

Kapitel 5

Anforderungserhebung und Analyse („Requirements Engineering“)

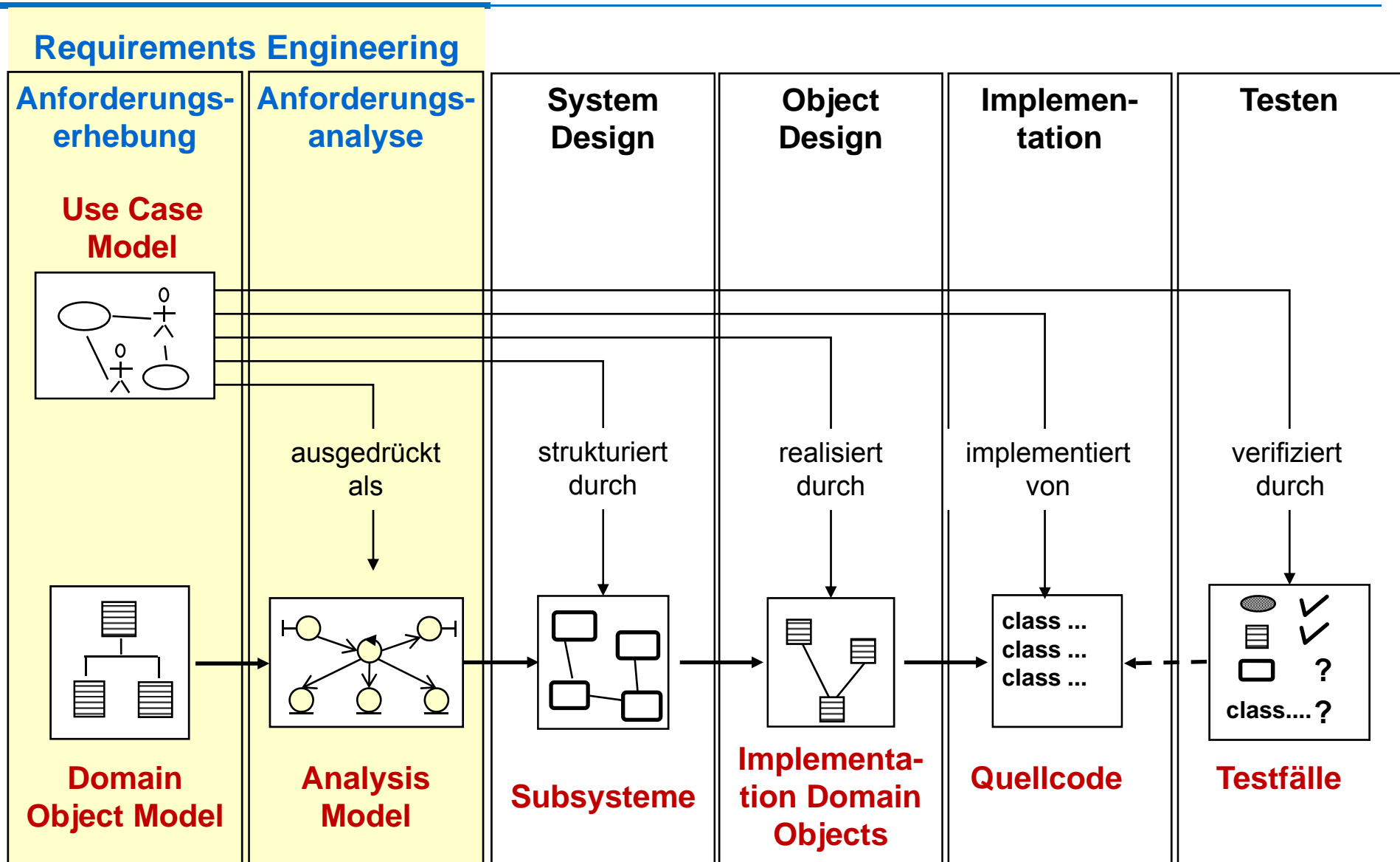
Stand: 11.11.2010

11.11.2010: Folien 134 – 139 überarbeitet

Kontext

- Bisher: Grundlagen
 - ◆ Konfigurationsmanagement und Werkzeuge (Eclipse + SVN)
 - ◆ UML Einführung und Werkzeuge (Eclipse + Paradigm)
- Ab jetzt: Arbeitsabläufe („Aktivitäten“ / „Workflows“) und dazugehörige Technologien, Werkzeuge, etc.
 - ◆ Anforderungserhebung
 - ◆ Systementwurf
 - ◆ Objektentwurf
 - ◆ Implementierung
 - ◆ Testen
 - ◆ Refactoring
- Später: Prozessmodelle
 - ◆ Organisation des Projektes
 - ◆ Wann, wie viel von welchem Workflow?

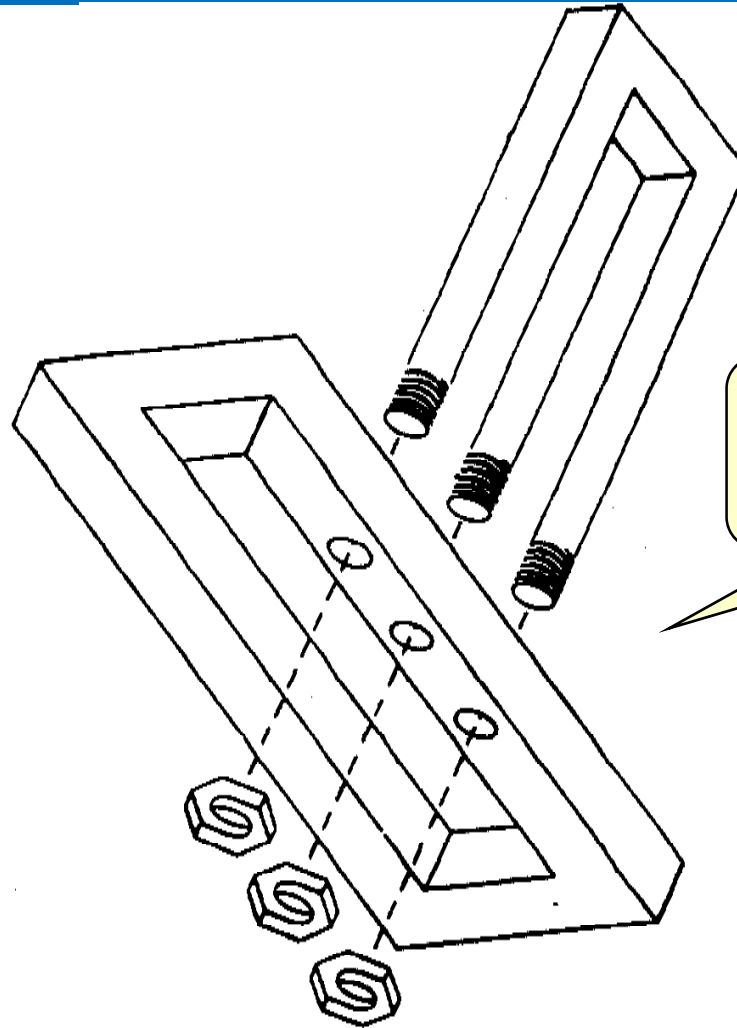
Aktivitäten bei der Softwareentwicklung („Workflows“)



Requirements Engineering – Was und warum? –

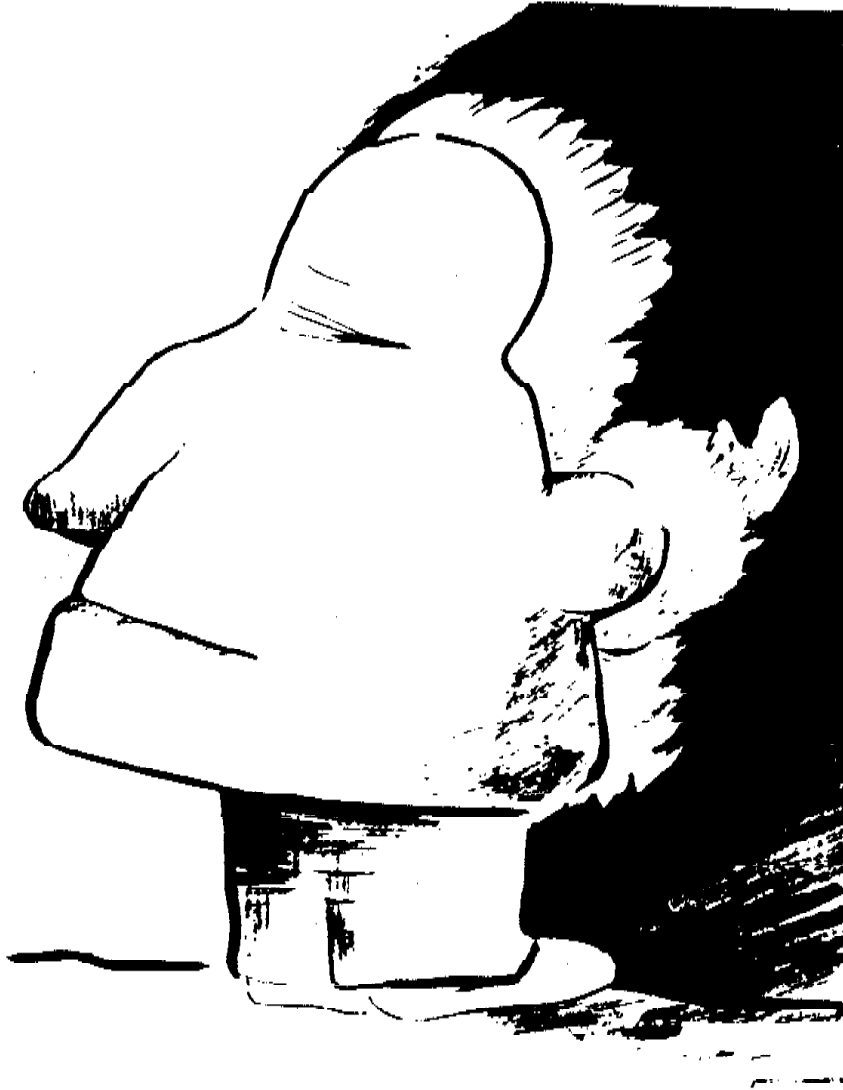
Motivation und Definition

Können Sie so etwas bauen?



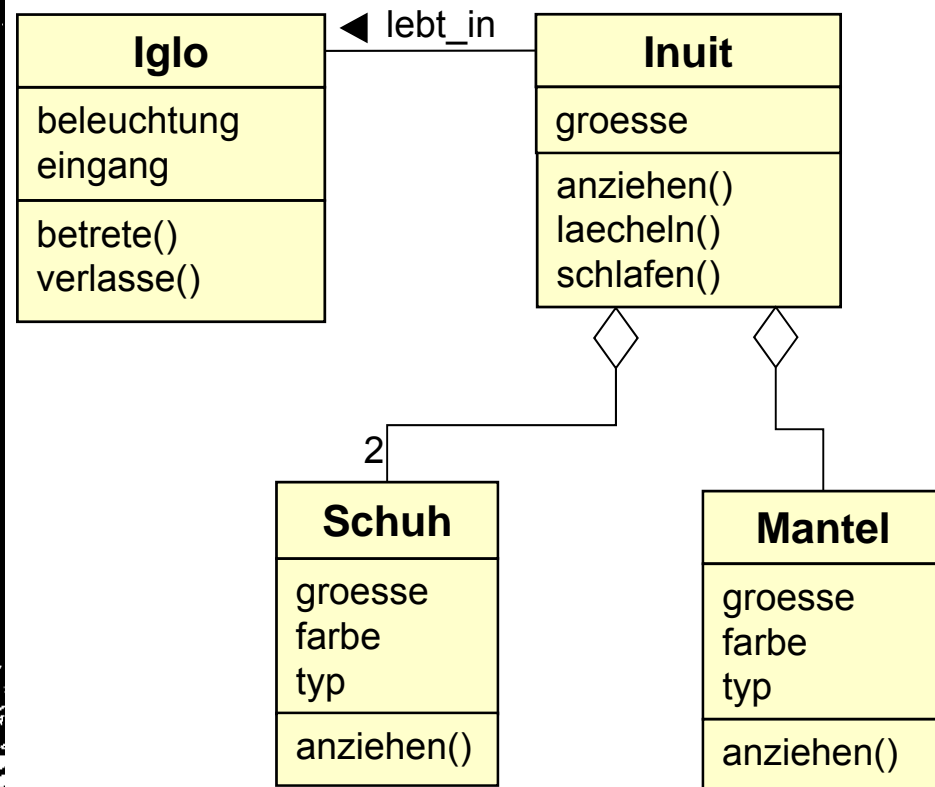
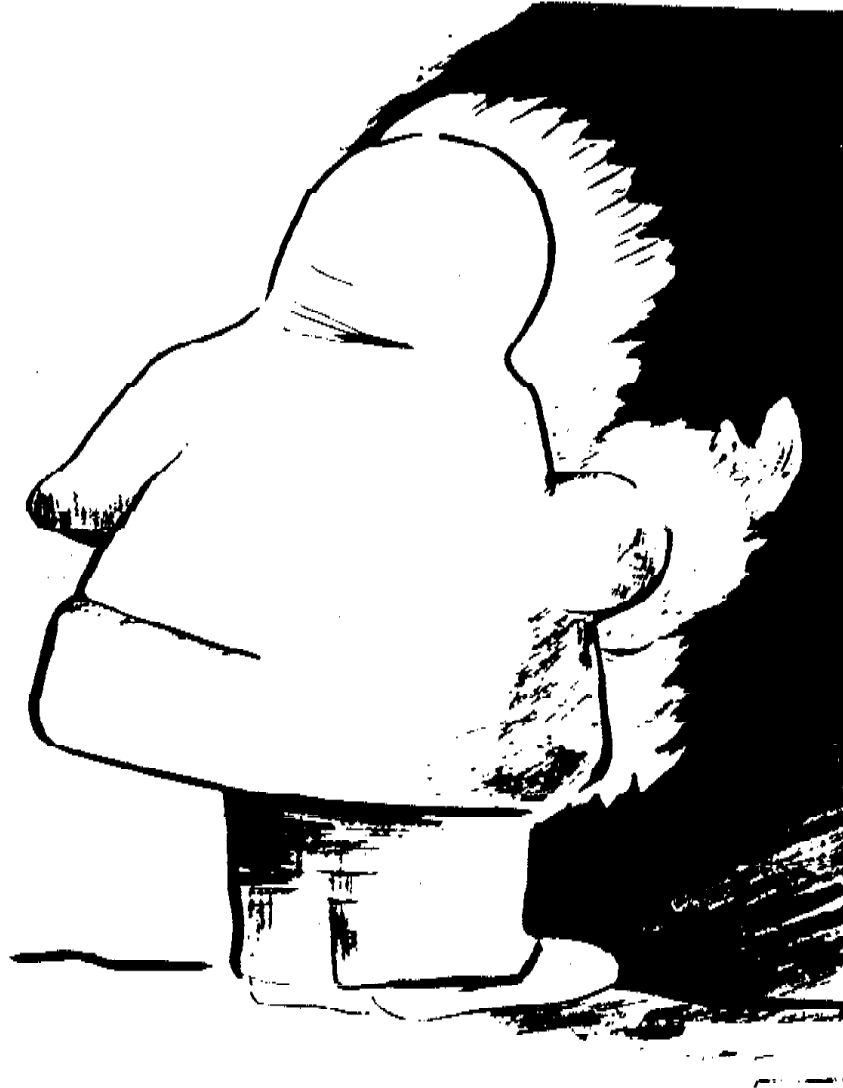
Widersprüchliche Anforderungen

Was ist das?

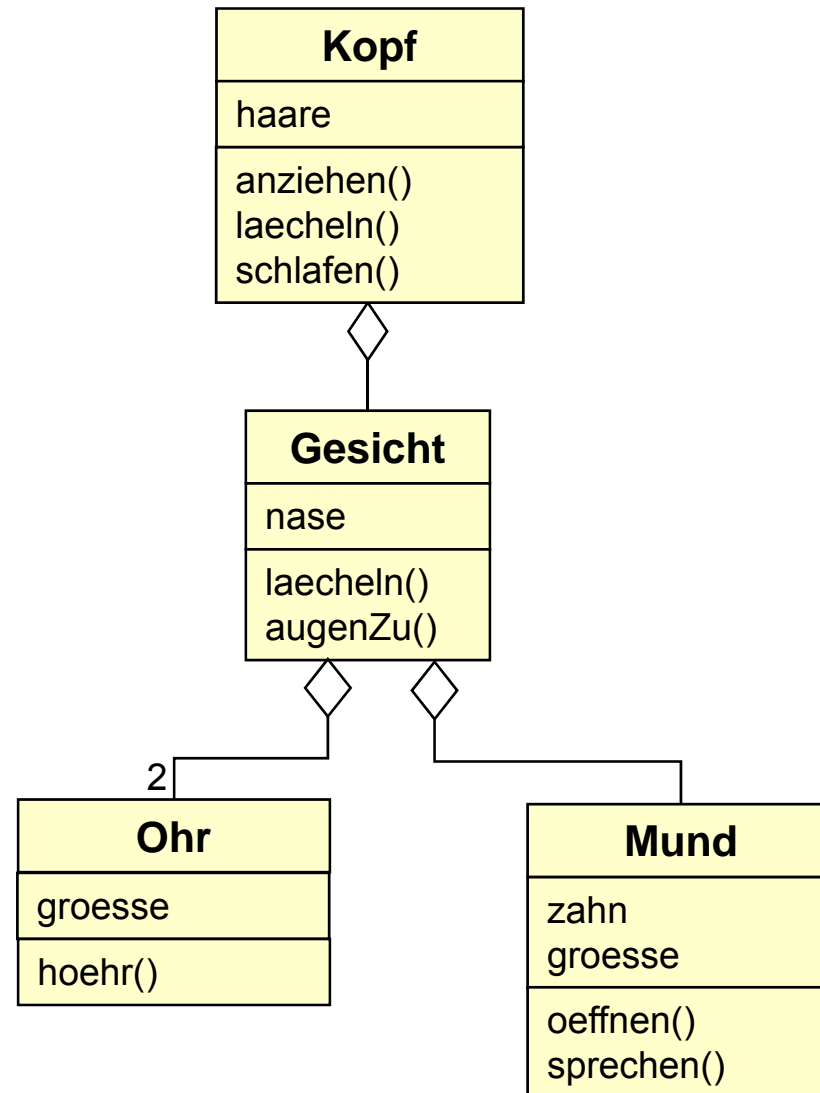
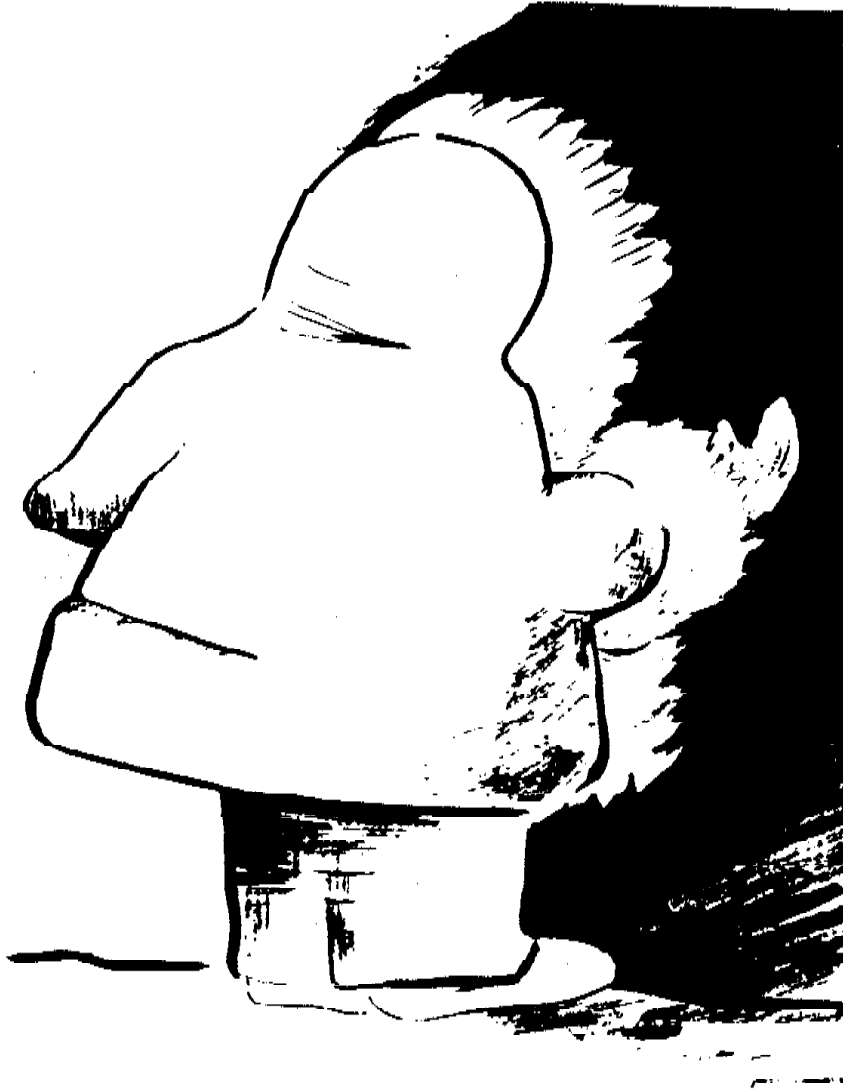


Mehrdeutige
Anforderungen

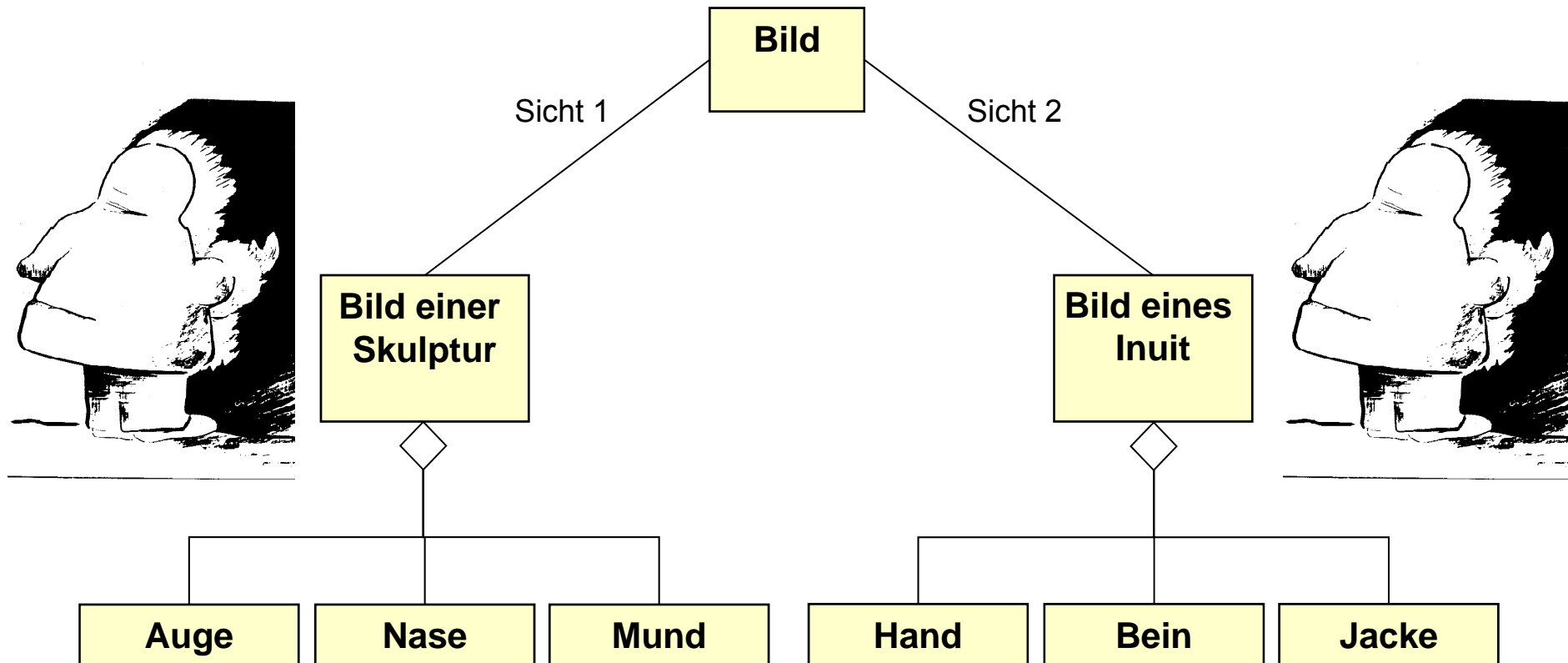
Mögliches Objektmodell ▶ Inuit



Genauso mögliches Objektmodell ▶ Kopf



Objektmodell aus der Sicht des Künstlers



Welches System sollen Sie bauen?



