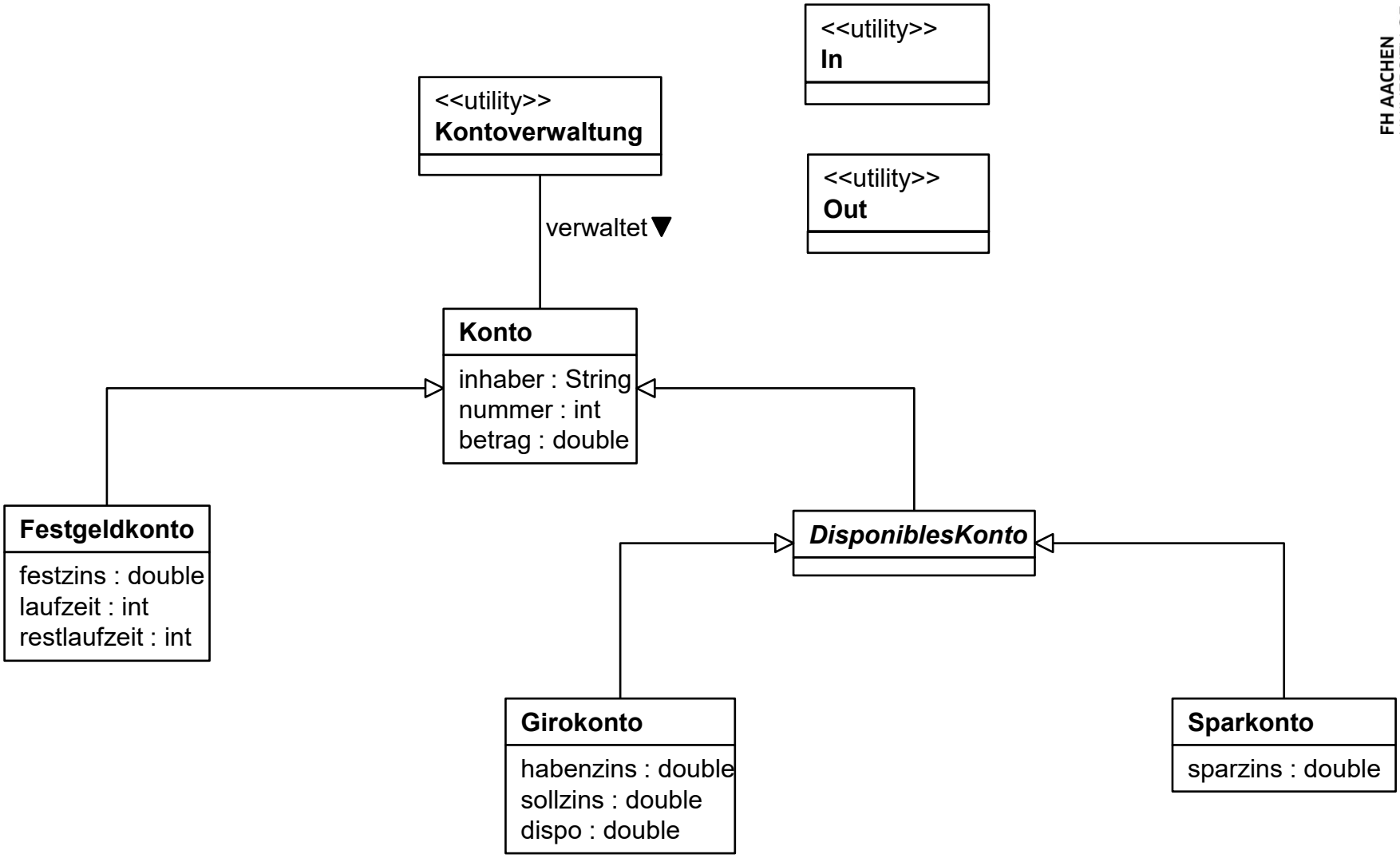


Analysemodell:

- Recht kleines Modell, nur Attributtypen festgelegt,
- keine Sichtbarkeiten,
- keine Operationen (Abschnitt ausgeblendet)

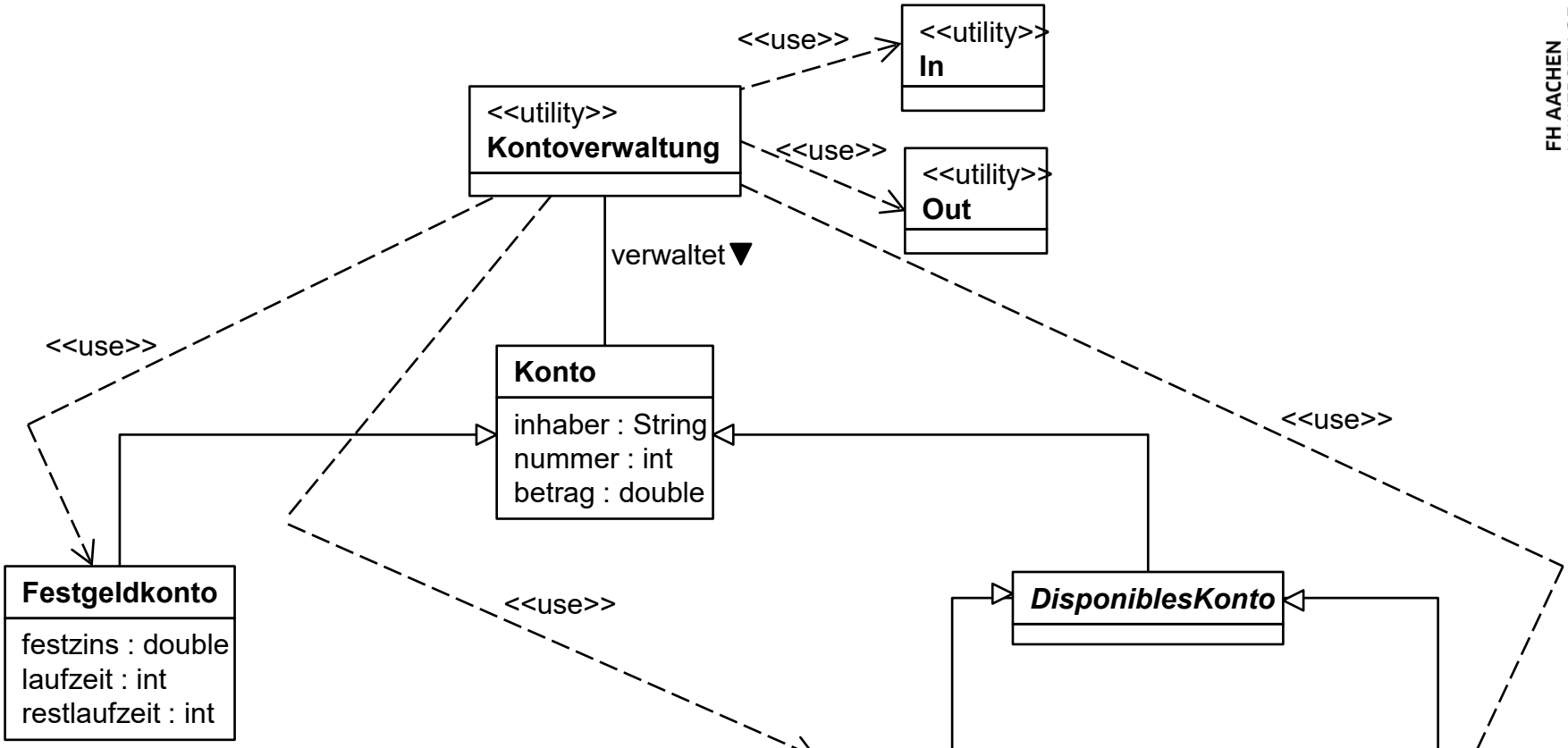
Schritt 2: Kontrolle und Ein-/Ausgabe hinzufügen