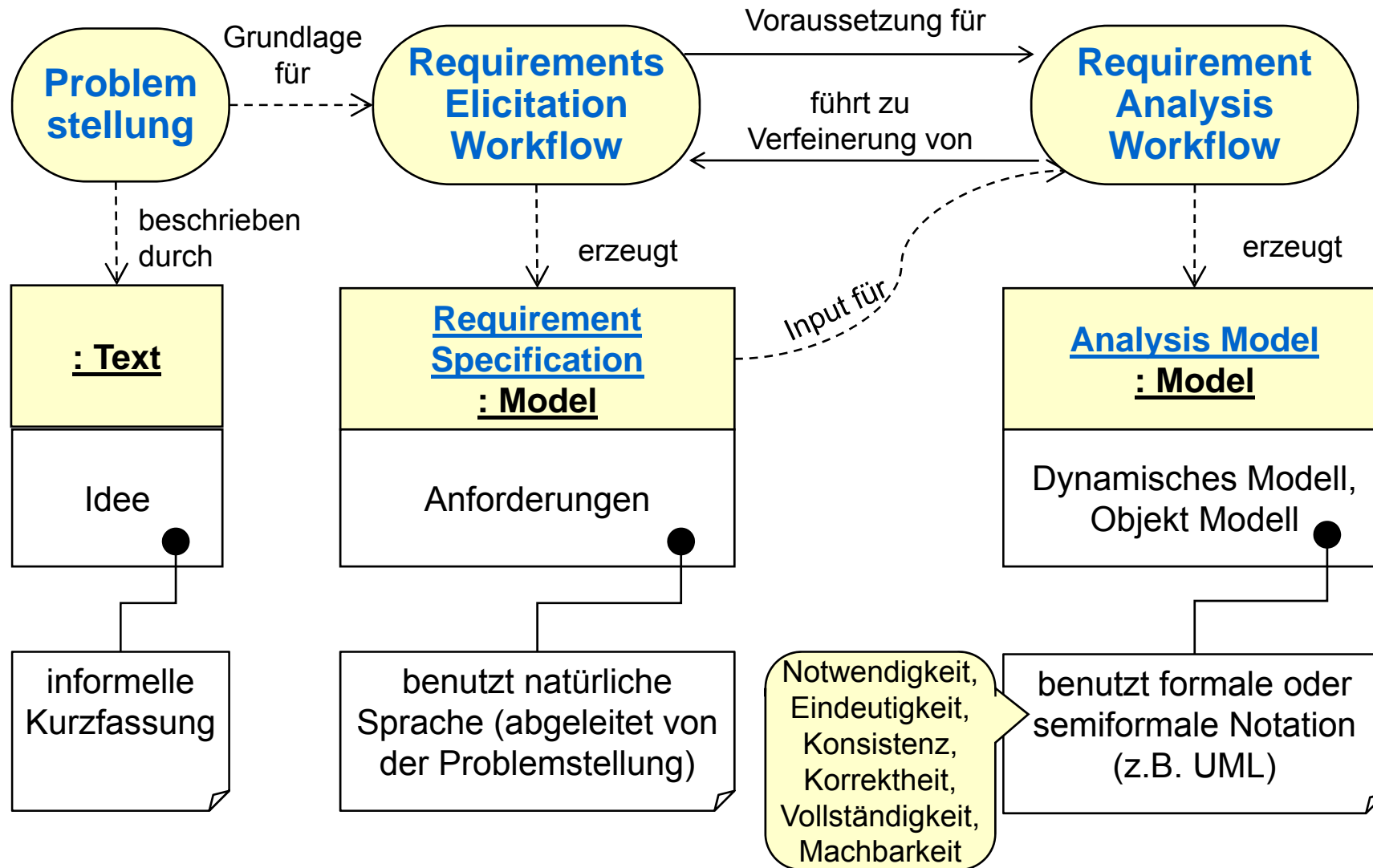
Unscharfe
Anforderungen

Was ist Requirements Engineering (RE)?

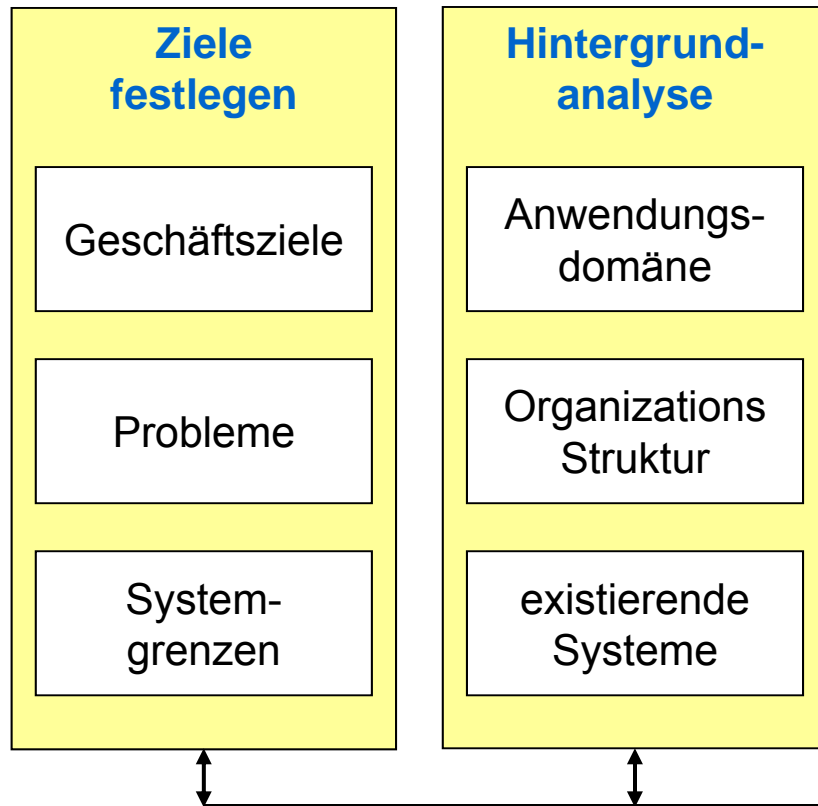
- Ziele
 - ◆ Kennen lernen der **Anwendungsdomäne**
 - ⇒ Identifikation von Konzepten, Beziehungen, grundlegendem Verhalten
 - ◆ Bestimmung der **Anforderungen an das System**
 - ⇒ Identifikation der **Geschäftsprozesse**, die das System unterstützen soll
 - ⇒ Identifikation der **Funktionen** die das System dafür bieten soll

- Aktivitäten („Workflows“)
 - ◆ **Anforderungserhebung** („Requirements Elicitation“):
 - ⇒ Definition des Systems in einer Form, die Kunden und Entwickler verstehen
 - ◆ **Anforderungsanalyse** („Requirements Analysis“)
 - ⇒ Technische Spezifikation des Systems in einer für die Entwickler verständlichen und handhabbaren Form.

Der Requirements Engineering Prozess: Aktivitäten und Produkte

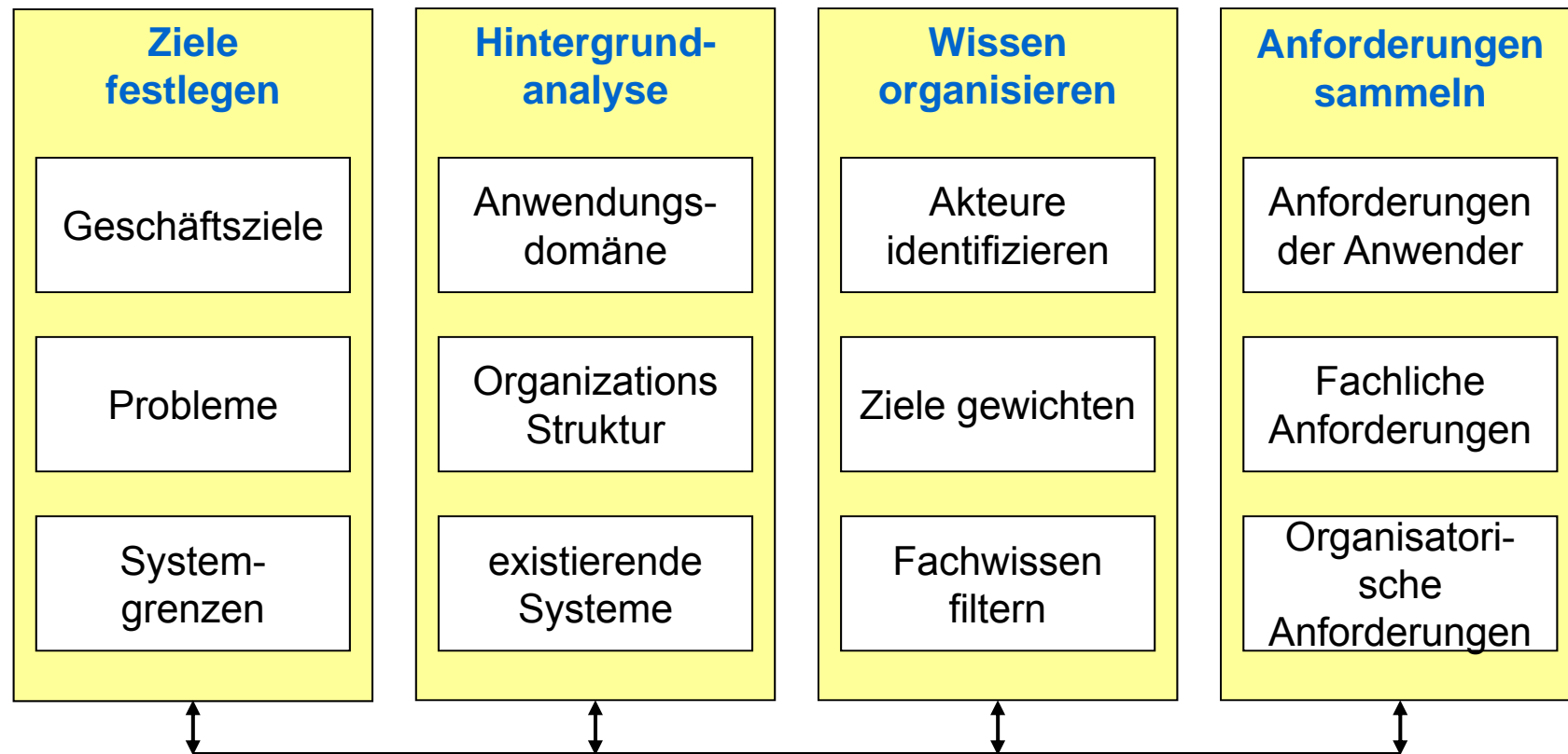


Anforderungserhebung ▶ Erhebungsstufen (1)



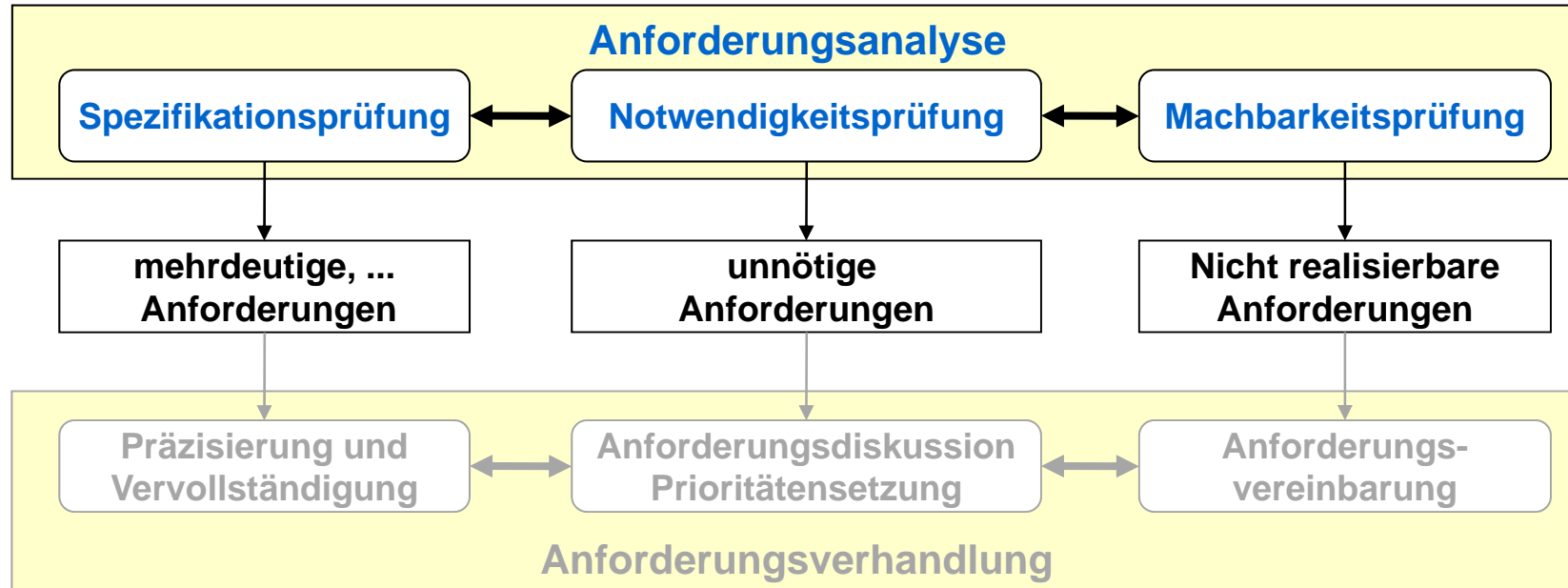
- **Zielsetzung:** Unternehmensziele identifizieren: Warum wird das System gebraucht
- **Hintergrundanalyse:** Sammeln und verstehen von Hintergrundinformationen über das System

Anforderungserhebung ▶ Erhebungsstufen (2)



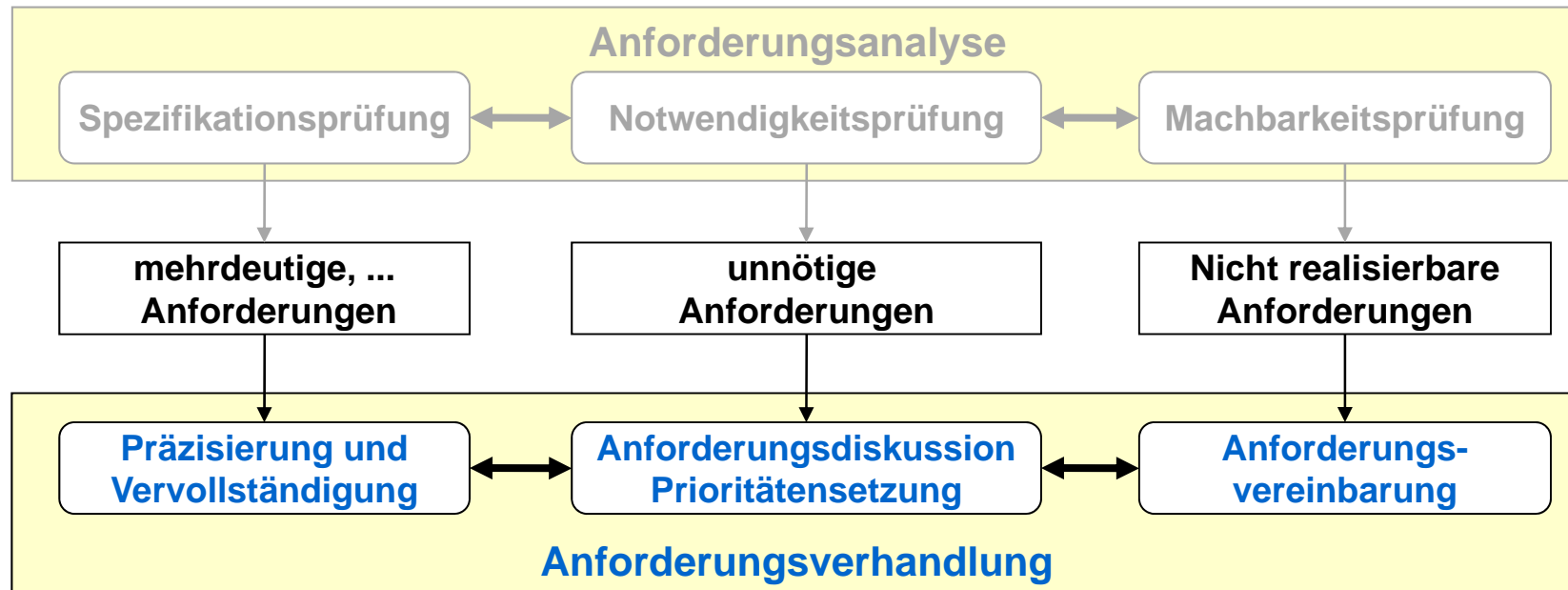
- **Wissensorganisation:** Organisation, Gewichtung und Zuordnung der Daten die in den vorherigen Phasen gesammelt wurden
- **Anforderungen sammeln:** Die Nutzer des Systems mit einbinden um deren Anforderungen zu erfahren

Anforderungsanalyse



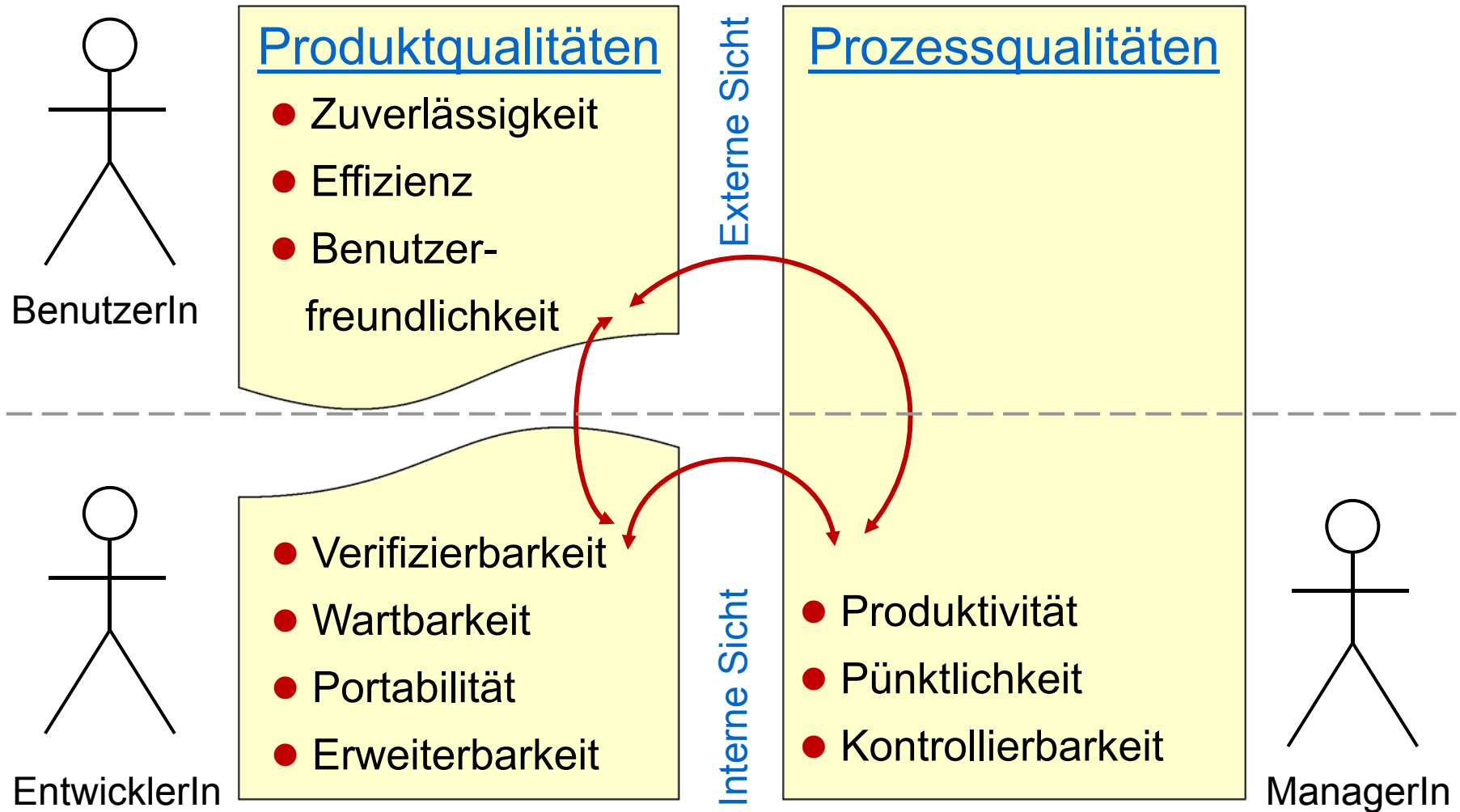
- **Spezifikationsprüfung**
 - ◆ Überkreuzprüfung der Eindeutigkeit, Konsistenz, Korrektheit und Vollständigkeit
- **Notwendigkeitsprüfung**
 - ◆ Welche Anforderungen tragen nicht zu den Geschäftszielen der Firma bei?
- **Machbarkeitsprüfung**
 - ◆ Budget und Zeitplan einhaltbar?

Anforderungsverhandlung



- **Anforderungsdiskussion**
 - ◆ Problematische Anforderungen mit Kunden und Anwendern diskutieren
- **Prioritätensetzung**
 - ◆ Identifizierung der kritischen Anforderungen
- **Anforderungsvereinbarung**
 - ◆ Verständigung auf die zu erfüllenden Anforderungen (und gegebenenfalls Änderungen von Anforderungen)

Qualität ▶ Verschiedene Sichten



5.1 Anforderungserhebung („Requirements Elicitation“)

5.1.1 Anforderungen, Szenarien und Anwendungsfälle („Use Cases“)

5.1.2 Aus Szenarien Anwendungsfälle destillieren

5.1.3 UML Anwendungsfall-Diagramme

5.1.4 Das statische Modell der Anwendungsdomäne („Domain Object Model“)

5.1.5 Dynamische Modellierung der Anwendungsfälle

5.1.6 Zusammenfassung „Anforderungserhebung“

5.1.1 Anforderungen, Szenarien und Anwendungsfälle („Use Cases“)

Arten von Anforderungen

Szenarien und Anwendungsfälle („Use Cases“)

UML Anwendungsfall-Diagramme („Use Case Diagrams“)

Anforderungstypen

- Funktionale Anforderungen
 - ◆ Beschreiben die Interaktionen zwischen dem System und seiner Umgebung unabhängig von der Implementierung
 - ⇒ Die Uhr muss die Zeit ortsabhängig anzeigen
- Nichtfunktionale Anforderungen
 - ◆ Für den Nutzer sichtbare Aspekte des Systems, welche nicht direkt mit dem funktionalen Verhalten in Beziehung stehen.
 - ⇒ Die Reaktionszeit muss unter einer Sekunde liegen
 - ⇒ Die Genauigkeit muss innerhalb einer Sekunde sein
 - ⇒ Die Uhr muss 24 Std am Tag verfügbar sein, außer zwischen 02:00-02:01 und 03:00-03:01
- Nebenbedingungen (“Pseudo requirements”)
 - ◆ Bedingt durch den Kunden oder der Operationsumgebung des Systems
 - ⇒ Die Implementierung muss in COBOL erfolgen unter Verwendung von DB2.
 - ⇒ Muss mit dem Abfertigungssystem von 1956 zusammenarbeiten.

Arten der Anforderungserhebung

- Greenfield Engineering („Planung auf der grünen Wiese“)
 - ◆ Es existiert kein vorheriges System
 - ◆ Die Anforderungen werden vom Kunden und den Endbenutzern bestimmt
 - ◆ Ausgelöst durch Nutzerbedarf
- Reengineering
 - ◆ Re-Design und/oder Re-Implementierung eines existierenden Systems mit neuerer Technologie
 - ◆ Ausgelöst durch neue verfügbare Technologie
- Interface Engineering
 - ◆ Bietet die Dienste eines existierenden Systems in neuer Umgebung
 - ◆ Ausgelöst durch neue verfügbare Technologie oder neuen Marktbedarf

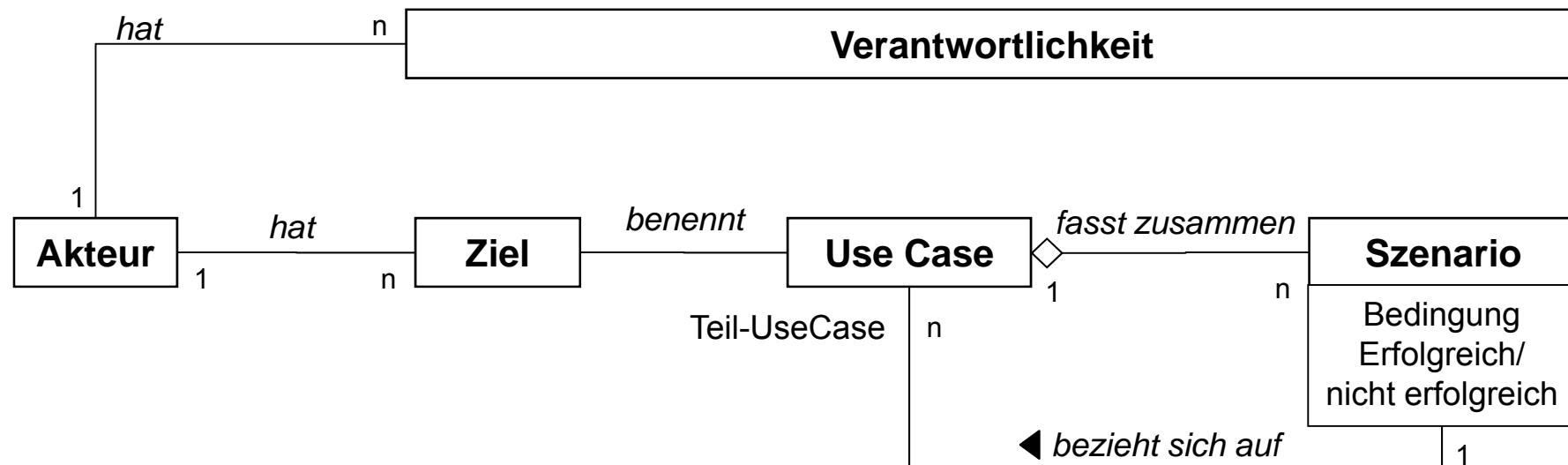
Szenarien und Use Cases

- Herausforderung der Anforderungserhebung: Zusammenarbeit von Leuten mit unterschiedlichem Hintergrund
 - ◆ Nutzer mit Wissen über die **Anwendungsdomäne**
 - ◆ Entwickler mit Wissen über die **Implementierungsdomäne**
- Überbrücken der Lücke zwischen Nutzer und Entwickler
 - ◆ **Szenario**: **Beispiel** der Nutzung des Systems in Form einer Reihe von Interaktionen zwischen externen Akteuren und dem System
 - ◆ **Use Case (UC)**: **Abstraktion**, welche eine Klasse von Szenarien beschreibt

Beziehungen ▶

Akteur-Ziel-UseCase-Szenario

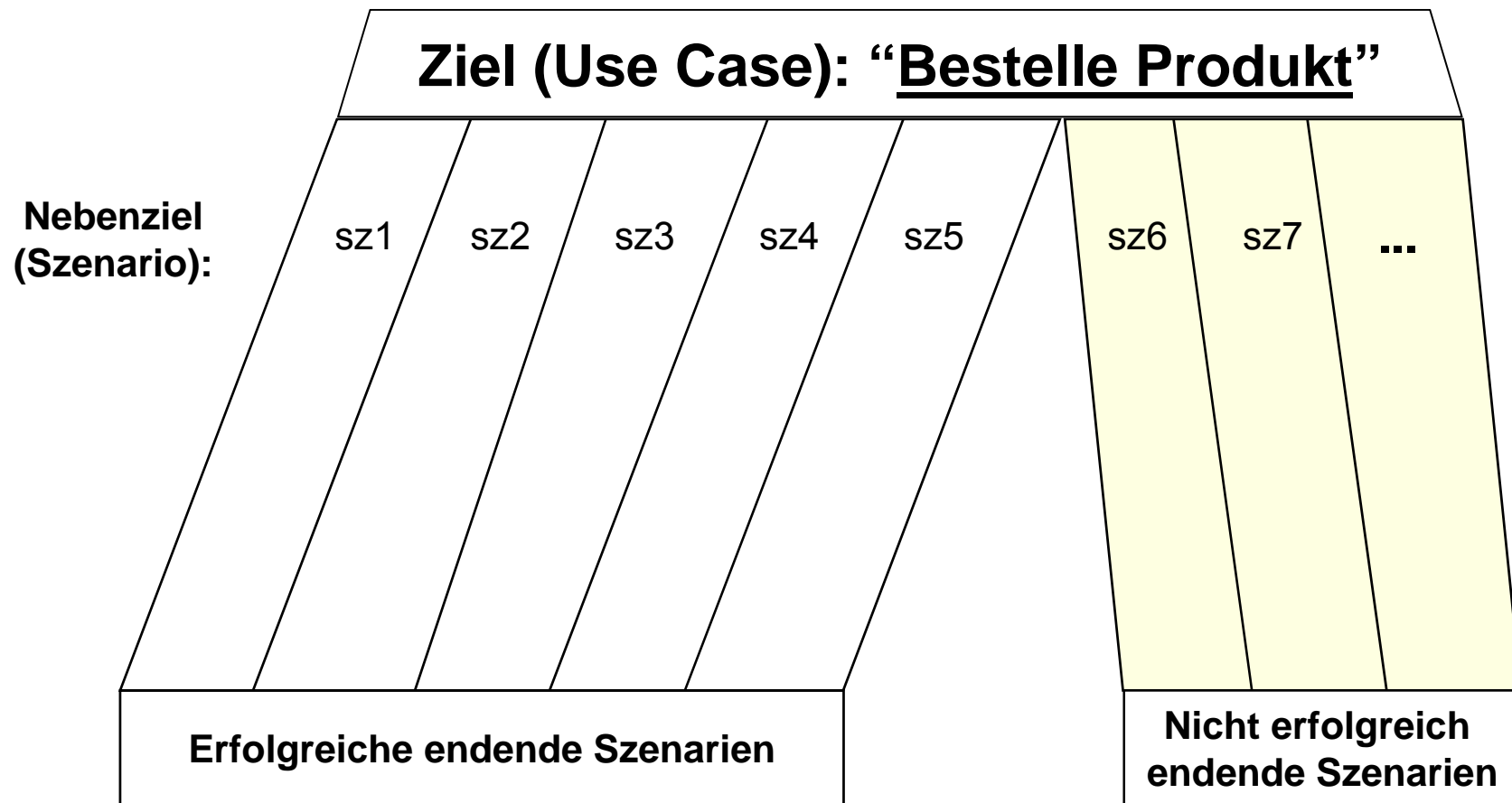
- Ein Akteur hat **Verantwortlichkeiten** aus denen sich **Ziele** ergeben
- Im System umgesetzte **Use Cases** dienen dem Erreichen der **Ziele**
 - ◆ Die Use Cases sind nach den Zielen benannt
- Jeder Use Case fasst eine Menge von **Szenarien** zusammen
- Ein Szenario kann sich auf Teil-Use Cases beziehen.



Ziele, Use Cases und Szenarien (1)

- Ein Ziel fasst eine Systemfunktion in einer verständlichen, überprüfbaren Weise zusammen
- Ein Use Case verbindet ein Ziel mit den zugehörigen Szenarien
 - ◆ Der Name des Use Case ist seine Zielsetzung:
“*Bestelle Produkt*”
 - ◆ Beachte die Grammatik: das aktive Verb steht am Anfang
- Ein Szenario spezifiziert wie sich eine Voraussetzung auswirkt
 - ◆ Szenario (1): Alles funktioniert wie vorhergesehen...
 - ◆ Szenario (2): Zu wenig Guthaben...
 - ◆ Szenario (3): Produkt nicht mehr vorrätig...

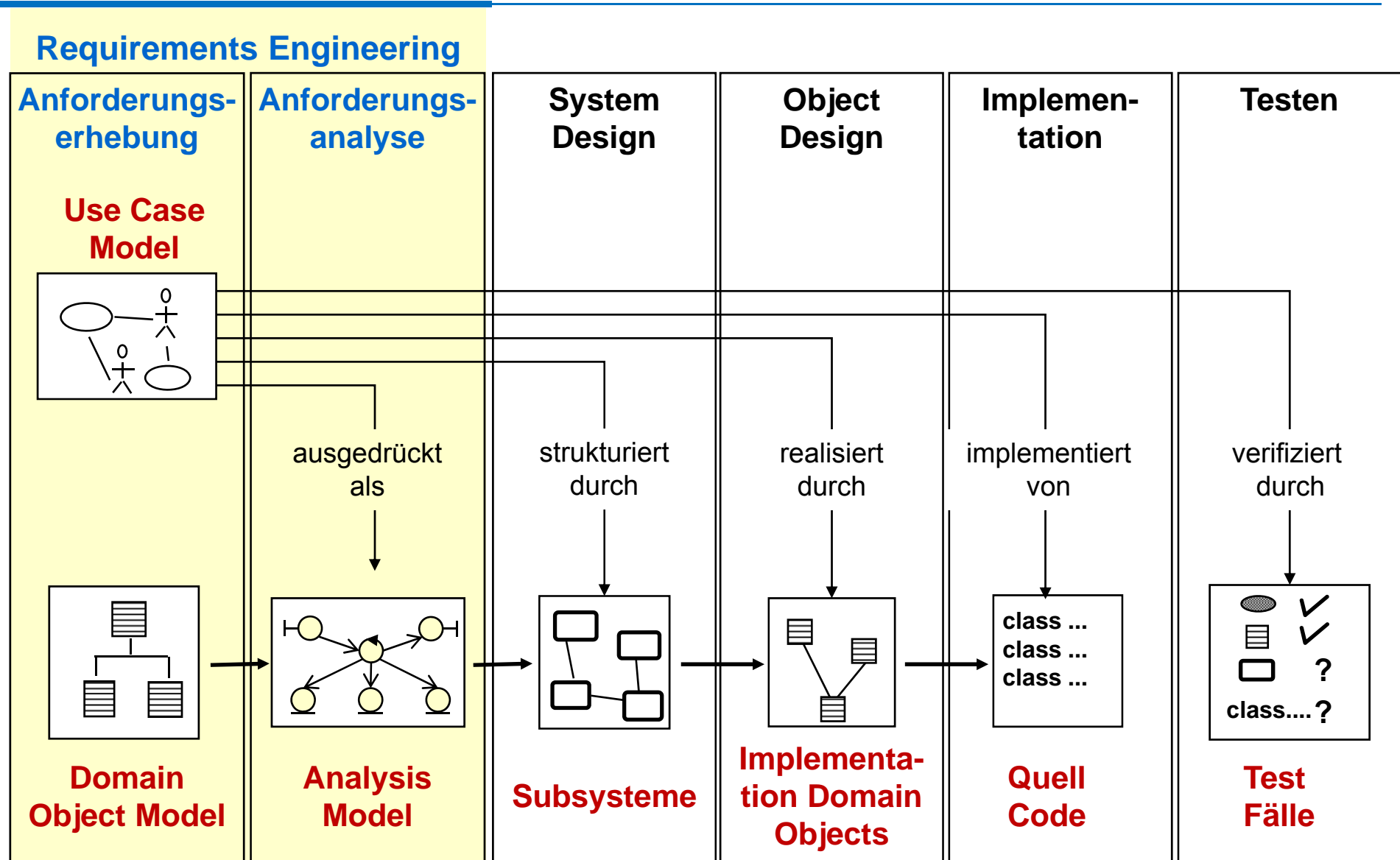
Ziele, Use Cases und Szenarien (2)



Wozu Szenarien und Use Cases?

- Nachvollziehbar für Nutzer
 - ◆ Use Cases modellieren ein System aus Sicht des Nutzers (funktionale Anforderungen)
 - ⇒ Bestimmung jedes möglichen Ereignisflusses durch das System
 - ⇒ Beschreibung von Interaktionen zwischen Objekten
- Großartiges Werkzeug um ein Projekt zu managen. Use Cases können eine Basis für den gesamten Entwicklungsprozess sein
 - ◆ Gebrauchsanleitung
 - ◆ Systemdesign und Objektdesign
 - ◆ Implementierung
 - ◆ Testspezifikation
 - ◆ Akzeptanztest beim Kunden
- Eine exzellente Grundlage für inkrementelle & iterative Entwicklung
- Use Cases wurden auch zur Restrukturierung von Geschäftsprozessen vorgeschlagen (Ivar Jacobson)

Use-Case-getriebener Softwareentwicklungsprozeß



* Die erzeugten dynamischen Modelle (und Andere) sind hier aus Platzgründen nicht explizit dargestellt.

Aktivitäten der Anforderungsanalyse: Ein Szenario- und Use-Case-getriebener Ansatz

- Identifiziere Akteure
- Identifiziere Szenarien
- Identifiziere Use Cases
- Verfeinere Use Cases
- Identifiziere Beziehungen zwischen Use Cases
- Identifiziere nichtfunktionale Anforderungen
- Identifiziere beteiligte Objekte
- Erstelle Glossar

Szenarien

- “Eine erzählende Beschreibung dessen, was Menschen tun und erfahren wenn sie versuchen, Computersysteme und Anwendungen zu benutzen”
[M. Carrol, Scenario-based Design, Wiley, 1995]
- Eine konkrete, fokussierte und informelle Beschreibung einer einzelnen Funktionalität eines Systems, aus Sicht eines einzelnen Akteurs.
- Szenarien können während eines Software-Lebenszyklus an vielen Stellen Anwendung finden.

Arten von Szenarien

- As-is Szenario
 - ◆ Zur Beschreibung einer momentanen Situation. Normalerweise während des Reengineering genutzt. Der Benutzer beschreibt das System.
- Visionäres Szenario
 - ◆ Zur Beschreibung eines zukünftigen Systems. Normalerweise genutzt beim Greenfield Engineering oder Reengineering.
 - ◆ Kann oft weder von Benutzer noch Entwickler alleine erstellt werden
- Evaluationsszenario
 - ◆ Benutzeraufgaben, anhand derer das System bewertet wird
- Trainingsszenario
 - ◆ Schritt-für-Schritt Anweisungen, um einen Neuling durch das System zu führen

Wie finden wir Szenarien?

- Erwarte keine **klaren** Aussagen vom Kunden, solange das System (noch) nicht existiert („greenfield engineering“)
 - ◆ Vorstellungen klären sich erst wenn man etwas Konkretes sieht / nutzen kann
- Erwarte auch bei existierendem Alt-System nicht auf **zuverlässige** Informationen
 - ◆ Diskrepanz zwischen Soll und Ist
 - ◆ Kann nicht bewusst sein oder darf evtl. nicht offiziell genannt werden
- Versuche einen dialektischen Ansatz (evolutionär, inkrementell)
 - ◆ Du hilfst dem Kunden, die Anforderungen zu formulieren
 - ◆ Der Kunde hilft dir, sie zu verstehen
 - ◆ Die Anforderungen entfalten sich während der Entwicklung der Szenarien

Wie finden wir Szenarien?

- Stelle dir selbst oder dem Kunden die folgenden Fragen
 - ◆ Was sind die primären Aufgaben des Systems?
 - ◆ Welche Daten möchte der Akteur im System anlegen, speichern, ändern, löschen oder einfügen?
 - ◆ Über welche externen Änderungen muss das System informiert sein?
 - ◆ Bei welchen Änderungen oder Ereignissen soll der Akteur des Systems informiert werden?
- Wenn ein System existiert, bestehede darauf zu beobachten, wie die Aufgaben damit bearbeitet werden (z.B. bei Interface-Engineering oder Reengineering)
 - ◆ Bitte um ein Gespräch mit dem Endbenutzer, nicht nur mit dem Auftraggeber!
 - ◆ Erwarte Widerstand und versuche ihn zu überwinden → Erkläre dem Auftraggeber die Bedeutung der Akzeptanz durch Benutzer

Wie finden wir Szenarien?