Vom Analysemodell zum Entwurfsmodell



Schritt 3: Benutzungsabhängigkeiten hinzufügen

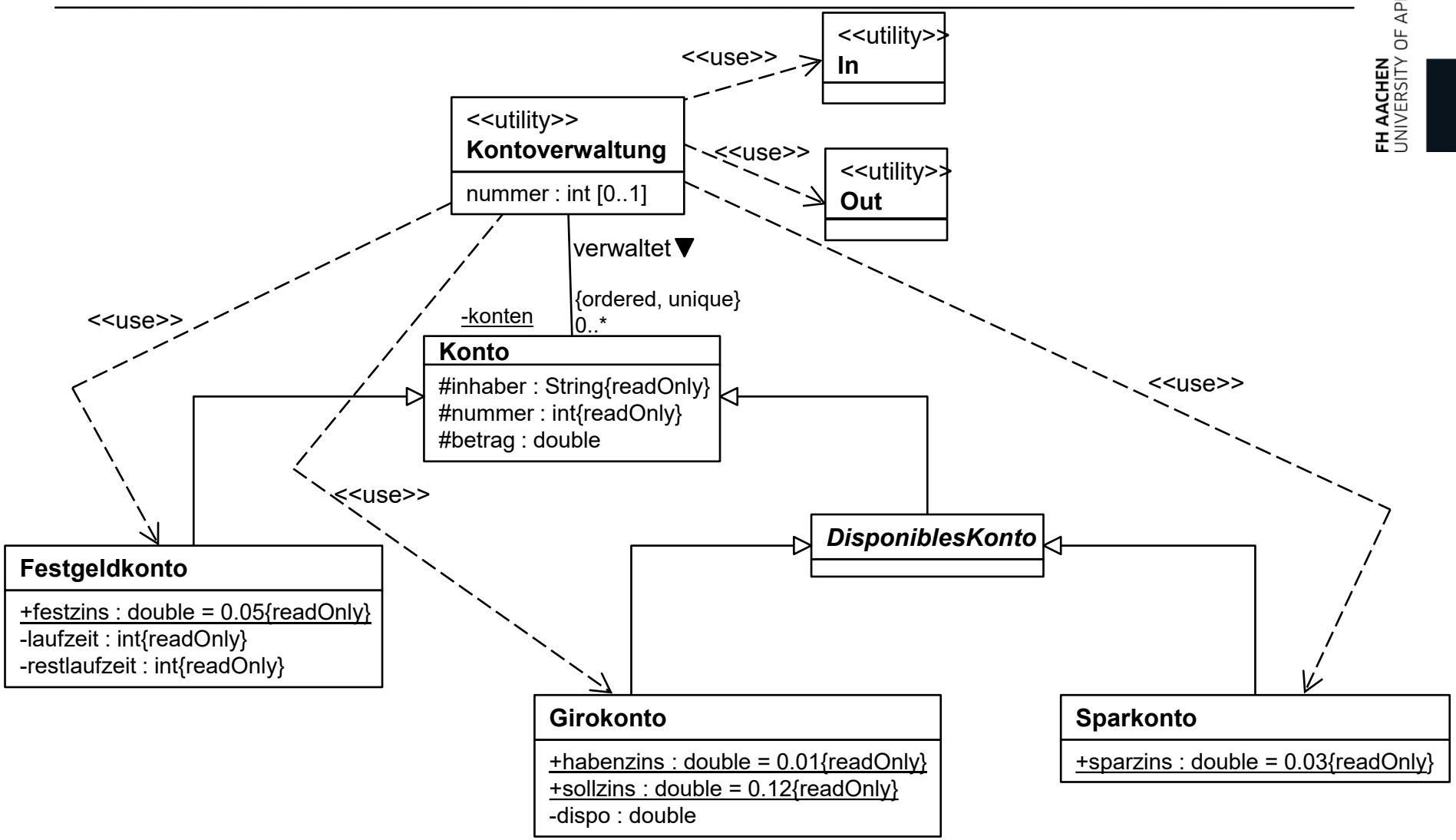
Vom Analysemodell zum Entwurfsmodell



Kontoklassen werden bei Objekterzeugung benutzt, In und Out zur Ein-/Ausgabe DisponiblesKonto wird nicht verwendet (nur die erbenden Klassen)

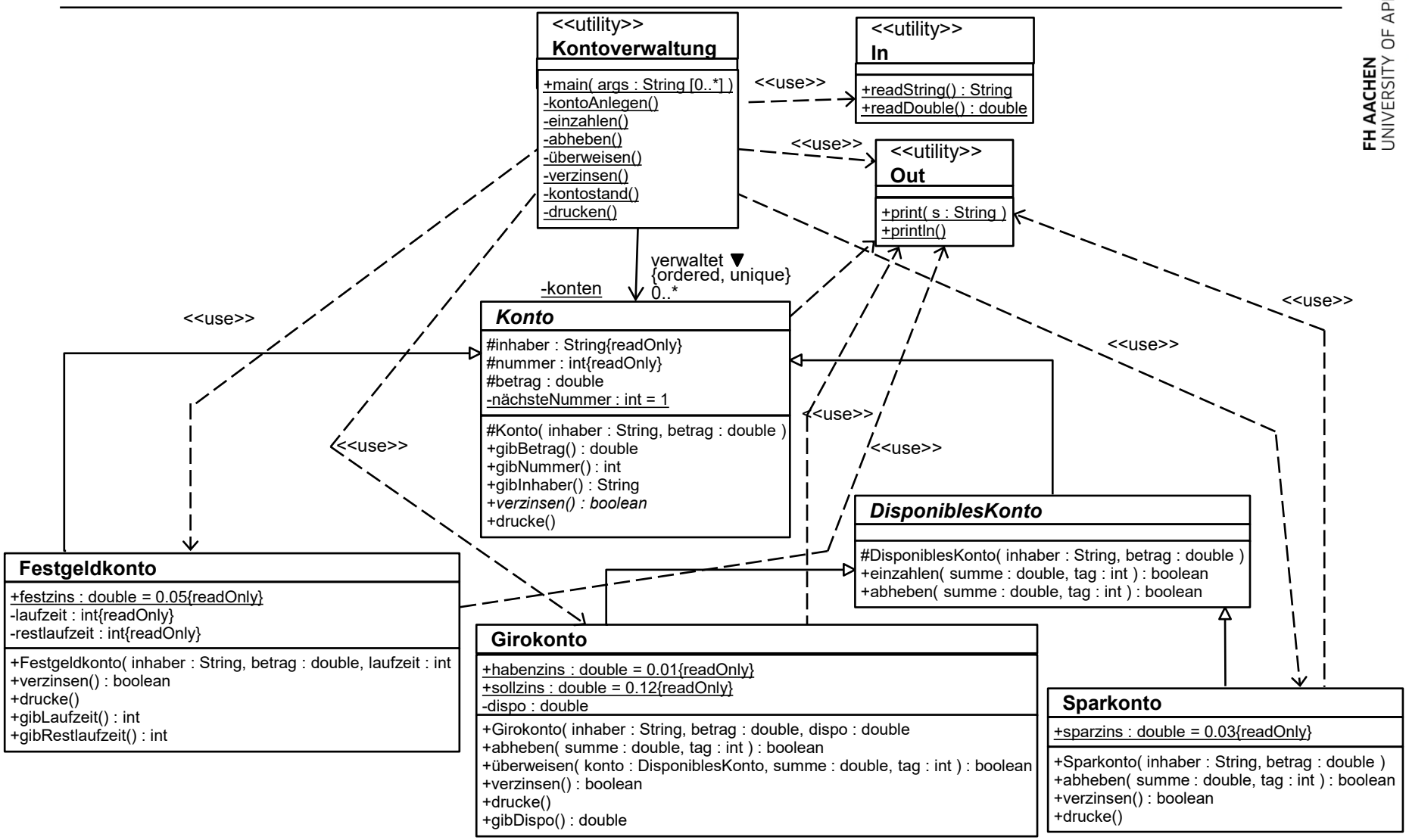
Schritt 4: Detaillierung von Attributen und Assozationen

Vom Analysemodell zum Entwurfsmodell



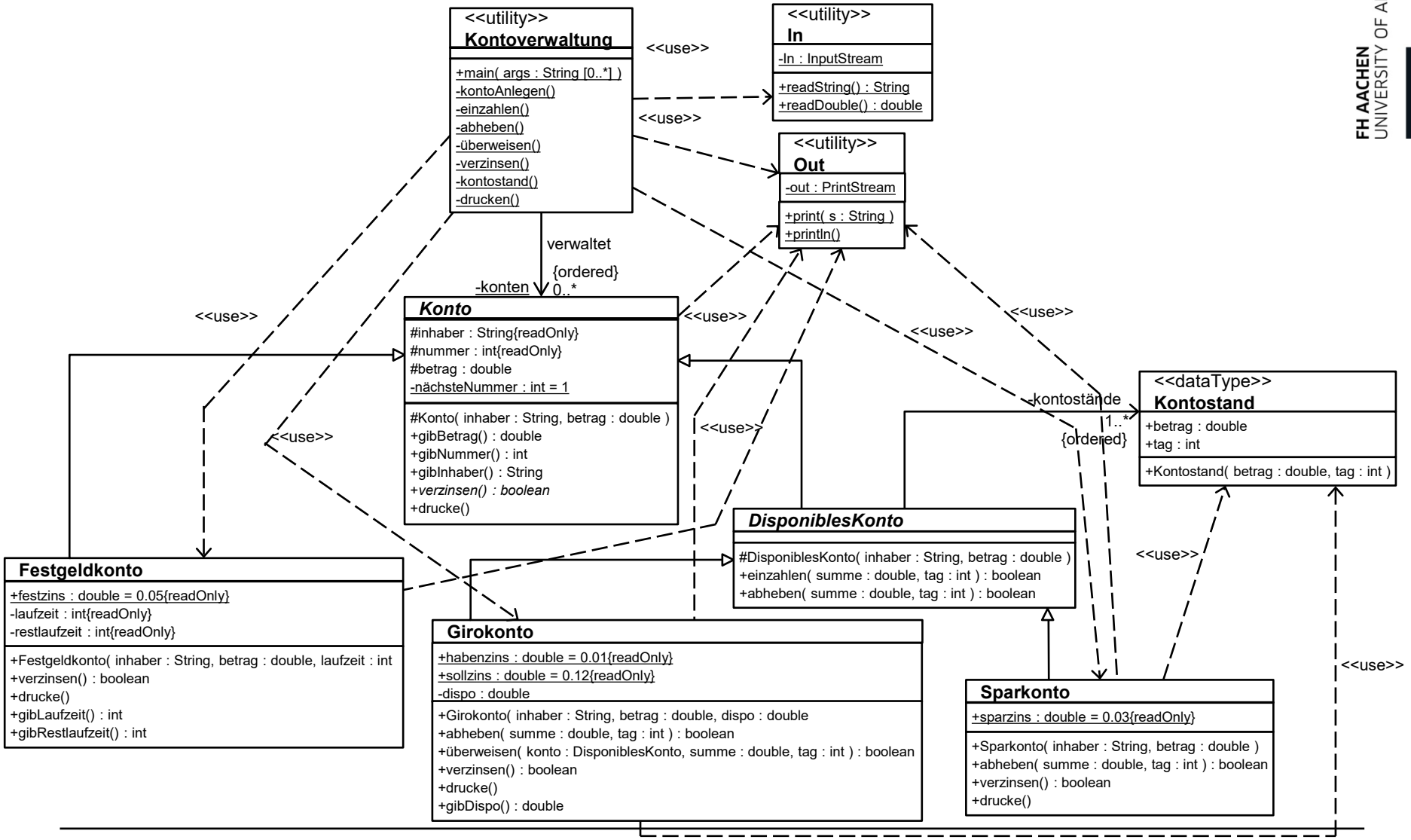
Schritt 5: Spezifikation von Operationen