▶ Interview-Techniken

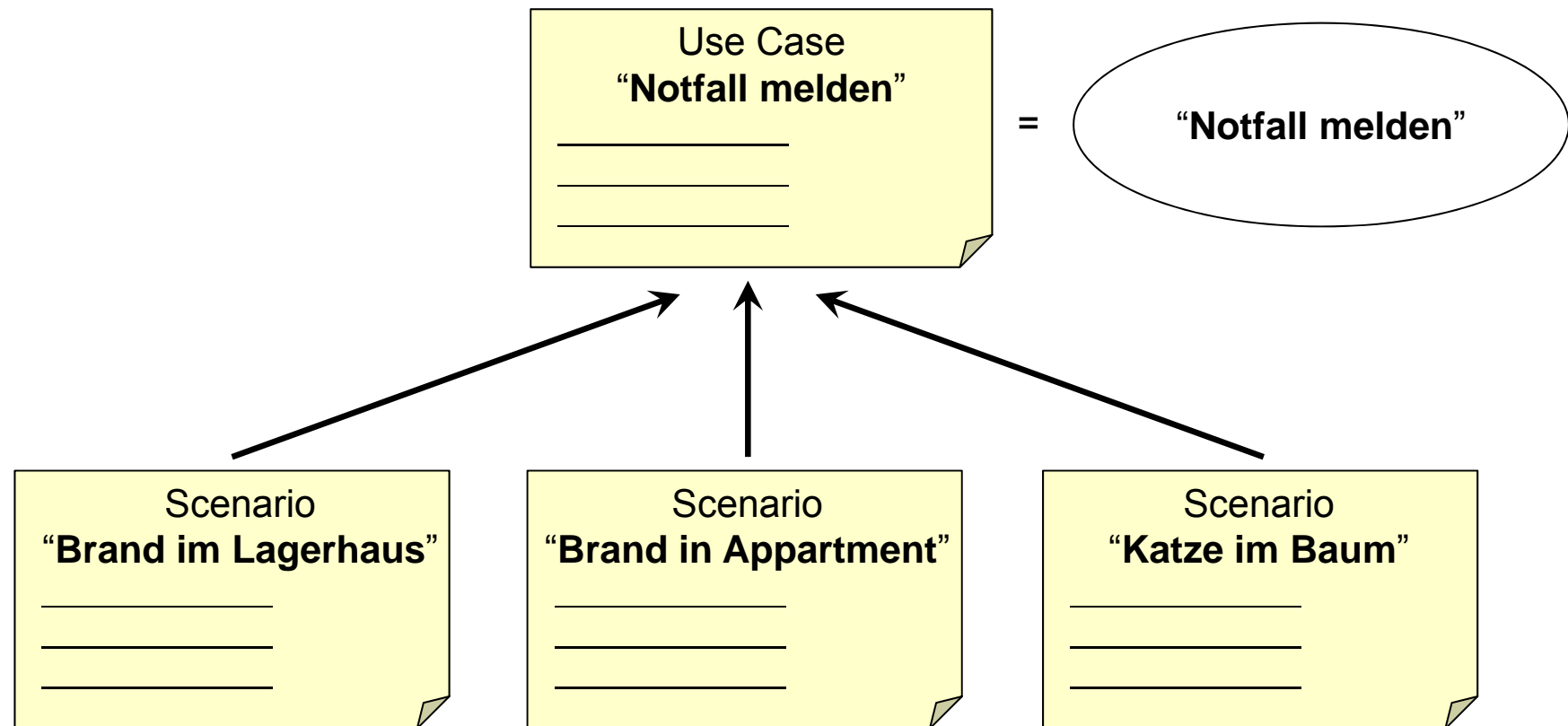
Siehe Exkurs in Ergänzungsfoliensatz „04-exkurs-interviews“.

5.1.2 Von Szenarien zu Use Cases

Aus Szenarien Use Cases destillieren an einem Beispiel

Aus Szenarien Use Case destillieren

- Use case ▶ Abstraktion zusammengehöriger Szenarios
- Szenario ▶ Konkreter Einzelfall = Instanz eines Use Case



Beispiel ▶ „Brand im Lagerhaus“ Szenario

- **Bob**, der die Hauptstraße mit seinem Streifenwagen entlangfährt, bemerkt Rauch der aus einem Lagerhaus dringt.
- Seine Kollegin **Alice** meldet den Notfall von ihrem Wagen aus. **Alice** gibt die Adresse des Gebäudes ein, eine kurze Beschreibung des Ortes (z.B., nordwestliche Ecke) und eine Notfallstufe. Zusätzlich zu einem Feuerwehrgewagen fordert sie mehrere **Sanitäter** an, da die Gegend sehr belebt scheint. Sie bestätigt ihre Eingabe und wartet auf Bestätigung.
- **John, der Disponent**, wird durch einen Piepton seiner Workstation alarmiert. Er überprüft die Informationen von **Alice** und bestätigt ihre Meldung. Er schickt einen Feuerwehrgewagen und zwei **Sanitäter** zum Unglücksort und sendet **Alice** ihre voraussichtliche Ankunftszeit.
- **Alice** empfängt die Bestätigung und die voraussichtliche Ankunftszeit.

Beispiel ▶ Notfall-Management-System

- Was benötigt man, wenn jemand einen „Brand in einem Lagerhaus“ meldet?
 - ◆ Wer ist an der Meldung eines Unfalls beteiligt?
 - ◆ Was tut das System, wenn keine Polizeiwagen verfügbar sind?
 - ◆ Was, wenn der Polizeiwagen auf dem Weg zum Brand einen Unfall hat?
- Was ist zu tun, um eine „Katze im Baum“ zu melden?
 - ◆ Was tut man, wenn die „Katze im Baum“ zu einer „von der Leiter gefallenen Oma“ wird?
 - ◆ Kann das System mit gleichzeitigen Brand-im-Lagerhaus-Meldungen umgehen? Wie?

Aus Szenarien Use Case destillieren

- Finde eine „Use Case“-Bezeichnung, die alle Szenarien adäquat beschreibt
 - ◆ “Notfall melden“ im zweiten Paragraph des „Brand im Lagerhaus“-Szenarios ist ein möglicher Use Case
- Erstelle eine strukturierten textuellen Beschreibung des Use Cases
 - Siehe Schema auf nächster Folie
 - Siehe Beispiel auf übernächster Folie

Strukturierte Use Case Beschreibung

► Schema

- Name des Use Case und Kurzbeschreibung
- Akteure
 - ◆ Beschreibung der am Use Case beteiligten Akteure
- Vorbedingung
 - ◆ Was gelten muss, damit der Use Case durchgeführt werden kann
- Ereignisfluss
 - ◆ Schritte die im Standardablauf des Use Case durchgeführt werden
- Nachbedingung
 - ◆ Was nach erfolgreichem Ende des Ereignisflusses gilt
- Sonderfälle (Alternativabläufe und Fehlerfälle)
 - ◆ Jeweils Name, Verzweigungspunkt („extension point“) im Standardablauf, auslösende Bedingung, Ereignisfluss im Sonderfall und Nachbedingung des Sonderfalls (→ siehe „extends“-Beziehung)
- Spezielle Anforderungen
 - ◆ Nichtfunktionale Anforderungen und Nebenbedingungen

Strukturierte Use Case Beschreibung

▶ Beispiel „Termin erfassen“

Standardablauf

Kurzbeschreibung:	Ein Termin wird für einen oder mehrere Teilnehmer eingetragen.
Akteure:	Benutzer E-Mail-System (zum Versenden von Benachrichtigungen) ...
Vorbedingung:	Benutzer ist dem System bekannt und eingeloggt.
Ereignisfluss:	1. Neuer Termin wird erfasst (Zeit, Ort, ...) 2. Teilnehmer werden zugeordnet. 3. Benutzer ist berechtigt für alle Teilnehmer Termine zu erfassen. 4. Benachrichtigungen werden verschickt. 5. Sichten werden aktualisiert
Nachbedingung:	Neuer Termin ist erfasst. Alle Teilnehmer sind verständigt und alle Sichten sind aktualisiert.
Alternativablauf A1:	3'. Benutzer hat für mind. einen Teilnehmer keine Berechtigung. ...
Fehlerfall F1:	Benutzer hat für keinen Teilnehmer die Berechtigung, einen ...
Nachbedingung zu F1:	Neuer Termin erfasst, aber ohne Zuordnung zu Teilnehmern. ...

Schrittweise Formulierung eines Use Case

▶ Beispiel „Melde Notfall“ (1)

- Benenne zuerst den Use Case
 - ◆ „Melde Notfall“
- Benenne Akteure: Ersetze Namen durch Rollen die die Akteure spielen
 - ◆ „Streifenbeamter“ (Rolle von Bob und Alice)
 - ◆ „Disponent“ (Rolle von John)
- Schreibe den Ereignisfluss auf
 - ◆ Der **Streifenbeamte** betätigt die “Melde Notfall”-Funktion seines Terminals. Das System reagiert durch Anzeige eines Formulars.
 - ◆ Der **Streifenbeamte** füllt das Formular durch Angabe von Stufe, Art und Ort des Notfalls sowie einer kurzen Lagebeschreibung aus. Zudem schlägt er mögliche Reaktionen vor. Wenn es ausgefüllt ist, sendet der **Streifenbeamte** das Formular, und der **Disponent** wird sofort benachrichtigt.
 - ◆ Der **Disponent** überprüft die erhaltenen Informationen und erstellt einen Notfall in der Datenbank durch Aufruf des **OpenIncident** Use Case. Er wählt eine passende Reaktion auf den Notfall aus und bestätigt die Notfallmeldung.
 - ◆ Der **Streifenbeamte** empfängt die Bestätigung und die gewählte Reaktion.

Schrittweise Formulierung eines Use Case

▶ Beispiel „Melde Notfall“ (2)

- Schreibe die **Ausnahmen** auf
 - ◆ **Wenn** die Verbindung zwischen seinem Terminal und der Zentrale abbricht **wird** der Streifenbeamte sofort benachrichtigt, indem ...
 - ◆ **Wenn** die Verbindung zu einem der eingeloggten Streifenbeamten abbricht **wird** der Disponent sofort benachrichtigt, indem ...
- Notiere **Nichtfunktionale Anforderungen und Nebenbedingungen**
 - ◆ Die Meldung des Streifenbeamten wird **innerhalb von 30 Sekunden** bestätigt.
 - ◆ Die gewählte Reaktion trifft **innerhalb von 0,5 Sekunden nach** der Wahl durch den Disponenten ein.

Verwendung der Use Case Beschreibung

▶ Beispiel „Melde Notfall“

Aus der textuellen Beschreibung des Use Cases muss alles Weitere herleitbar sein

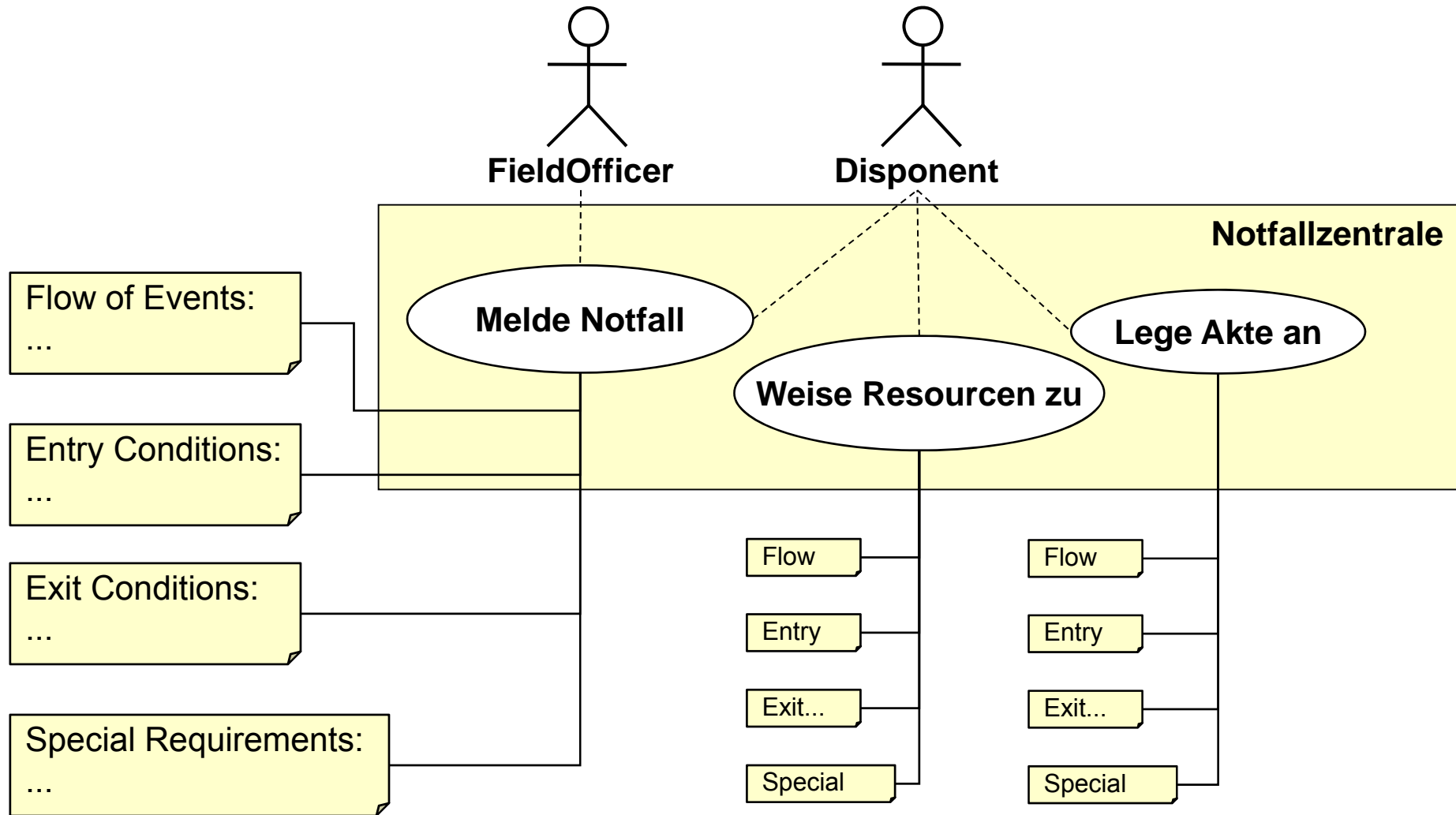
- Konzepte und Beziehungen der Objektdomäne
- Verhalten des Systems

Techniken

- Abbott's Technik zur Objektidentifikation (siehe Abschnitt über → Anforderungs-Analyse)
- Sie kann schon während der Use Case Modellierung genutzt werden zwecks Erstellung des → „Domain Object Model“

5.1.3. Anwendungsfalldiagramme ("UML Use Case Diagrams")

Use Case Diagramm „Notfallzentrale“

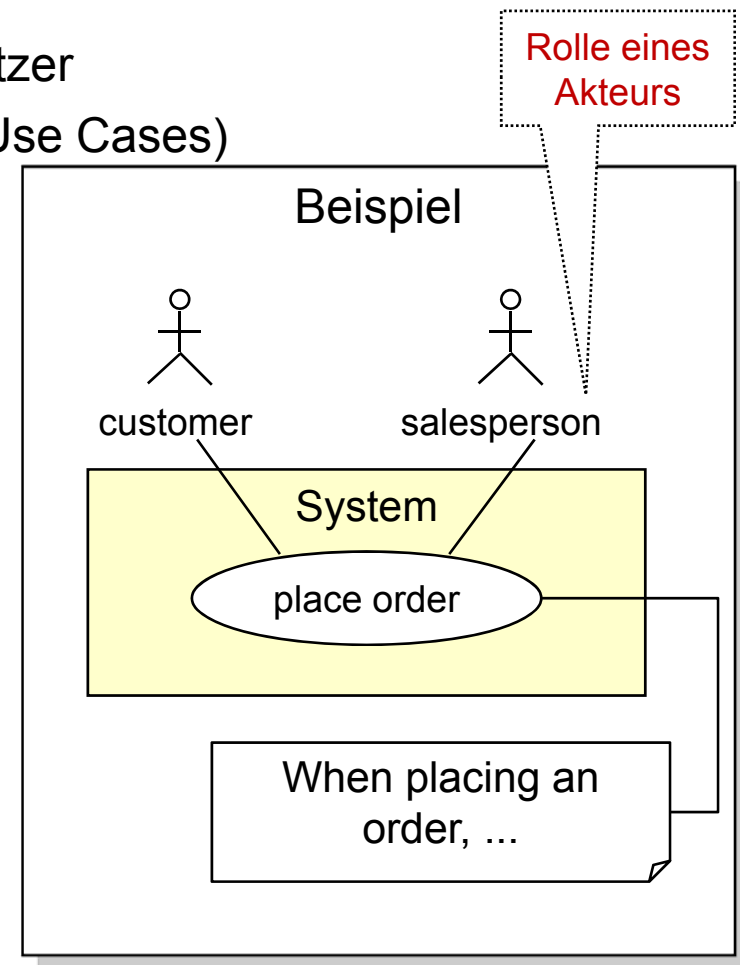
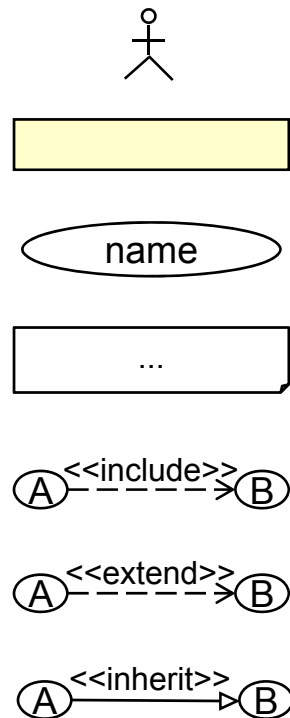


Use-Case Diagramm ▶ Elemente

- Einsatz
 - ◆ Analysephase
 - ◆ Kommunikation mit Auftraggeber / Benutzer
 - ◆ Erfassung von Anwendungsszenarien (Use Cases)

- Elemente

- ◆ Akteure
- ◆ System
- ◆ Use Cases
- ◆ Annotationen
- ◆ Beziehungen



Verfeinerung und Strukturierung von Use Cases

- Ausgangspunkt: Mehrere Use Cases erfasst
- Verfeinere und strukturiere die Use Cases durch Assoziationen
 - ◆ Use Case Assoziation = Beziehung zwischen Use Cases

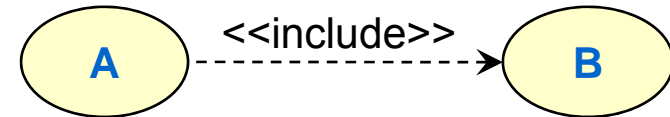
Beziehungen zwischen Use-Cases

- Extend (Sonderfall)
 - ◆ Ein Use Case ergänzt einen anderen unter bestimmten Bedingungen
- Include (funktionale Dekomposition)
 - ◆ Ein Use Case benutzt einen anderen
- Inherit (Generalisierung/Spezialisierung)
 - ◆ Ein abstrakter Use Case mit verschiedenen Spezialisierungen

Beziehungen zwischen Use-Cases

● Include-Beziehung

- ◆ Immer wenn **A** ausgeführt wird, **muss** auch **B** ausgeführt werden
- ◆ **B** ist unbedingt nötig, um **A** auszuführen
- ◆ **B** stellt Verhalten dar, das von mehreren Use Cases oder von einem Akteur direkt genutzt wird



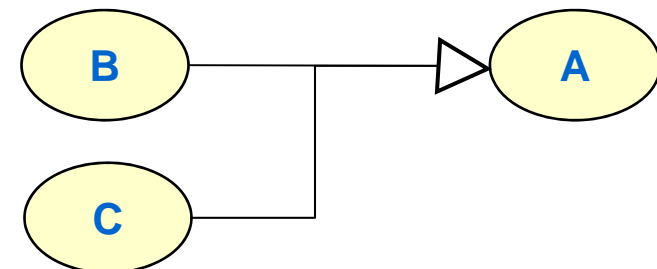
● Extend-Beziehung

- ◆ Wenn **A** ausgeführt wird, **kann** auch **B** ausgeführt werden
- ◆ **B** ist nicht zwingend nötig, um **A** auszuführen
- ◆ **B** stellt Sonder- oder Fehelfälle, optionales oder seltenes Verhalten dar



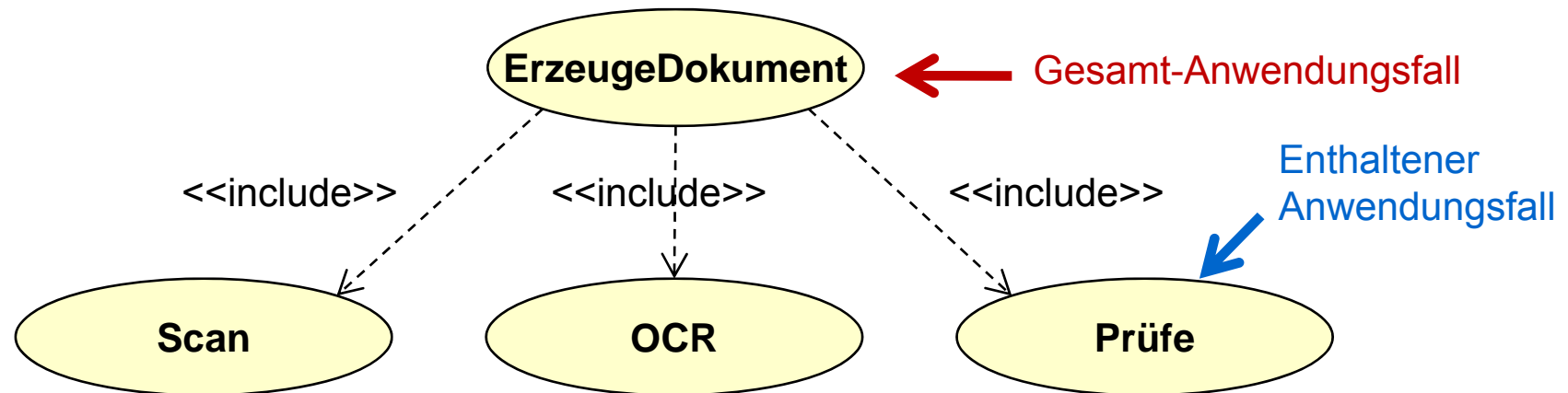
● Generalisierungs-Beziehung

- ◆ **B** und **C** sind jeweils spezielle Ausprägungen des **gesamten** Ablaufs von **A**
- ◆ **Nur eine** der verschiedenen Spezialisierungen wird durchgeführt, die aber **komplett**