Vom Analysemodell zum Entwurfsmodell



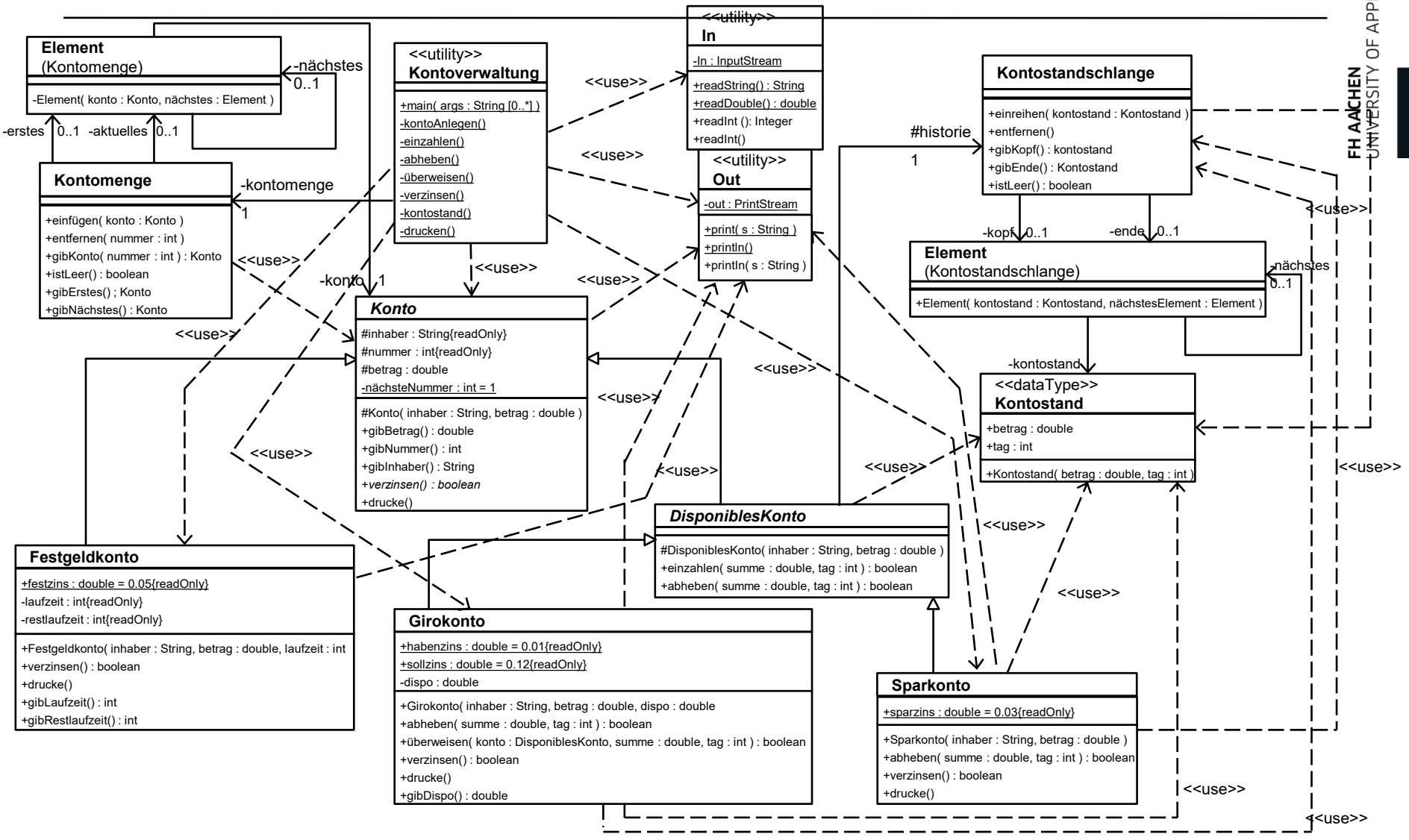
Schritt 6: Weitere Realisierungsdetails

Vom Analysemodell zum Entwurfsmodell



Schritt 7: Realisierung von Kollektionen

Vom Analysemodell zum Entwurfsmodell

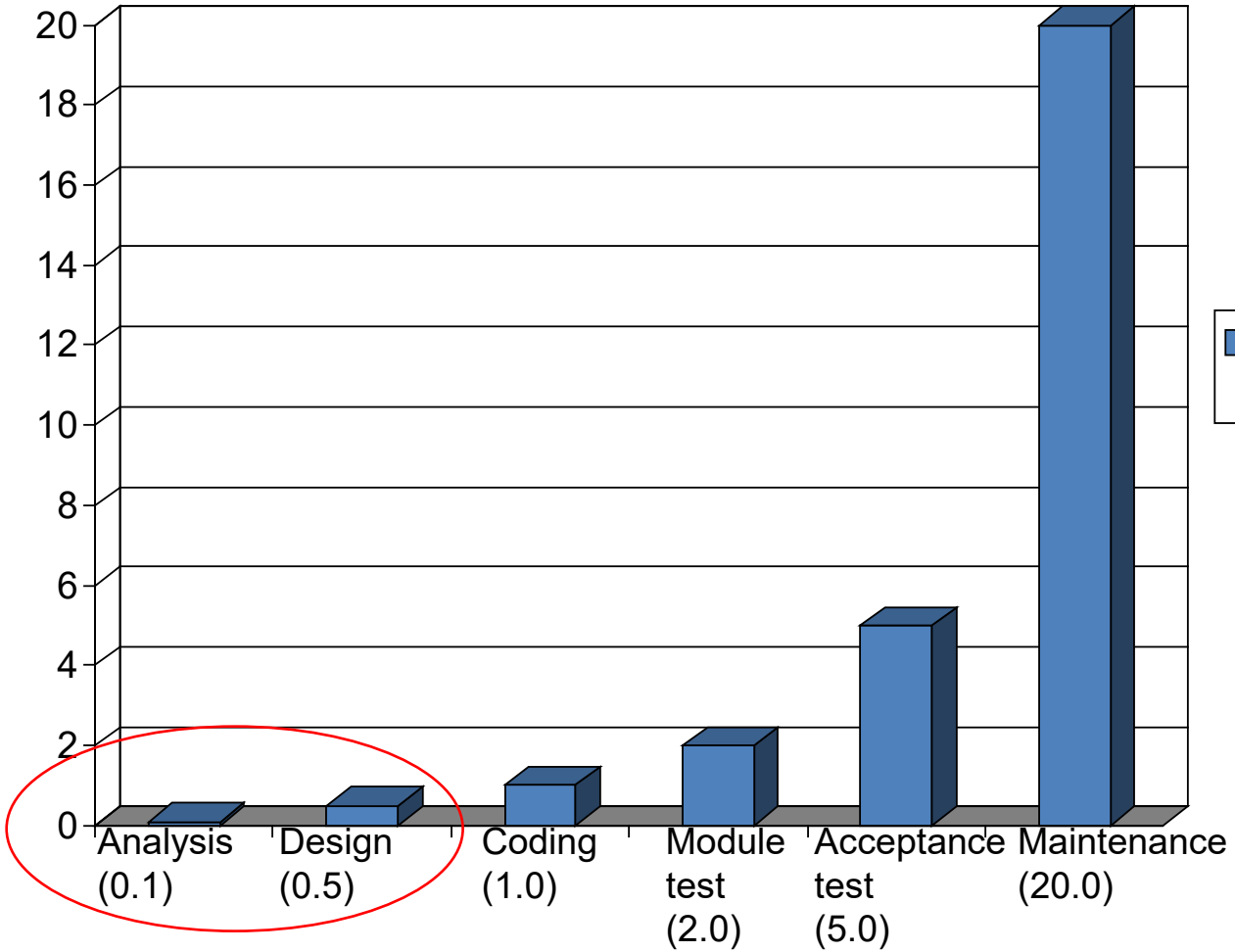


Fazit
Vom Analysemodell zum Entwurfsmodell

- Ausgehend von einem kompakten Analysemodell wird schrittweise ein detailliertes Entwurfsmodell abgeleitet
- Dabei werden zusätzliche Aspekte berücksichtigt (Ein- und Ausgabe, Kontrolle), die in der Anforderungsanalyse noch keine Rolle spielen
- Es entsteht ein Entwurfsmodell (Schritt 7), das 1:1 in die Implementierung (z.B. in Java) umgesetzt werden kann

- Die Implementierung von Attributen und Assoziationen sollte idealerweise automatisch erfolgen (durch einen Codegenerator)
- Allerdings reichen dazu in diesem Beispiel die Sprachmittel der UML nicht vollständig aus

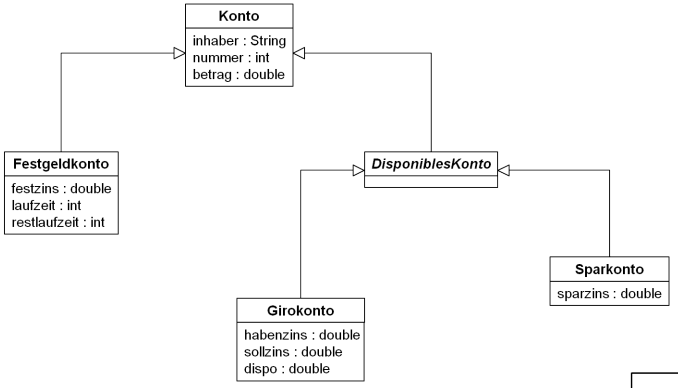
Erinnerung: Kosten der Fehlerbeseitigung
Klassendiagramme im strukturellen Entwurfsmodell



Bemerkung:
Ein in Phase X entfernter Fehler kann in Phase X oder in einer früheren Phase verursacht worden sein.

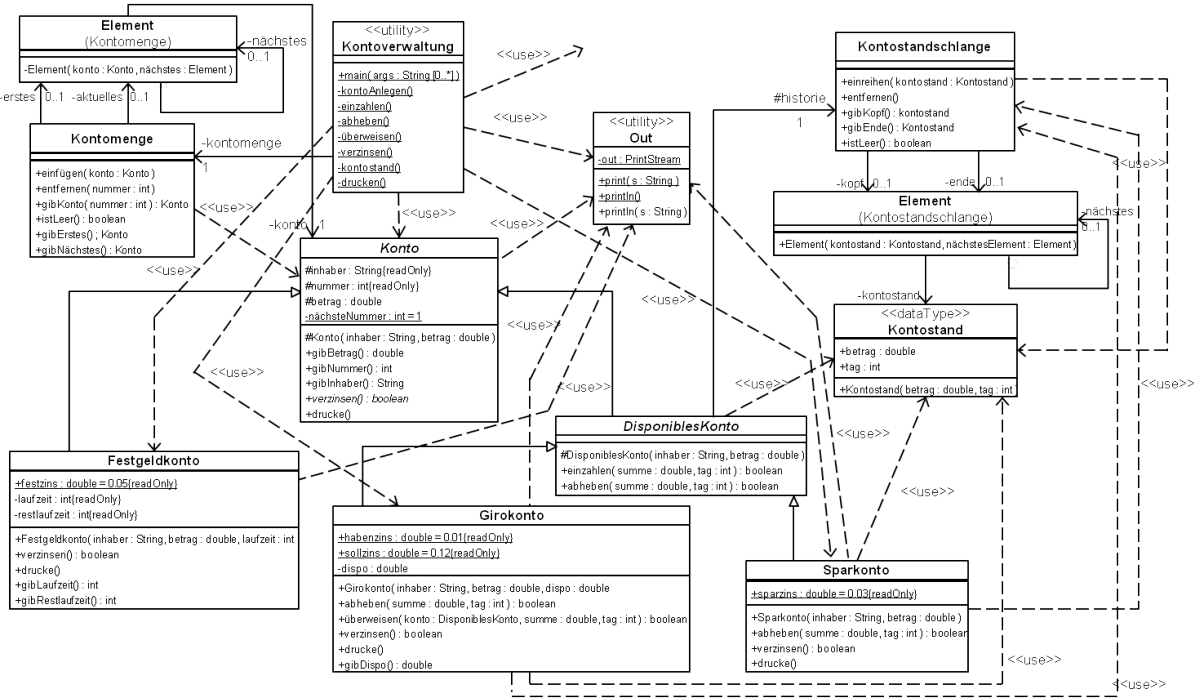
[Well 2000]

Änderungsaufwand in Analyse und Entwurf
Klassendiagramme im strukturellen Entwurfsmodell