<<include>>-Beziehung ▶ Semantik

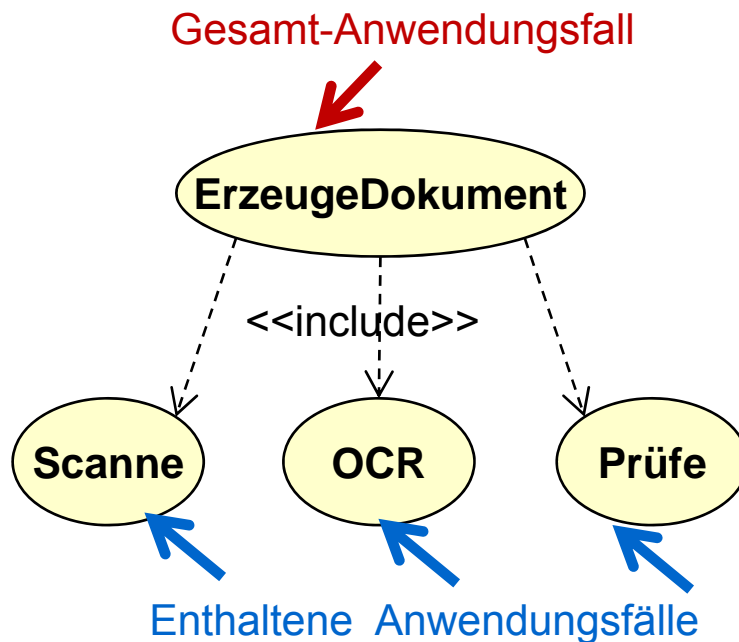
- Jeder Ablauf des Gesamt-Anwendungsfalls beinhaltet einen Ablauf des enthaltenen Anwendungsfalls
- Die include-Beziehung von A nach B bedeutet, dass A all das Verhalten von B ausführt (“A delegiert an B”).
- A ist abhängig von B (daher auch die Richtung und Art des Pfeiles: der übliche Abhängigkeitspfeil, lediglich mit einem passenden Stereotyp)



<<include>>-Beziehung ▶ Verwendung

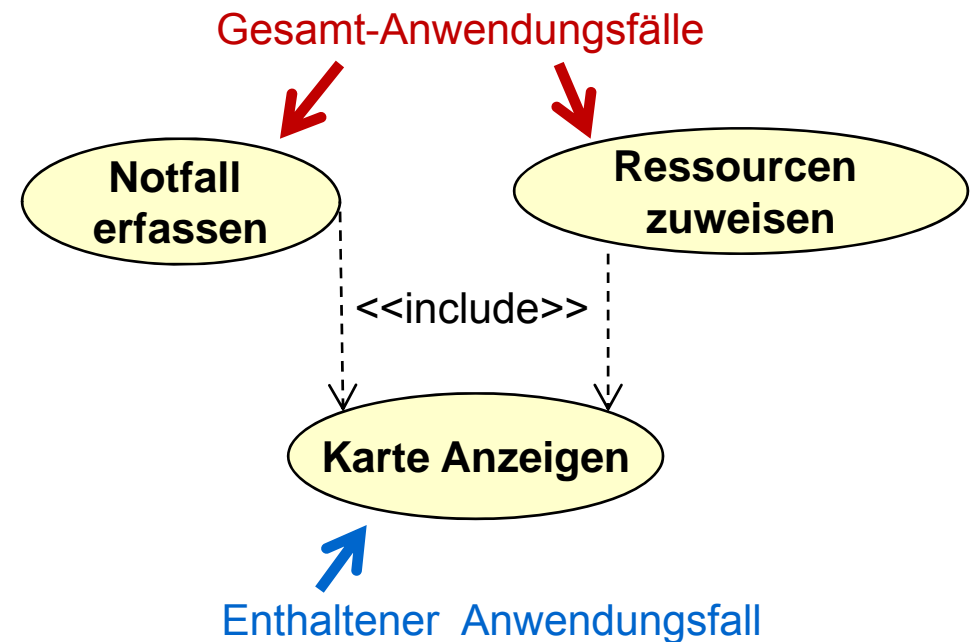
Funktionale Dekomposition

- Problem
 - ◆ Ein Use Case ist zu komplex und muss zerlegt werden.



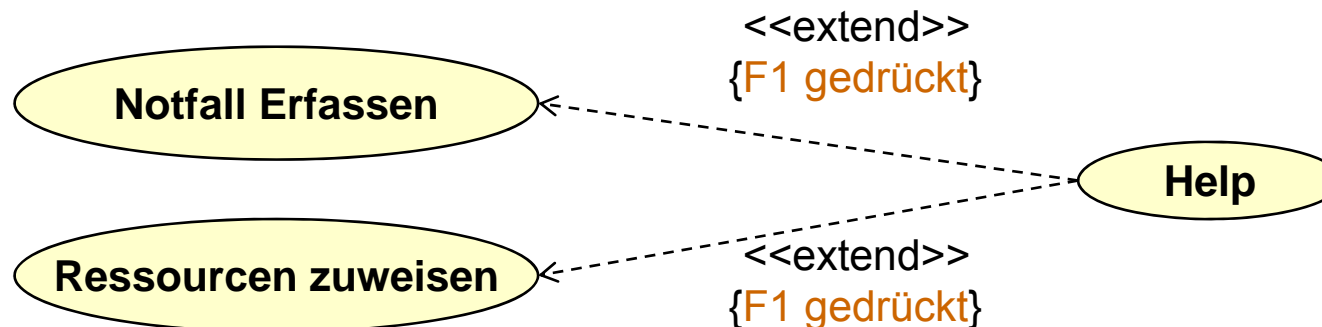
Gemeinsame Verwendung

- Problem
 - ◆ Gemeinsames Verhalten verschiedener Use Cases muss ausgedrückt werden



<<extend>>-Beziehung

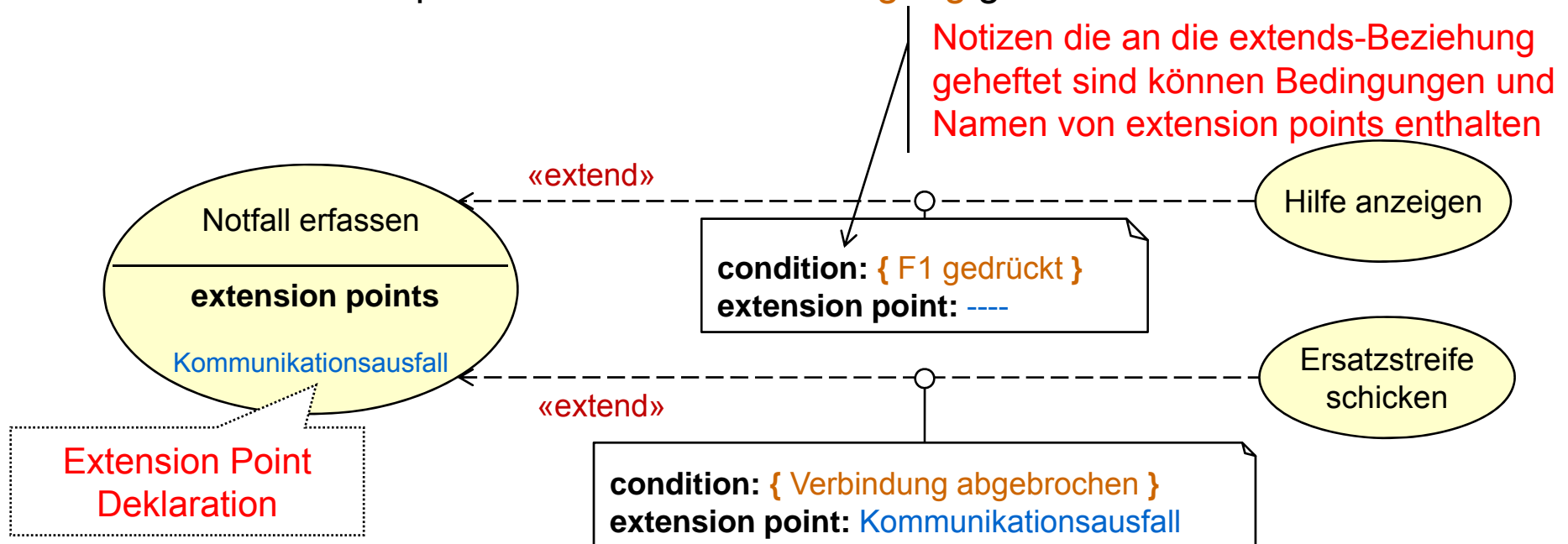
- Problem
 - ◆ Unter bestimmten Bedingungen (Sonderfälle, Fehlerfälle, ...) muss der Ablauf eines Use-Cases erweitert werden
- Lösung / Semantik
 - ◆ Eine extend-Beziehung von Use Case A nach B bedeutet, dass A eine Erweiterung von B ist, die nur unter der **angegebenen Bedingung** aktiv ist.
 - ◆ Der erweiterte UC ist unabhängig vom erweiternden UC.
- Beispiel
 - ◆ “Notfall erfassen” ist komplett, kann aber in einem Szenario, in dem der Benutzer Hilfe braucht, durch „Help“ erweitert werden.



<<extend>>-Beziehung

► Extension Points (1)

- Deklaration im zu erweiternden Use Case
 - ◆ Benannte Stellen im **Ereignisfluss** des erweiterten Use Case, wo der Ablauf des erweiternden Use Cases einzufügen ist
- Bezugnahme in <<extends>>-Beziehung
 - ◆ Erweiternder Use Case wird am **Extension Point** aktiviert
 - ◆ ... falls die spezifizierte **extends-Bedingung** gilt



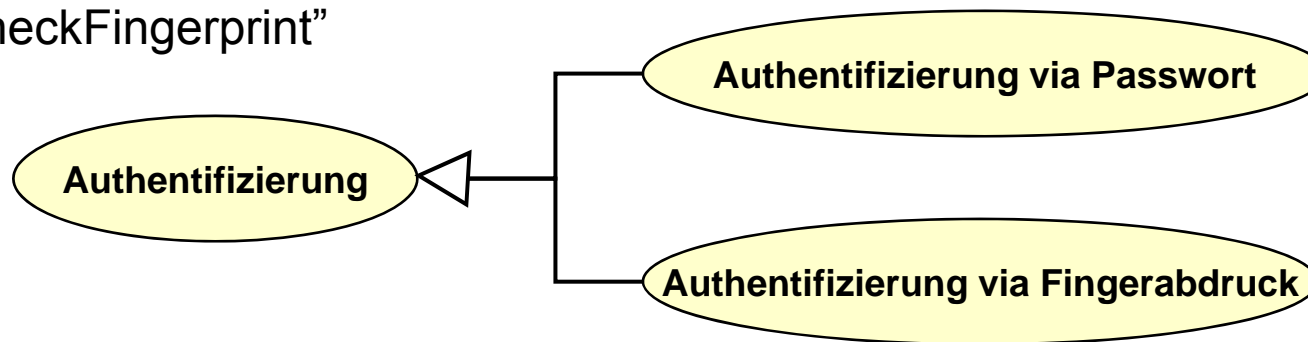
<<extend>>-Beziehung

▶ Extension Points (2)

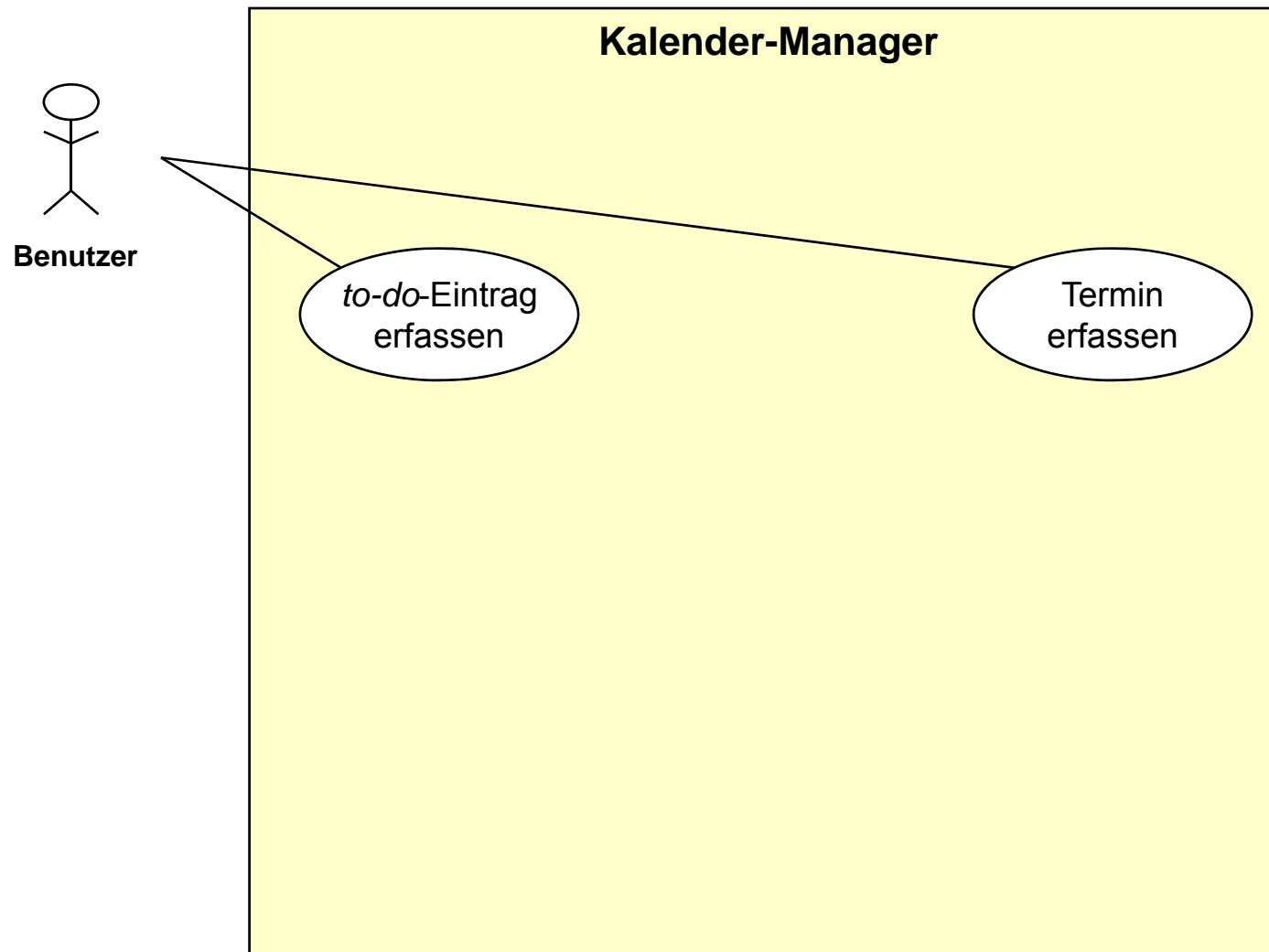
- Extension points werden im zu erweiternden Use Case deklariert
 - ◆ Sie sind Namen für die Stellen im Ereignisfluss des erweiterten UC, wo der Ablauf des erweiternden UC einzufügen ist
- Die Angabe von Extension points an einer <<extends>>-Beziehung spezifiziert, dass die **Verhaltensfragmente** des erweiternden Use Case an den **jeweiligen Extension Points** aktiviert werden sollen
 - ◆ Meist gibt es aber pro Use Case nur ein Verhaltensfragment und somit nur einen Erweiterungspunkt auf den Bezug genommen wird.
 - ◆ Sonst werden die verschiedenen Fragmente der Reihe nach an den jeweiligen Erweiterungspunkten aktiviert
 - ⇒ Erstes Fragment am ersten Erweiterungspunkt,
 - ⇒ zweites Fragment am zweiten Erweiterungspunkt,
 - ⇒ ...
- **Vorschau: Extends-Beziehung und Aspektorientierte Programmierung!**

Generalisierung von Use Cases

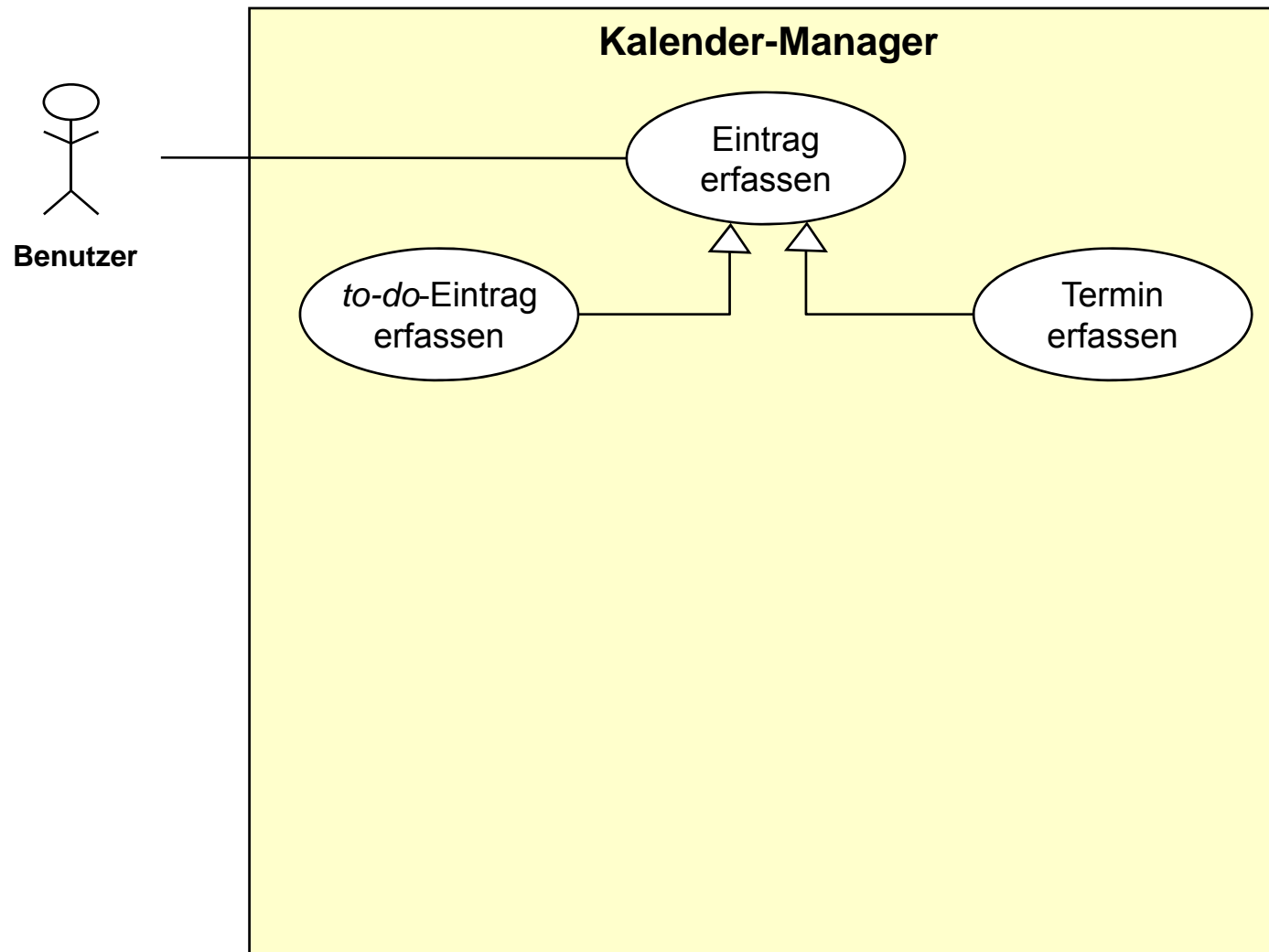
- Problem
 - ◆ Man möchte in Use Cases wiederkehrendes Verhalten auslagern.
- Lösung
 - ◆ Die Generalisierungsbeziehung lagert gleiches Verhalten von Use Cases aus. Die „Kinder“ erben die Kommunikationsbeziehungen, die Bedeutung und das Verhalten, der „Eltern“, ersetzen Teile davon und fügen neues hinzu.
- Beispiel
 - ◆ “ValidateUser” ist verantwortlich für die Überprüfung der Benutzeridentität. Der Kunde könnte zwei Umsetzungen fordern: “CheckPassword” und “CheckFingerprint”