Analysemodell:
niedriger Aufwand

Entwurfsmodell:
hoher Aufwand



Agenda und Quellen
Anforderungsanalyse mit UML

Motivation und Einordnung

Klassendiagramme im strukturellen Entwurfsmodell

Vom Analysemodell zum Entwurfsmodell

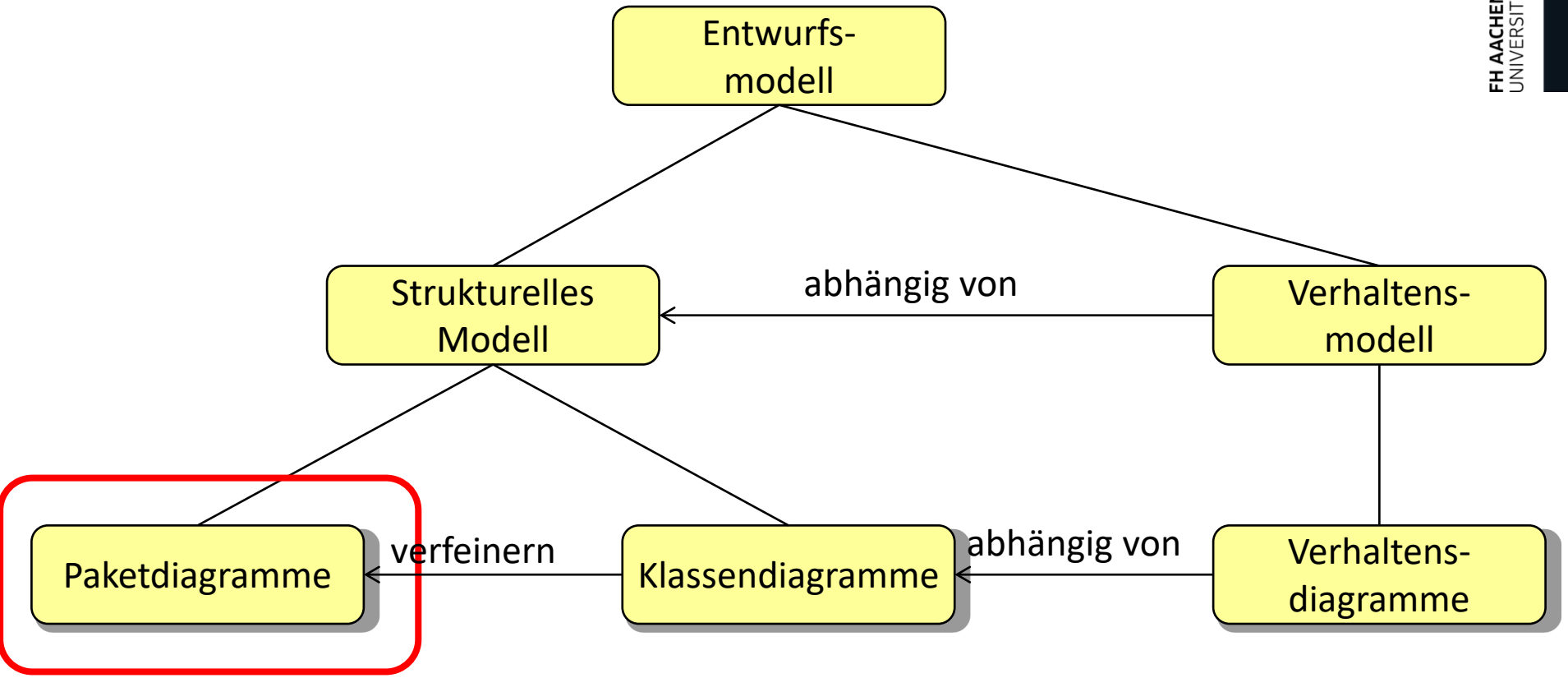
Pakete im strukturellen Entwurfsmodell

Verhaltensdiagramm im Entwurfsmodell

Quellen

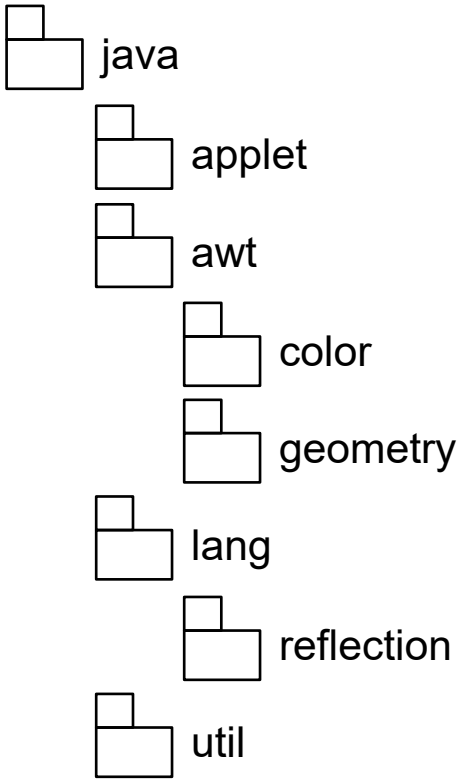
Vorlesung von
Prof. Westfechtel
Uni Beireuth

Aufbau des Entwurfsmodells
Motivation und Einordnung



Pakete im strukturellen Entwurfsmodell
Einordnung und Motivation

Pakethierarchie der Java-Klassenbibliothek (Ausschnitt)

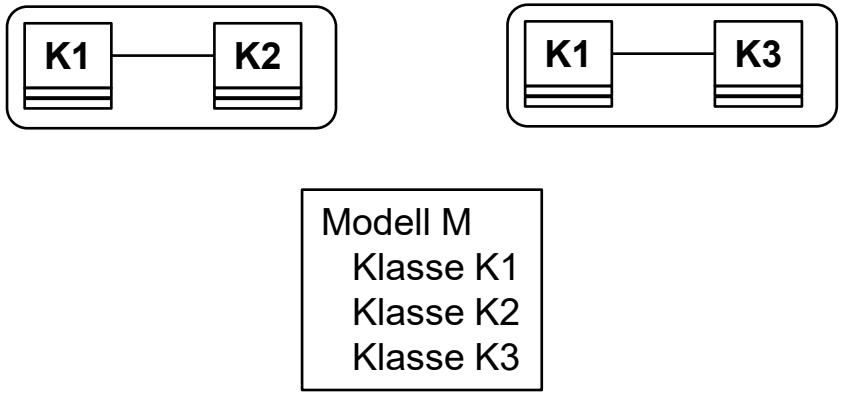


In der UML dienen Pakete zur Strukturierung großer Modelle

Pakete lassen sich hierarchisch schachteln

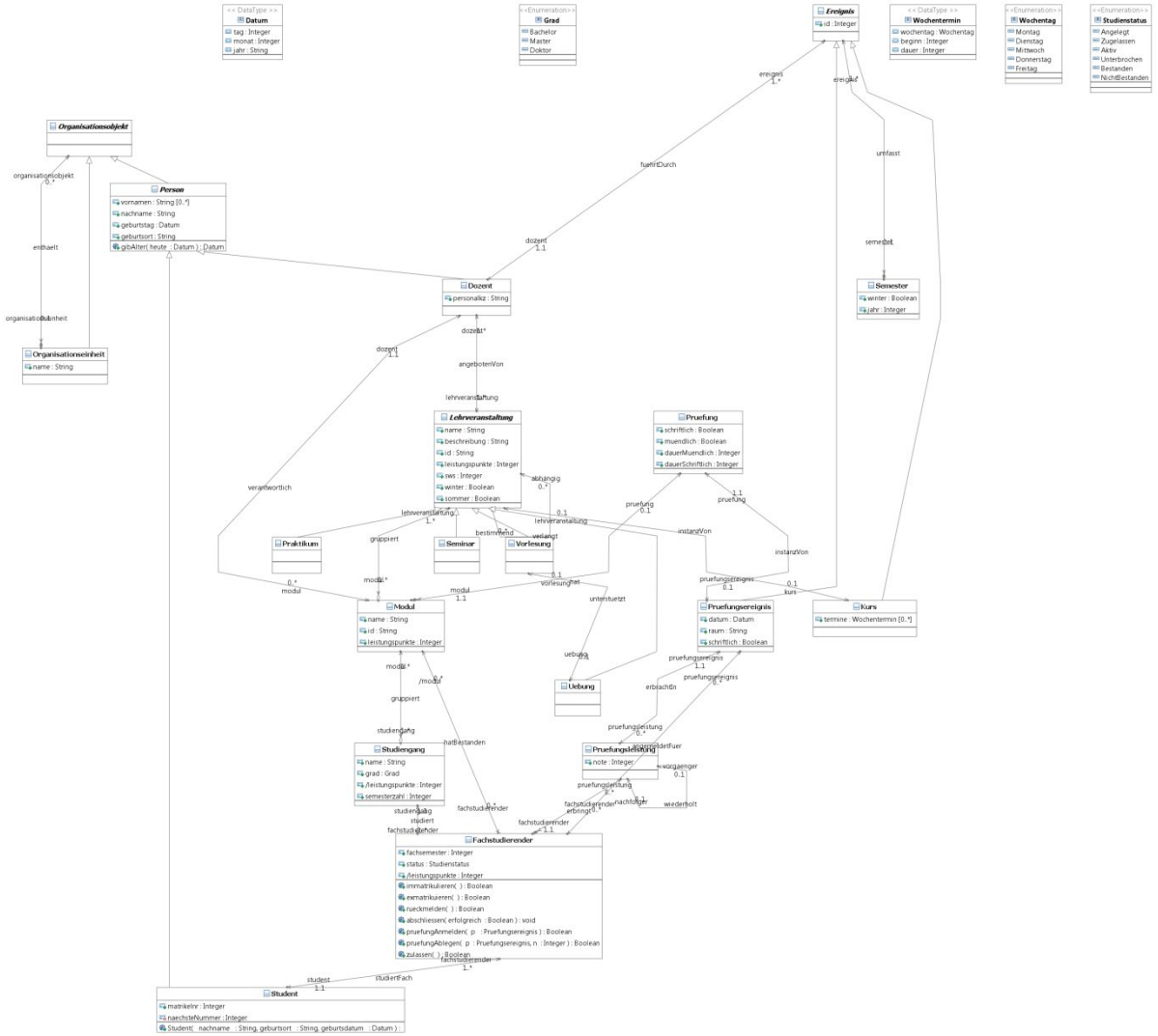
Jedes Modellelement gehört zu höchstens einem Paket

Modellelemente können aber in mehreren Diagrammen auftreten

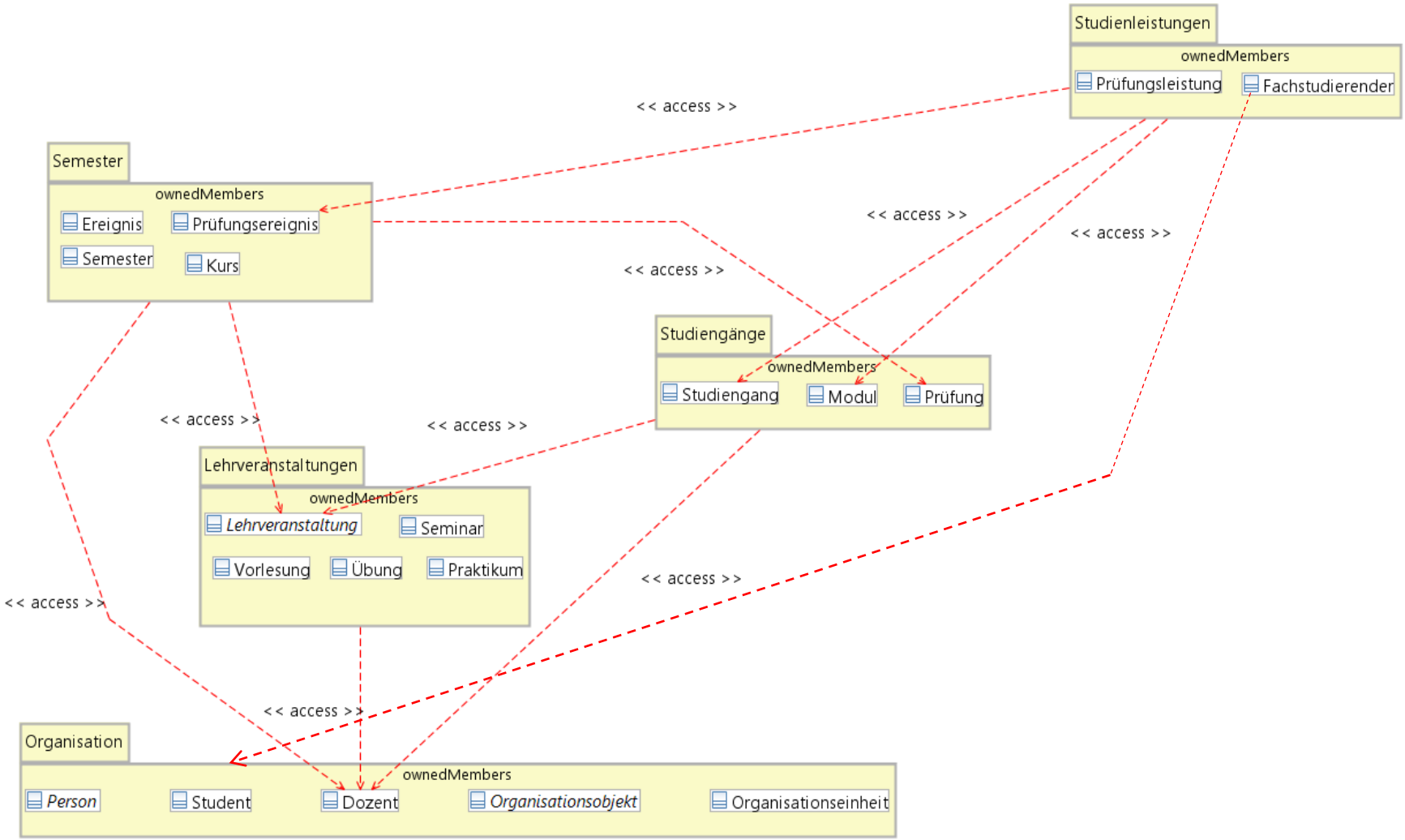


Beispiel: Campus-Management-System ohne Pakete

Einordnung und Motivation



Beispiel: Campus-Management-System mit Paketen
Einordnung und Motivation



Architekturmodellierung mit Paketdiagrammen
Einordnung und Motivation

Mit Hilfe von Paketen lässt sich ein großes Entwurfsmodell

- in handhabbare Einheiten modular zerlegen
- durch Einschränkung von Sichtbarkeiten robuster und wartbarer machen

Kriterien zum Entwurf von Paketen

- Ein Paket ist eine Entwurfseinheit angemessener Größe
  - Große Pakete hierarchisch zerlegen
- Ein Paket ist eine logisch abgeschlossene Einheit
- Prinzip der losen Kopplung
  - Zwischen den Paketen der Gesamtarchitektur bestehen jeweils nur schwache Abhängigkeiten
  - Auswirkungen von Änderungen werden damit begrenzt
- Prinzip der hohen Kohäsion
  - Die Klassen eines Pakets sind vergleichsweise eng miteinander verbunden
  - Sonst wäre ihre Gruppierung in einem Paket nicht sinnvoll

Prinzip der hohen Kohäsion
Einordnung und Motivation

Die Kräfte, die ein Modul im Inneren zusammenhalten

Starke Kohäsion bedeutet, dass die Teile des Moduls/Pakets eng zueinander gehören.