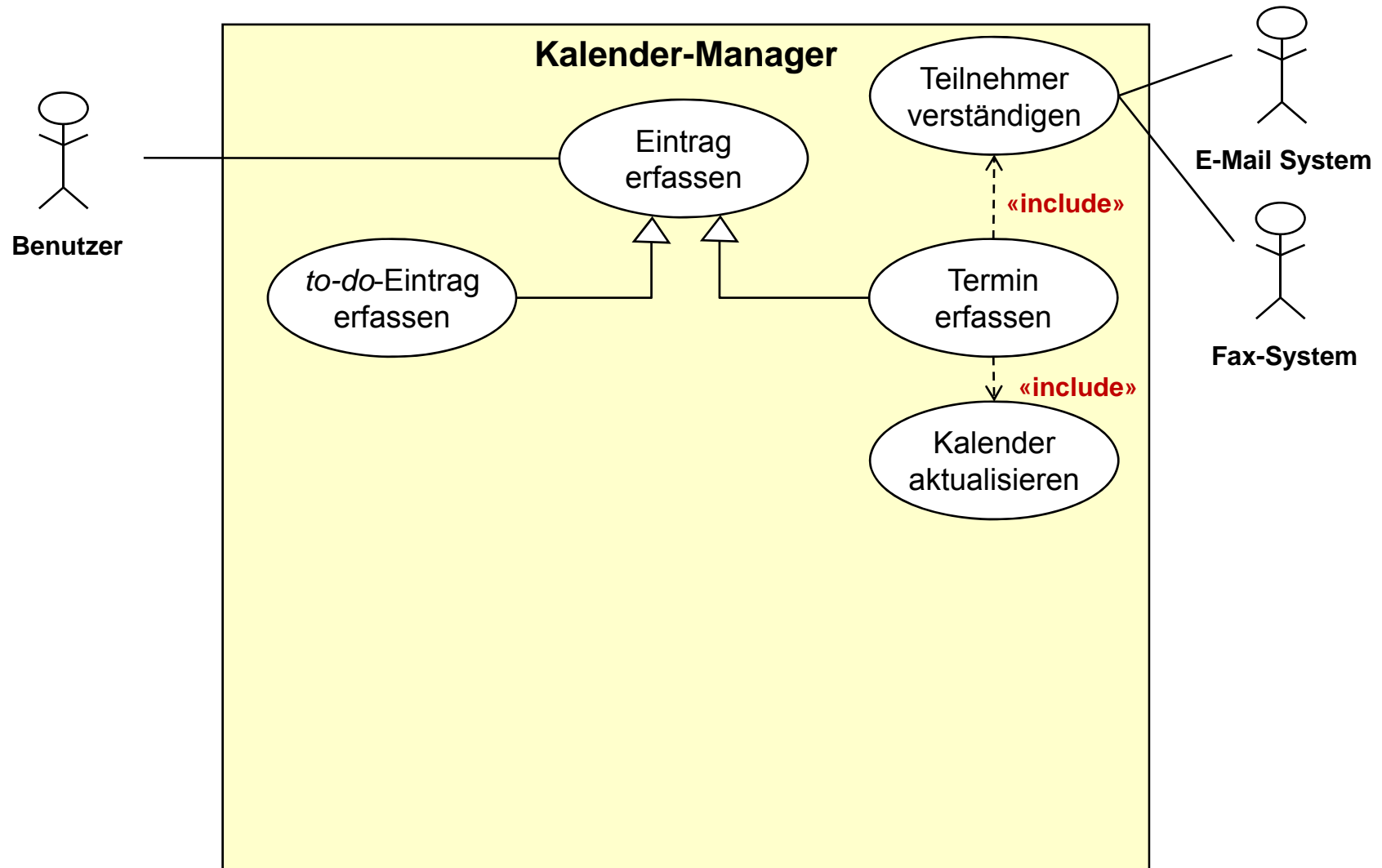
Beispiel „Kalender-Manager“



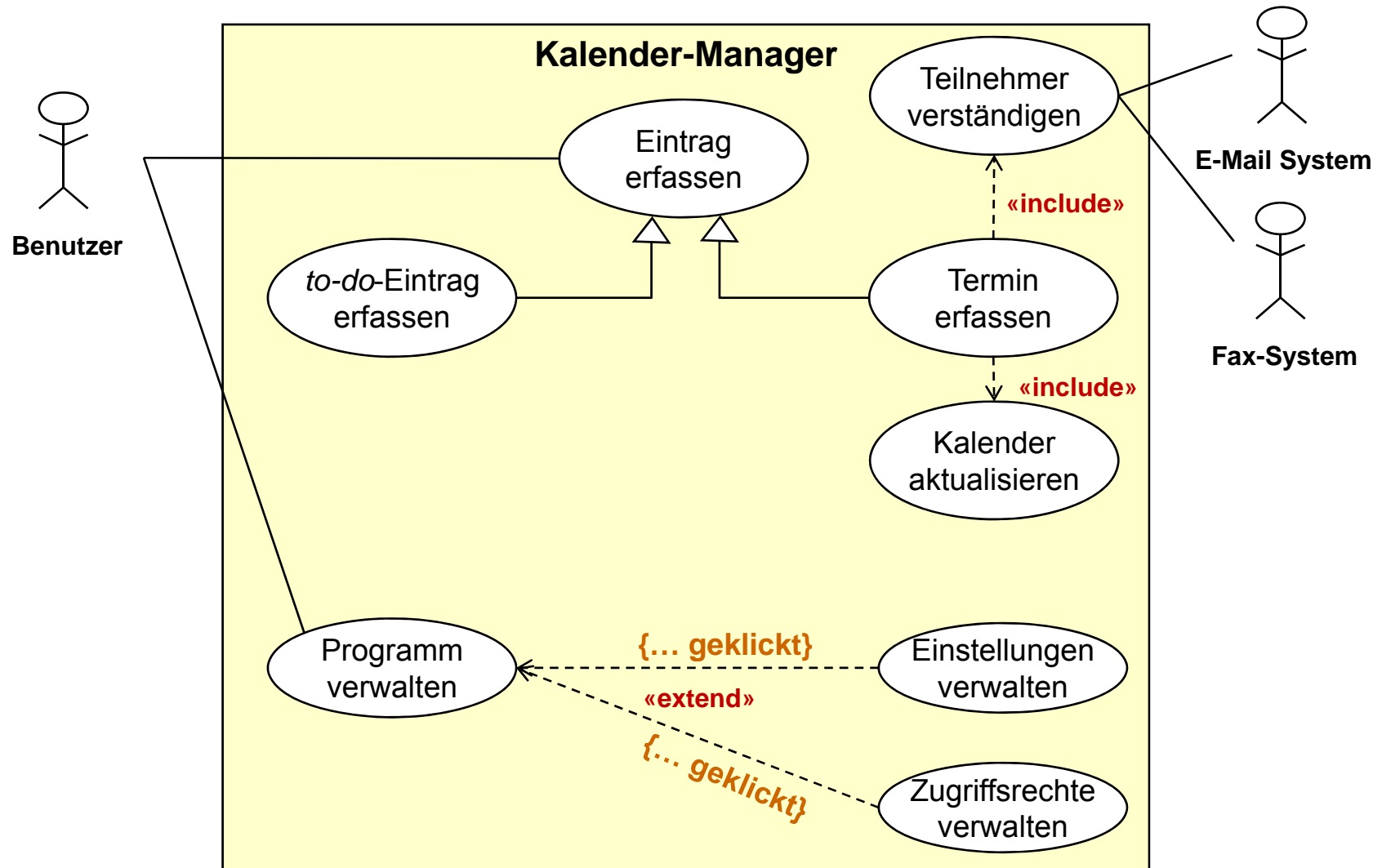
Beispiel „Kalender-Manager“ ▶ Varianten der Eintragungserfassung als Generalisierung



Beispiel „Kalender-Manager“ ▶ «include» für Teilschritte der Terminerfassung

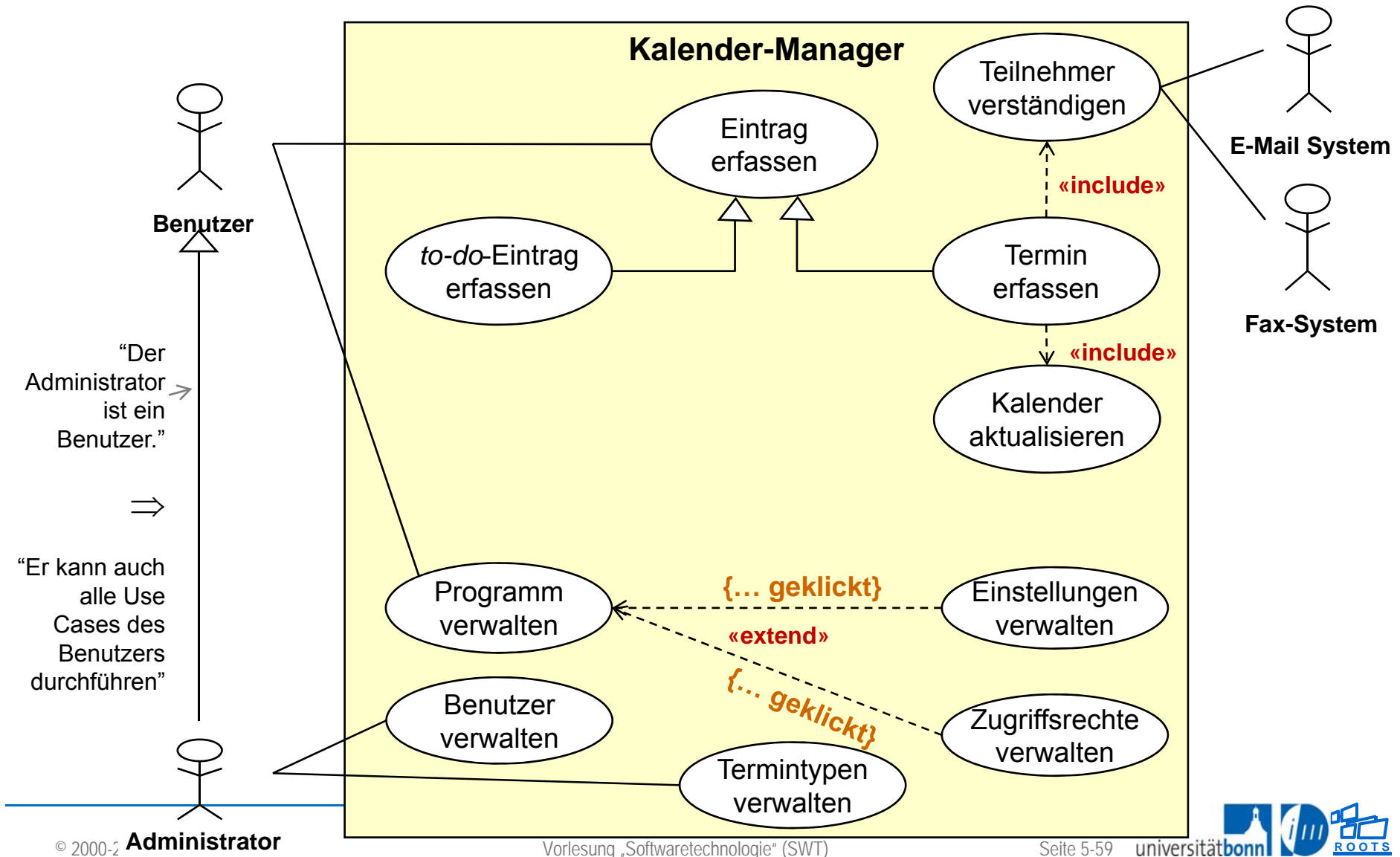


Beispiel „Kalender-Manager“ ▶ <<extend>> für Sonderfälle der Programmverwaltung



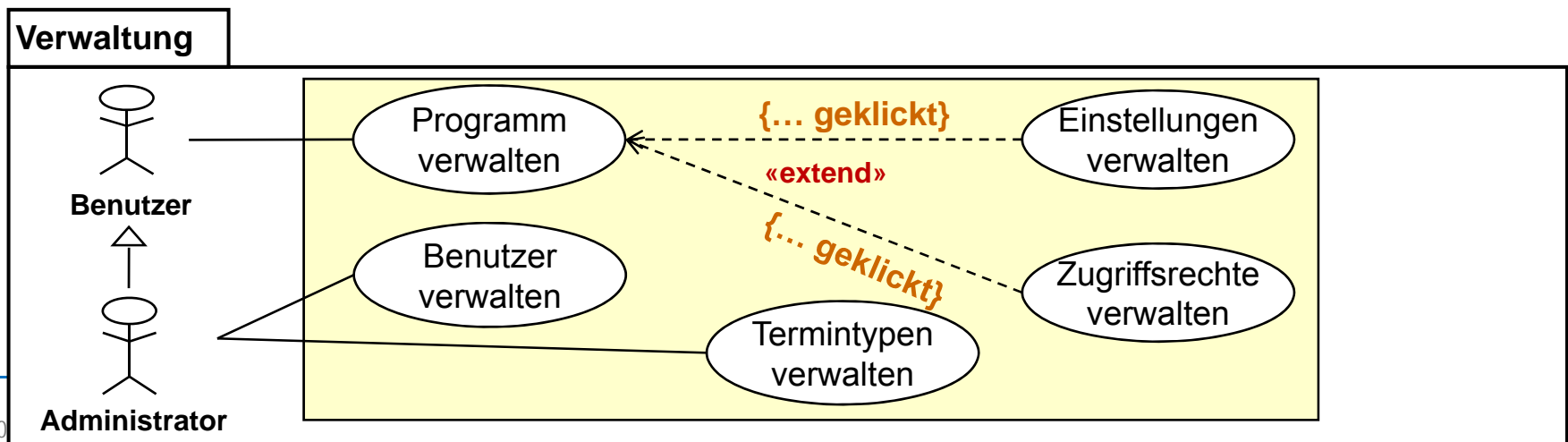
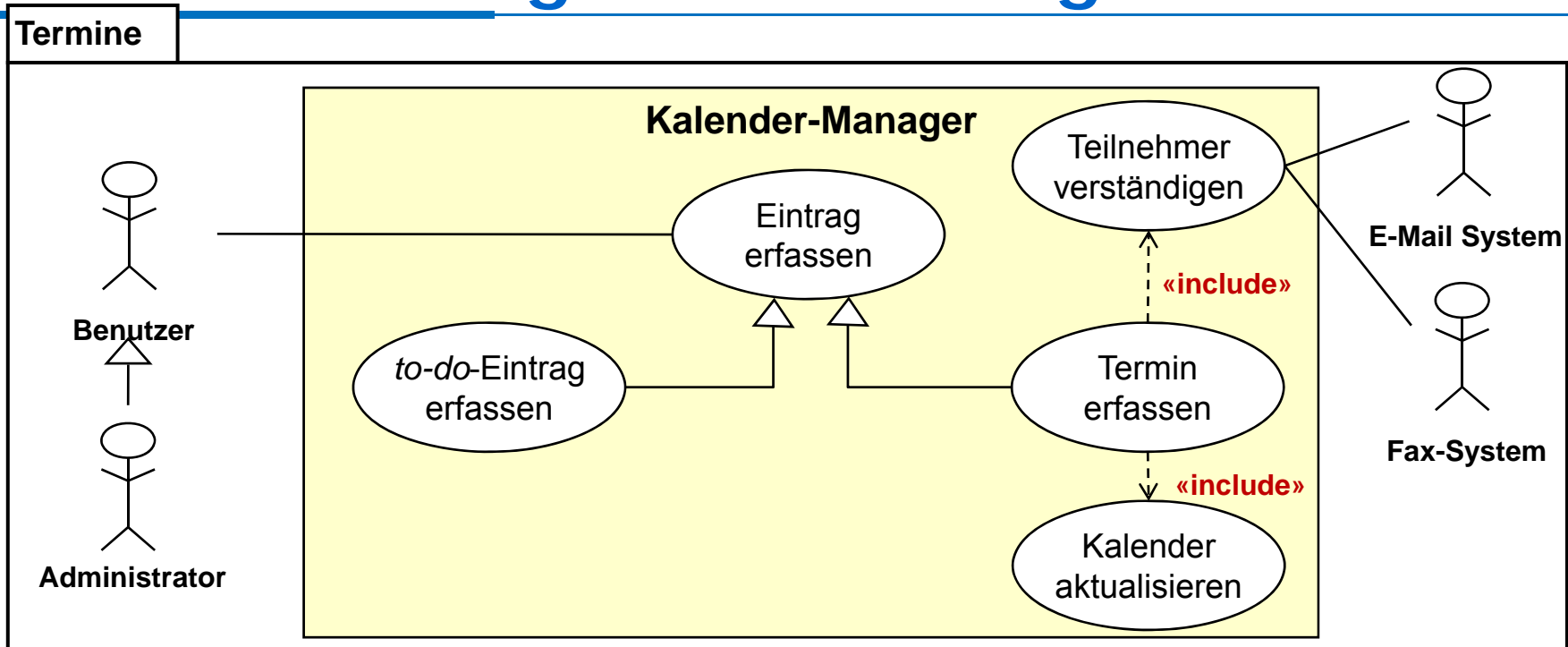
Beispiel „Kalender-Manager“ ▶

Generalisierung zwischen Akteuren

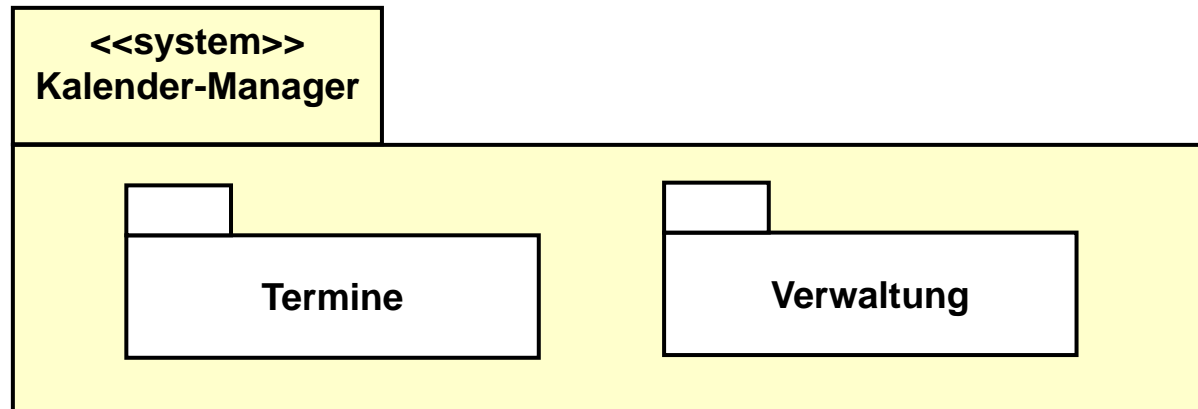


Beispiel „Kalender-Manager“ ▶

Modularisierung durch Packages



Beispiel „Kalender-Manager“ ▶ Gesamt-System als geschachtelte Pakete



- Pakete können weitere Artefakte beinhalten
 - ◆ Aktivitätsdiagramme
 - ◆ Sequenzdiagramme
 - ◆ Klassendiagramme des Domain Object Model
 - ◆ Textuelle Beschreibungen der Use Cases

5.1.4 Domain Object Model

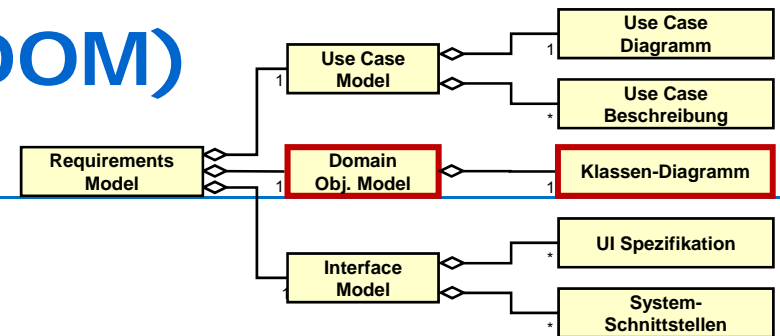
Abbott's Technik zur Objektidentifikation

● Objektmodellierung

Beispiele

Grundlagen der Objektmodellierung auf dem Detaillierungsgrad der Anforderungserhebung. Weitere Notationen, Techniken und Methoden der Objektmodellierung werden später schrittweise ergänzt

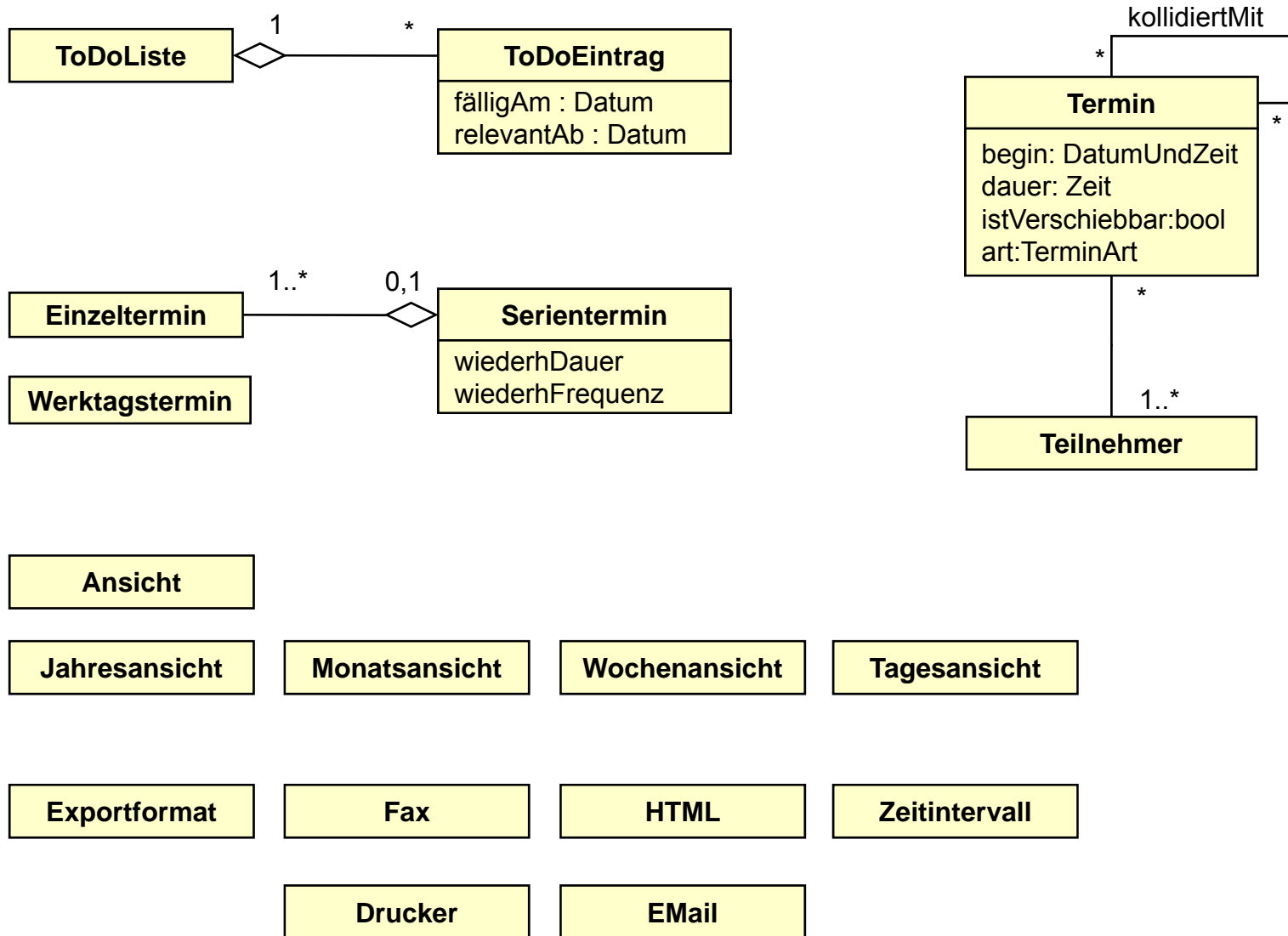
Domain Object Model (DOM)