Beispiel starke (funktionale) Kohäsion in einem Modul:

- `setzeNamen(Person, Name);`
- `String leseNamen(Person);`

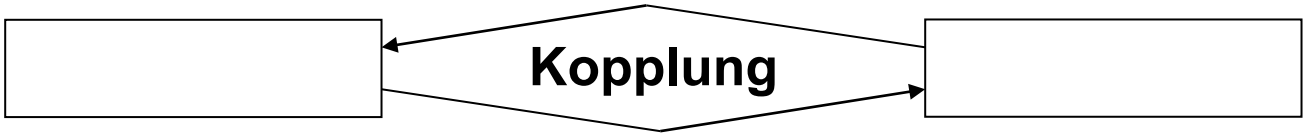
Beispiel schwache (zufällige) Kohäsion in einem Modul:

- `setzePANr(Person, int);`
- `druckeInventarliste();`

Prinzip der losen Koppelung
Einordnung und Motivation

Die **Bindungen**, die ein Modul **mit anderen Modulen** eingeht

Lose Koppelung bedeutet, dass die anderen Module möglichst wenig über ein Modul wissen (z.B. die interne Datenstruktur)



Beispiel lose (Daten-) Koppelung:

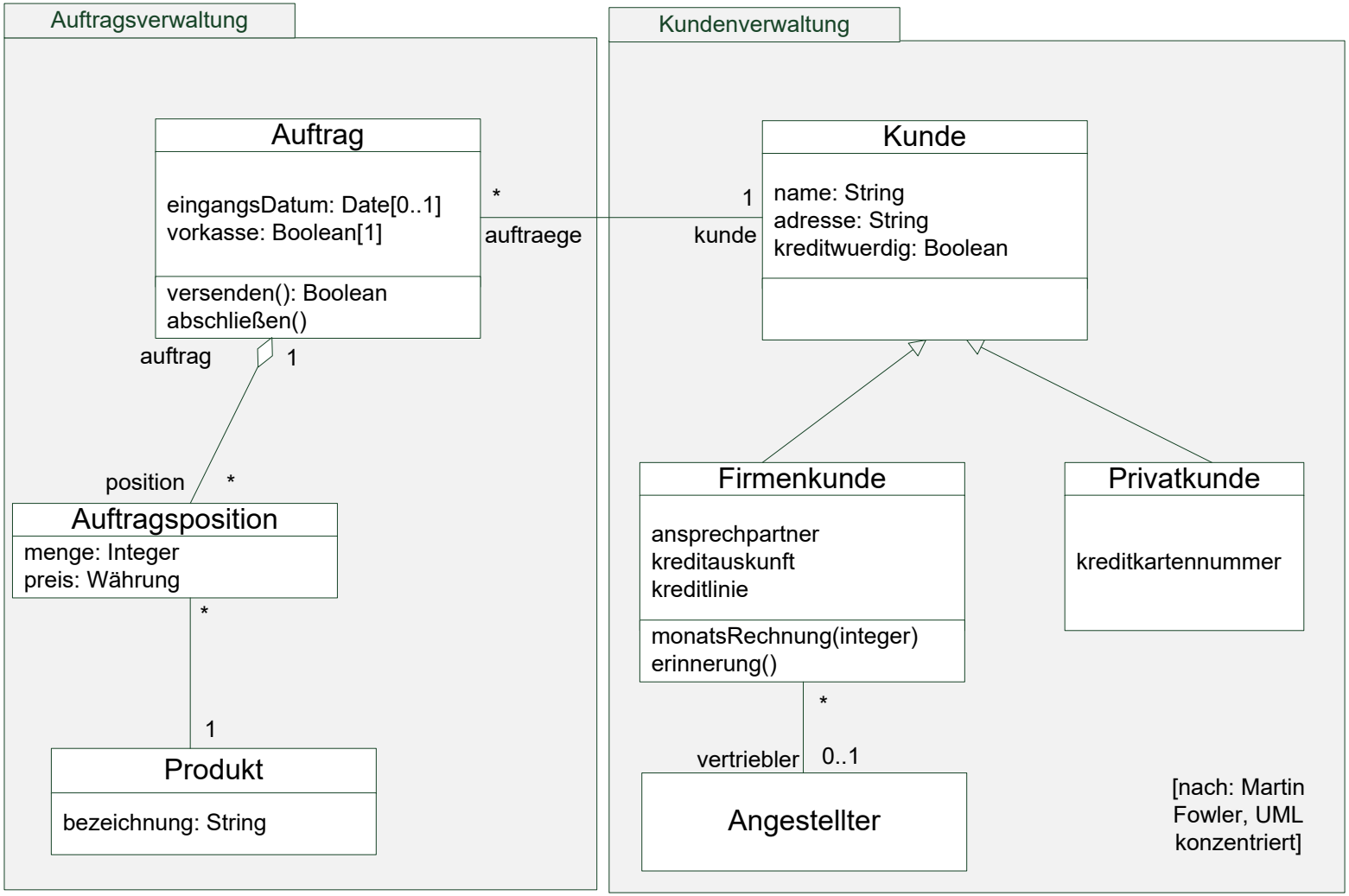
Klasse Rechteck mit `setzeLinks(ord)`, `setzeRechts(ord)`, `setzeBreite(laenge)`

→ andere Module wissen nicht, welche Daten gespeichert und welche berechnet werden → Information Hiding

Beispiel starke (inhaltliche) Koppelung: Verzeigerte Liste, deren Zeiger bei jeder Verwendung umgebogen werden

```
neuerArtikel.next = aktuellerArtikel.next;
aktuellerArtikel.next = neuerArtikel;
```

Beispielsystem mit Paketen
Einordnung und Motivation



Literatur

[Alf 2010] *Action Language for Foundational UML (Alf) - Concrete Syntax for a UML Action Language*, Version FTF – Beta 1, Object Management Group, Dokument ptc/2010-10-05 (2010)

[Booch 1999] G. Booch, J. Rumbaugh, I. Jacobson: *The Unified Modeling Language User Guide*, Addison-Wesley Verlag, Object Technology Series (1999)

[Buchmann 2011] T. Buchmann, A. Dotor, B. Westfechtel: *Model-Driven Software Engineering: Concepts and Tools for Modeling-in-the-Large with Package Diagrams,* Computer Science Research and Development, Online First (2011)

[Buchmann 2012] T. Buchmann: *Valkyrie: A UML-Based Model-Driven Environment for Model-Driven Software Engineering*, Proceedings 7th International Conference on Software Paradigm Trends, Rom, Insticc Press (2012)

[DeRemer 1976] F. DeRemer, H.H. Kron: *Programming in the Large versus Programming in the Small*, IEEE Transactions on Software Engineering, vol. 2, no. 2, 80-86 (1976)

[Dotor 2011] A. Dotor: *Entwurf und Modellierung einer Produktlinie von Softare-Konfigurations-Management-Systemen*, Dissertation, Universität Bayreuth, 2011

[Harel 1987] D. Harel: *Statecharts – A Visual Formalism for Complex Systems*, Science of Computer Programming, vol. 8, 231-274 (1987)

[Hitz 2005] M. Hitz, G. Kappel, E. Kapsammer, W. Retzischegger: *UML@Work. Objektorientierte Modellierung mit UML 2*, dpunkt Verlag (2005)

[Nagl 1990] M. Nagl: *Softwaretechnik – Methodisches Programmieren im Großen*, Springer-Verlag, Springer Compass (1990)

[Well 2000] D.L. Well, D. Widdrig: *Managing Software Requirements – A Unified Approach*, Addison-Wesley Verlag (2000)