- Das DOM ist eine Menge von Klassendiagrammen, die Konzepte der Anwendungsdomäne beschreiben
 - ◆ Klassen
 - ◆ Attribute
 - ◆ Beziehungen
 - ◆ (wenige Operationen)
- Vorgehensweise
 - ◆ Dialog mit Benutzer → Textuelle Anforderungsspezifikation (Use Cases)
 - ◆ Hauptwörter-Erfassung und –Filterung → Verfahren von Abbott

Domain Object Model ▶ Beispiel

„Terminverwaltung“



Objektmodellierung

- Hauptziel
 - ◆ Allgemein: Finden der strukturellen Abstraktionen des Systems
 - ◆ Während Anforderungserhebung: Finden der *wichtigen* Abstraktionen des Systems soweit sie *für den Anwender beobachtbar* sind.
- Schritte der Objektmodellierung
 - ◆ 1. Identifikation von Typen
 - ◆ 2. Identifikation von Attributen
 - ◆ 3. Identifikation von Methoden
 - ◆ 4. Identifikation von Assoziationen zwischen Typen
- Reihenfolge der Schritte ist nicht wichtig
 - ◆ Ziel: Erreichen der gewünschten Abstraktionen
 - ◆ Wenn wir zu falschen Abstraktionen kommen iterieren wir das Ganze um das Modell zu korrigieren

An Use Cases beteiligte Objekte finden

Für jeden Use Case führe folgende Schritte durch:

- Finde Begriffe die Nutzer oder Entwickler klären müssen um den Ereignisfluss zu verstehen
- Identifiziere Entitäten der realen Welt, die das System im Auge behalten muss. Beispiele: `FieldOfficer`, `Disponent`, `Resource`
- Identifiziere Vorgänge der realen Welt, die das System im Auge behalten muss. Beispiel: `Notfallplan`
- Identifiziere Datenquellen oder -senken. Beispiel: `Drucker`
- Identifiziere Schnittstellen. Beispiel: `PoliceStation`
- Führe eine Textanalyse zum Finden zusätzlicher Objekte durch (Abbott's Technik)
- Modelliere den Ereignisfluss mit einem dynamischem Diagramm

Fokus hier

Beispiel ▶ Ein Szenario aus der Problembeschreibung

- Jim Smith ist Kunde bei einer großen Bank.
- Dort besitzt er ein kostenpflichtiges Konto mit einem Kontostand von derzeit 2.000 Euro.
- Jim Smith geht zum Schalter und zahlt 100,- Euro ein.
- Am nächsten Tag betritt Jim Smith die Bank erneut. Er geht zum Geldautomat und hebt 400,- Euro ab.

Gibt uns diese Beschreibung Hinweise, wie unser Objektmodell aussehen sollte?

Beispiel ▶ Ein Szenario aus der Problembeschreibung

- Jim Smith **ist Kunde** bei einer **großen Bank**.
- Dort **besitzt** er ein **kostenpflichtiges Konto** mit einem **Kontostand** von derzeit **2.000 Euro**.
- Jim Smith **geht** zum **Schalter** und **zahlt** **100,- Euro ein**.
- Am nächsten Tag **betrifft** Jim Smith die **Bank** erneut. Er **geht** zum **Geldautomat** und **hebt** **400,- Euro ab**.

Was haben wir hier gemacht? Satzelemente kategorisiert!
→ Abbott's Textanalyse

Textanalyse von Abbott [Abbott 1983]

- Zuordnung von Teilen der Sprache zu Komponenten des Objektmodells
- Wird vor allem genutzt bei Erstellung des Domain Object Models und Analysemodells

<i>Sprachelement</i>	<i>Modellelement</i>	<i>Beispiel</i>
Eigenname	Objekt	Jim Smith
Nomen	Klasse	Kunde, Konto, Bank
„ist“	Generalisierung	Sparkonto ist ein Konto
„hat“, „enthält“, ...	Aggregation	Ein Konto hat eine Nummer
Modalverb („müssen“, „können“, „dürfen“, „sollen“)	Einschränkung (<i>Constraint</i>)	Kontostand darf bestimmte Grenze nicht unterschreiten
Adjektiv	Attribut	kostenpflichtig
Transitives Verb	Methode	bringen
Intransitives Verb	Methode (Event)	erscheinen

Textanalyse von Abbott [Abbott 1983]

Abbott's Technik ist eine Hilfestellung für denkende Menschen, nicht ein starres Regelwerk für Automaten!

- Die Entsprechungen sind **Hinweise, keine Gesetze!** Selbst entscheiden, was im konkreten Fall zutrifft ist immer noch erforderlich.
 - ◆ Z. B.: „Tisch **hat** Beine“ → Aggregation
 - ◆ Aber: „Kunde **hat** Konto“ → einfache Assoziation.
 - ◆ Z. B.: „Kind **ist** Mensch“ → Generalisierung
 - ◆ Aber: „Thomas **ist** Kind“ → Instanz!
- Die Entsprechungen aus der Abbott-Tabelle sind **nicht vollständig!** Sie sollten sie sinngemäß ergänzen.
 - ◆ Z.B: „hat“ → „beinhaltet“, „enthält“, „ist Teil von“, ...

Beispiel ▶ Szenario aus der Problembeschreibung

- Jim Smith **ist Kunde** bei einer **großen Bank**.
- Dort **besitzt** er ein **kostenpflichtiges Konto** mit einem **Kontostand** von derzeit **2.000 Euro**.
- Jim Smith **geht** zum **Schalter** und **zahlt** **100,- Euro ein**.
- Am nächsten Tag **betrifft** Jim Smith die **Bank** erneut. Er **geht** zum **Geldautomat** und **hebt** **400,- Euro ab**.

OK, wir haben nun erste Hinweise was Objekte, Methoden, ... sein könnten.

Wie geht's nun weiter? → Diagramme erstellen!

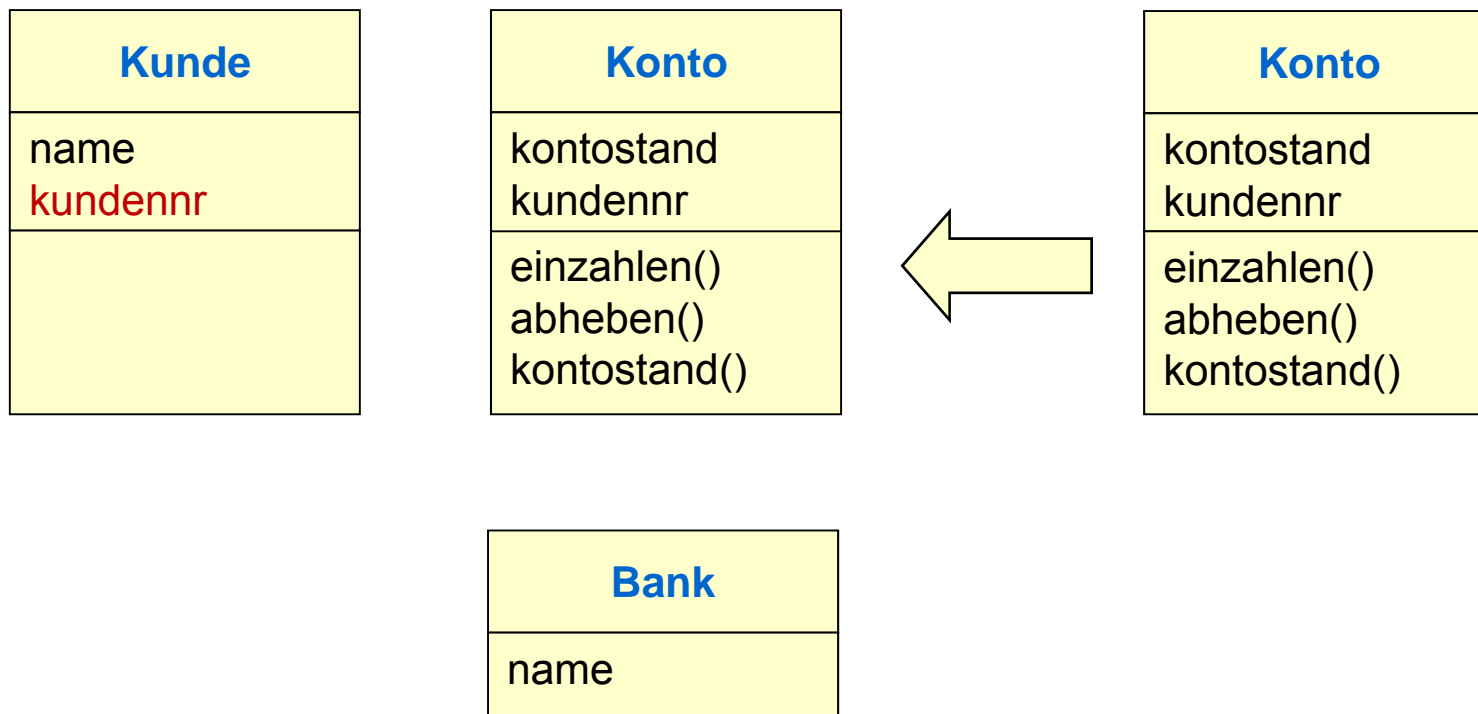
Objektmodellierung ▶ Klassenidentifikation

- Name der Klasse
- Attribute
- Methoden



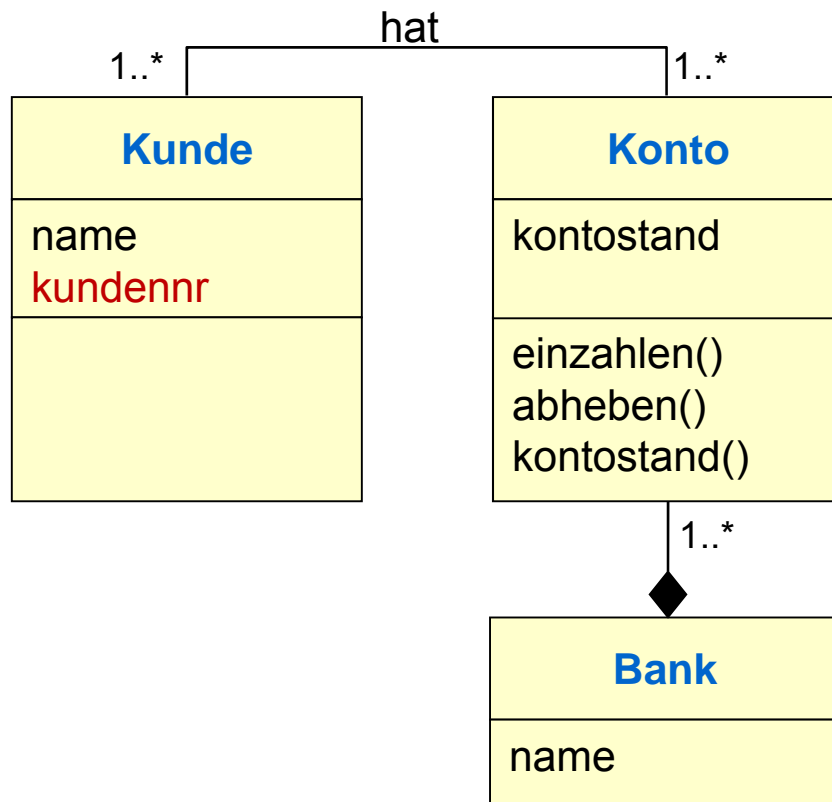
Objektmodellierung ▶ Iteration

- Finde neue Objekte: Kunde, Bank
- Iteriere über Namen, Attribute und Methoden
- Verlagere Daten und Verantwortlichkeiten an die passendste Stelle

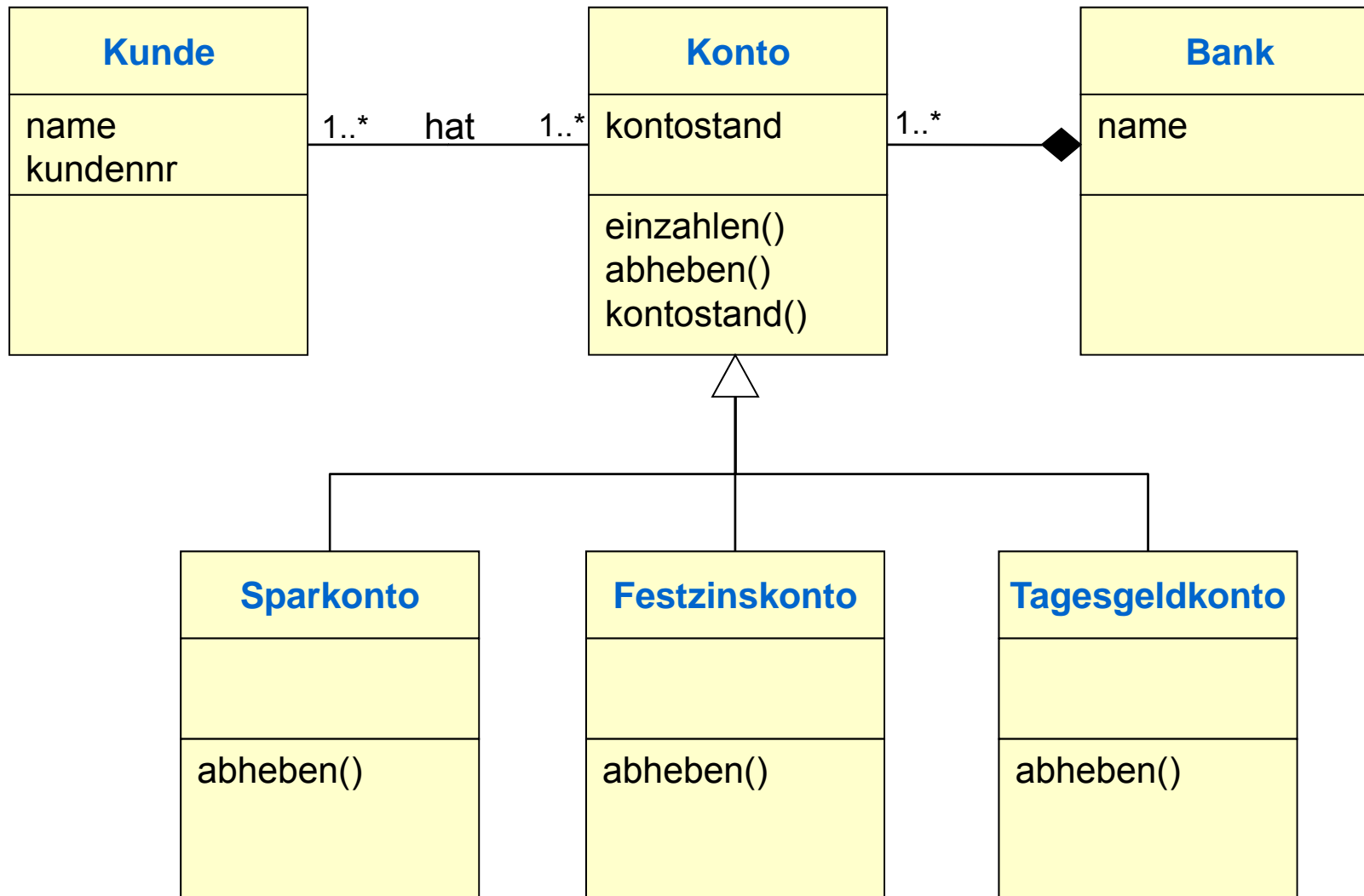


Objektmodellierung ▶ Assoziationen

- Name und Rollen
- Art (Assoziation, Aggregation, Komposition)
- Kardinalitäten



Objektmodellierung ▶ Kategorien finden (Vererbung)



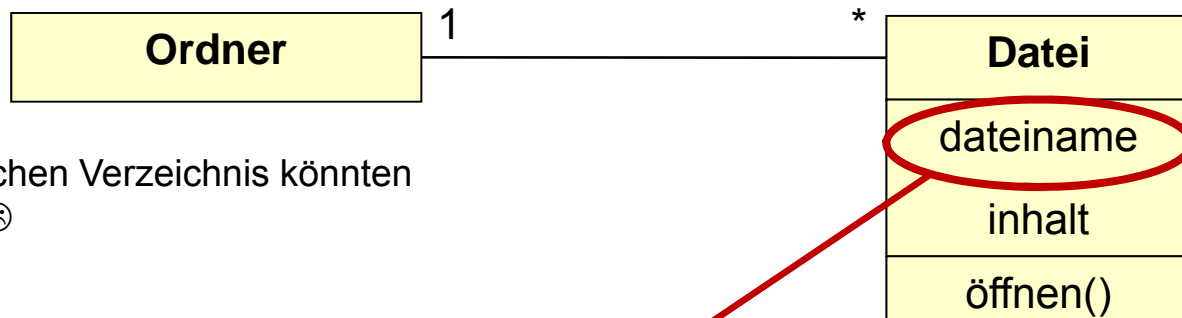
Objektmodellierung ▶ Qualifizierte Assoziation

- Eine „Qualifikation“ (engl. „Qualifier“) präzisiert die Multiplizitätsangaben einer Assoziation
 - ◆ Er wird benutzt, um 1-n Multiplizitäten auf 1-1 Multiplizitäten zu reduzieren
 - ◆ Das entspricht der Indizierung der Elemente am gegenüberliegenden Ende der Assoziation durch „Qualifikations“-Werte

Ohne Qualifikation:

Ein Verzeichnis enthält viele Dateien.

Verschiedene Dateien im gleichen Verzeichnis könnten den gleichen Namen haben! ☹

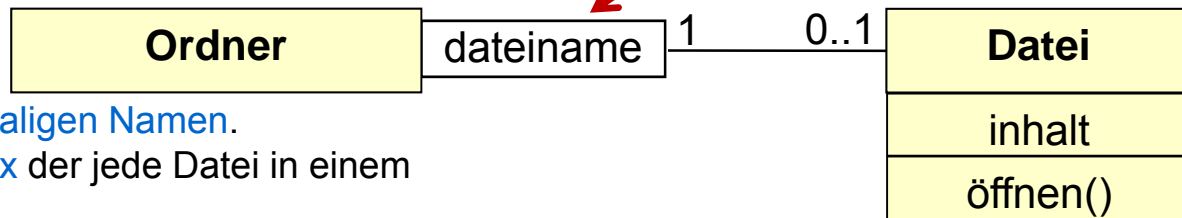


Mit Qualifikation:

Ein Verzeichnis enthält viele

Dateien, jede mit einem **einmaligen Namen**.

Der Dateiname dient als **Index** der jede Datei in einem Ordner eindeutig identifiziert.



Objektmodellierung ▶ Zusammenfassung

- Ableitung eines Objektmodells aus einem Use Case Modell
 - ◆ Identifikation von Objekten / Klassen
 - ◆ Identifikation von Attributen and Operationen
 - ◆ Generalisierung
 - ◆ Identifikation von Assoziationen, Aggregation und Komposition
 - ◆ Reduzierung der Multiplizität mit Hilfe von qualifizierten Assoziationen
- Die besprochenen Techniken sind allgemeiner Natur und können jederzeit eingesetzt werden
 - ◆ Anforderungs-Erhebung → für Domain Object Model
 - ◆ Analyse → für Analyse-Modell
 - ◆ Design → für Design-Modell
- Sie wurden hier vorgestellt, da sie grundlegend sind und hier zum ersten Mal Verwendung finden

Erstellung des Domain Object Model – Ausführliches Beispiel –

Use Case als Ausgangspunkt

Anwendung der Technik von Abbott

Verfeinerung des nach Abbot erstellten Domain Object Model

Beispiel ▶ Ereignisfluss aus Use Case als Ausgangspunkt

- Stellen Sie sich vor, dass Ihre Kundin, eine Großhändlerin, die Softgetränke an Supermärkte liefert, ein System wie folgt beschreibt:

„Die Pfandrückgabemaschine (PRM) wird in Supermärkten aufgestellt um wiederverwendbare Getränkebehälter zurückzunehmen (leere Dosen, Flaschen und Kästen, welche Endbenutzern geliefert werden). Für jeden Behältertyp kann unsere Betreiberin einen individuellen Pfandbetrag einstellen.

Anstelle der Rückgabe von Münzgeld druckt die PRM eine Quittung über die Summe der entstandenen Pfandbeträge. Diese Quittung kann durch einen Bar Code Scanner eingelöst werden.

Die PRM generiert für unsere Betreiberin täglich einen Bericht. Ein Bericht beinhaltet eine Liste aller an diesem Tag zurückgegangenen Behälter, sowie die Tagessumme an ausgezahltem Pfand-Geld.“

Beispiel ▶ Textanalyse nach Abbott

▶ Hervorheben der Textelemente

- Substantive / Nomen / Hauptwörter → Klassen

„Die **Pfandrückgabemaschine** (PRM) wird in **Supermärkten** aufgestellt um wiederverwendbare **Getränkebehälter** zurückzunehmen (leere **Dosen**, **Flaschen** und **Kästen**, welche **Endbenutzern** geliefert werden). Für jeden **Behältertyp** kann unsere **Betreiberin** einen individuellen **Pfandbetrag** einstellen.

Anstelle der **Rückgabe** von **Münzgeld** druckt die **PRM** eine **Quittung** über die **Summe** der entstandenen **Pfandbeträge**. Diese **Quittung** kann durch einen Bar Code **Scanner** eingelöst werden.

Die **PRM** generiert für unsere **Betreiberin** täglich einen **Bericht**. Ein **Bericht** beinhaltet eine **Liste** aller an diesem **Tag** zurückgegangenen **Behälter**, sowie die **Tagessumme** an ausgezahltem **Pfand-Geld**.“

Beispiel ▶ Textanalyse nach Abbott

▶ Klassen extrahieren

- Substantive / Nomen / Hauptwörter → Klassen

„Die Pfandrückgabemaschine (PRM) wird in Supermärkten aufgestellt um wiederverwendbare Getränkebehälter zurückzunehmen (leere Dosen, Flaschen und Kästen, welche Endbenutzern geliefert werden). Für jeden Behältertyp kann unsere Betreiberin einen individuellen Pfandbetrag einstellen.“

Betreiberin

Supermarkt

Pfandrückgabemaschine

Getränkebehälter

Behältertyp

Pfandbetrag

Dose

Flasche

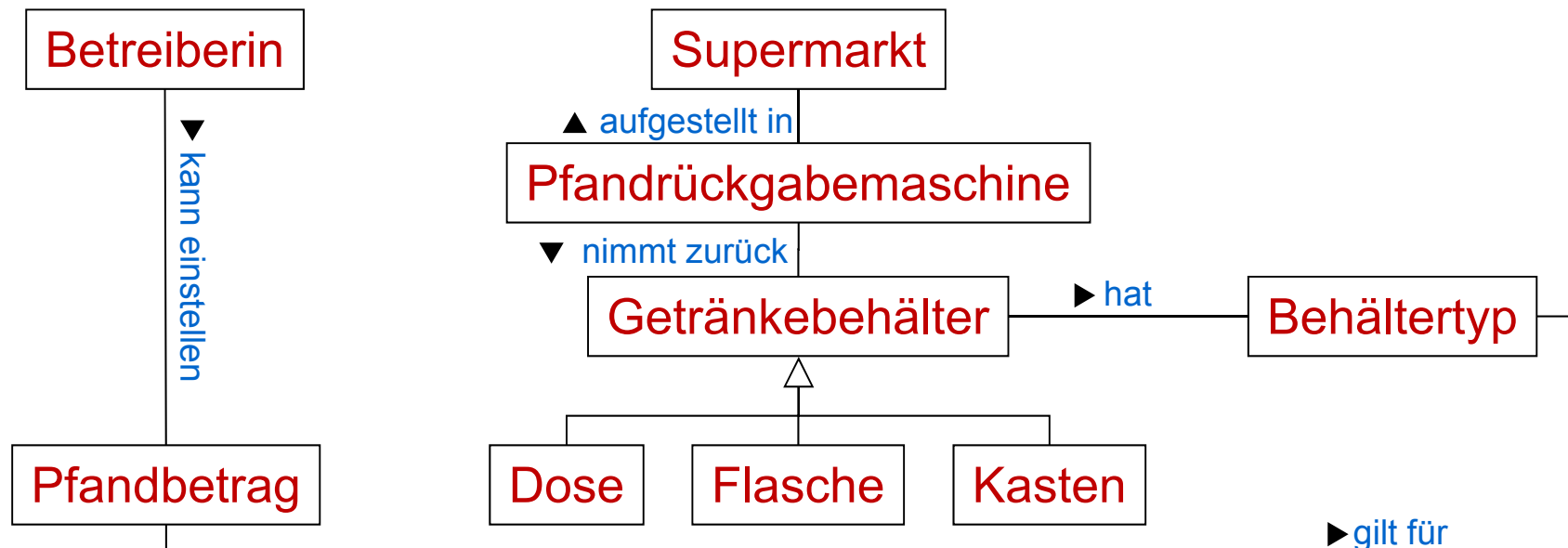
Kasten

Beispiel ▶ Textanalyse nach Abbott

▶ Assoziationen extrahieren

- Substantive + Verben + Attribute → Klassen + Assoziationen

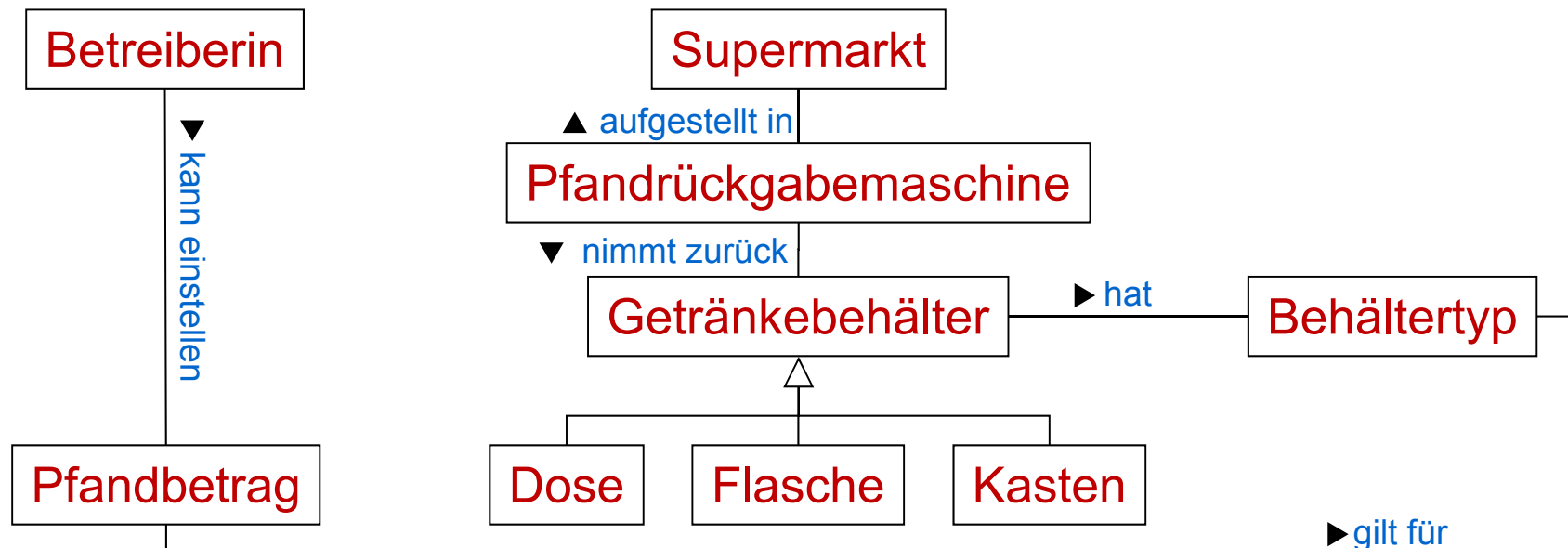
„Die Pfandrückgabemaschine (PRM) wird in Supermärkten aufgestellt um wiederverwendbare Getränkebehälter zurückzunehmen (leere Dosen, Flaschen und Kästen, welche Endbenutzern geliefert werden). Für jeden Behältertyp kann unsere Betreiberin einen individuellen Pfandbetrag einstellen.“



Beispiel ▶ Schematisch erstelltes Domain Object Model überdenken (1)

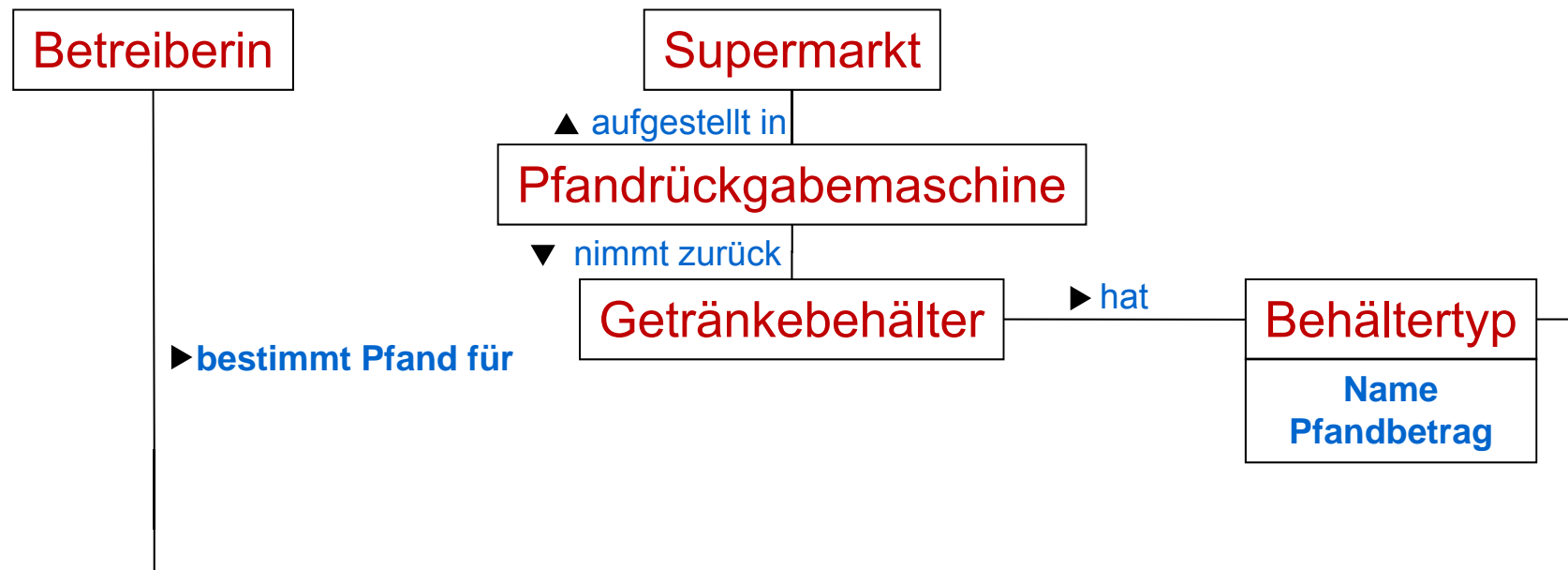
- Vererbungshierarchie überdenken
 - ◆ Dosen, Flaschen und Kästen sind doch eigentlich Behältertypen!
 - ◆ Die Klasse „Behältertyp“ oder die Unterklassen von „Getränkebehälter“ sind redundant.
- Entscheidungshilfen
 - ◆ Welche Klassen besitzen **kein eigenes Verhalten**?
 - ◆ Welche Klassen liefern als einzige Information ihre Klassenzugehörigkeit?

Hierzu sollten Sie unbedingt erst Methoden identifizieren! Hier wurde aus Platzgründen diesem Schritt vorgegriffen.



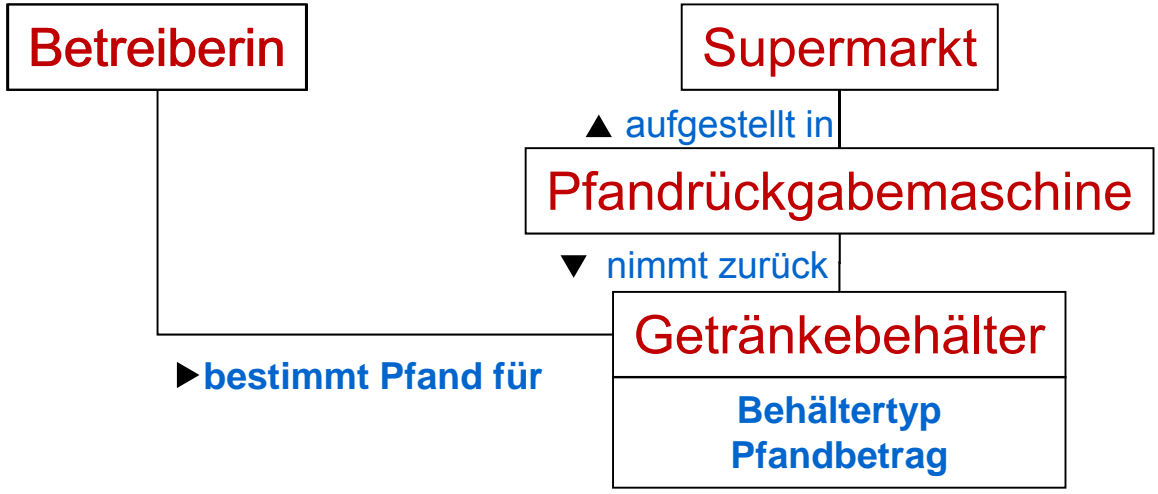
Beispiel ▶ Schematisch erstelltes Domain Object Model überdenken (2)

- Korrektur-Maßnahmen: Klasse zu Attribut konvertieren, wenn
 - ◆ sie kein eigenes Verhalten besitzt → Pfandbetrag
 - ◆ sie als einzige Information ihre Klassenzugehörigkeit liefert → Dose, ...
- Anwendung des Prinzips in unserem Beispiel
 - ◆ Pfandbetrag → Attribut von „Behälterttyp“
 - ◆ Dose, Flasche, Kasten → Werte des „Name“-Attributs von „Behälterttyp“



Beispiel ▶ Schematisch erstelltes Domain Object Model überdenken (3)

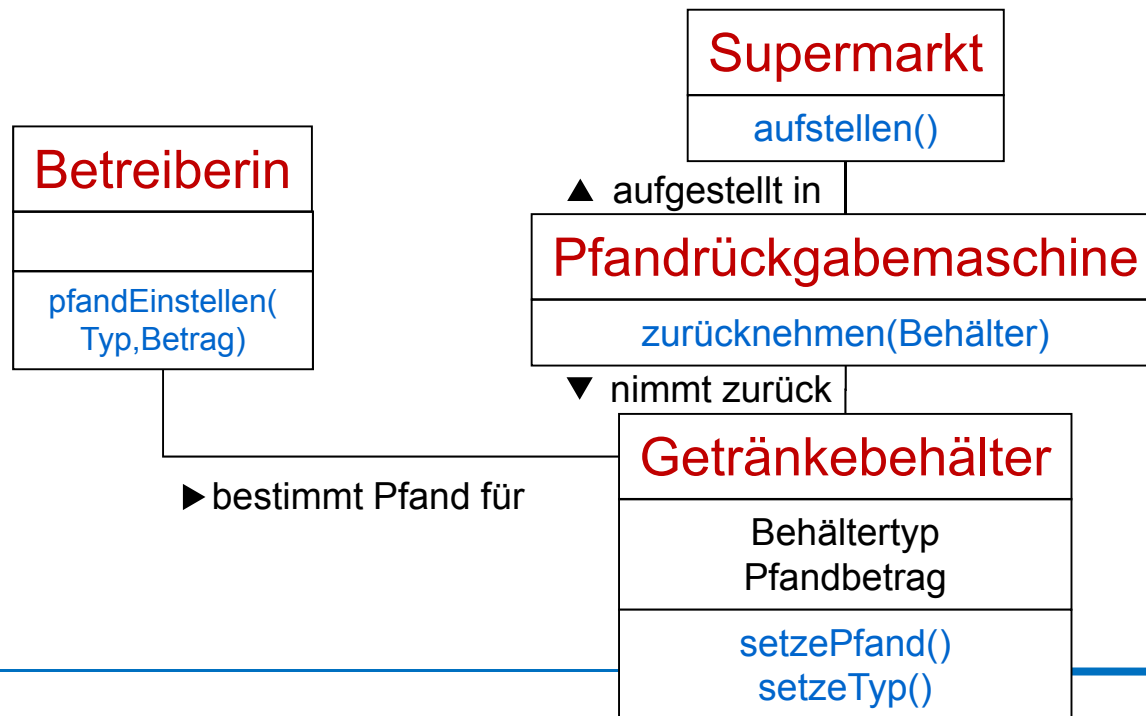
- Korrektur-Maßnahmen: „Inline Class“
 - ◆ Überflüssige Klasse (→ „Behältertyp“) identifizieren und ihre Elemente in eine per Assoziation verbundene Nachbarklasse einbetten
- Anwendung des Prinzips in unserem Beispiel
 - ◆ Die Attribute „Name“ und „Pfandbetrag“ wandern in „Getränkebehälter“
 - ◆ „Name“ wird in „Behältertyp“ umbenannt
 - ◆ Die Klasse „Behältertyp“ wird eliminiert



Beispiel ▶ Fortsetzung der Textanalyse nach Abbott ▶ Methoden extrahieren

- Verben → Methoden

„Die Pfandrückgabemaschine (PRM) wird in Supermärkten aufgestellt um wiederverwendbare Getränkebehälter zurückzunehmen (leere Dosen, Flaschen und Kästen, welche Endbenutzern geliefert werden). Für jeden Behältertyp kann unsere Betreiberin einen individuellen Pfandbetrag einstellen.“



Beispiel ▶ Fortsetzung der Textanalyse nach Abbott ▶ Alles Weitere

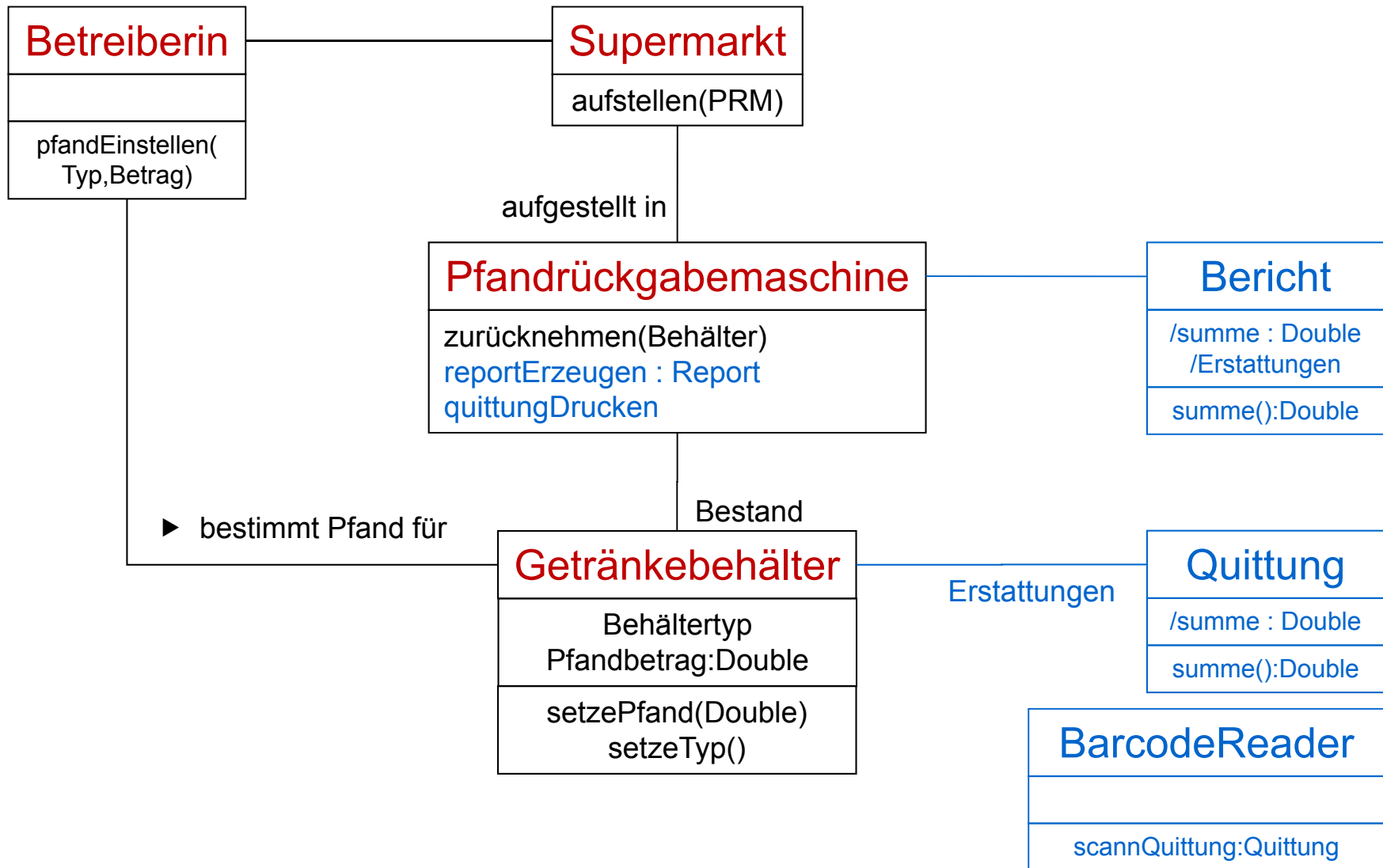
- Fortsetzen für weitere Sprachelemente (Z.B. Adjektive)
- Anwendung des Ganzen auf den Rest der Aufgabenstellung

„Die Pfandrückgabemaschine (PRM) wird in Supermärkten aufgestellt um wiederverwendbare Getränkebehälter zurückzunehmen (leere Dosen, Flaschen und Kästen, welche Endbenutzern geliefert werden). Für jeden Behältertyp kann unsere Betreiberin einen individuellen Pfandbetrag einstellen.

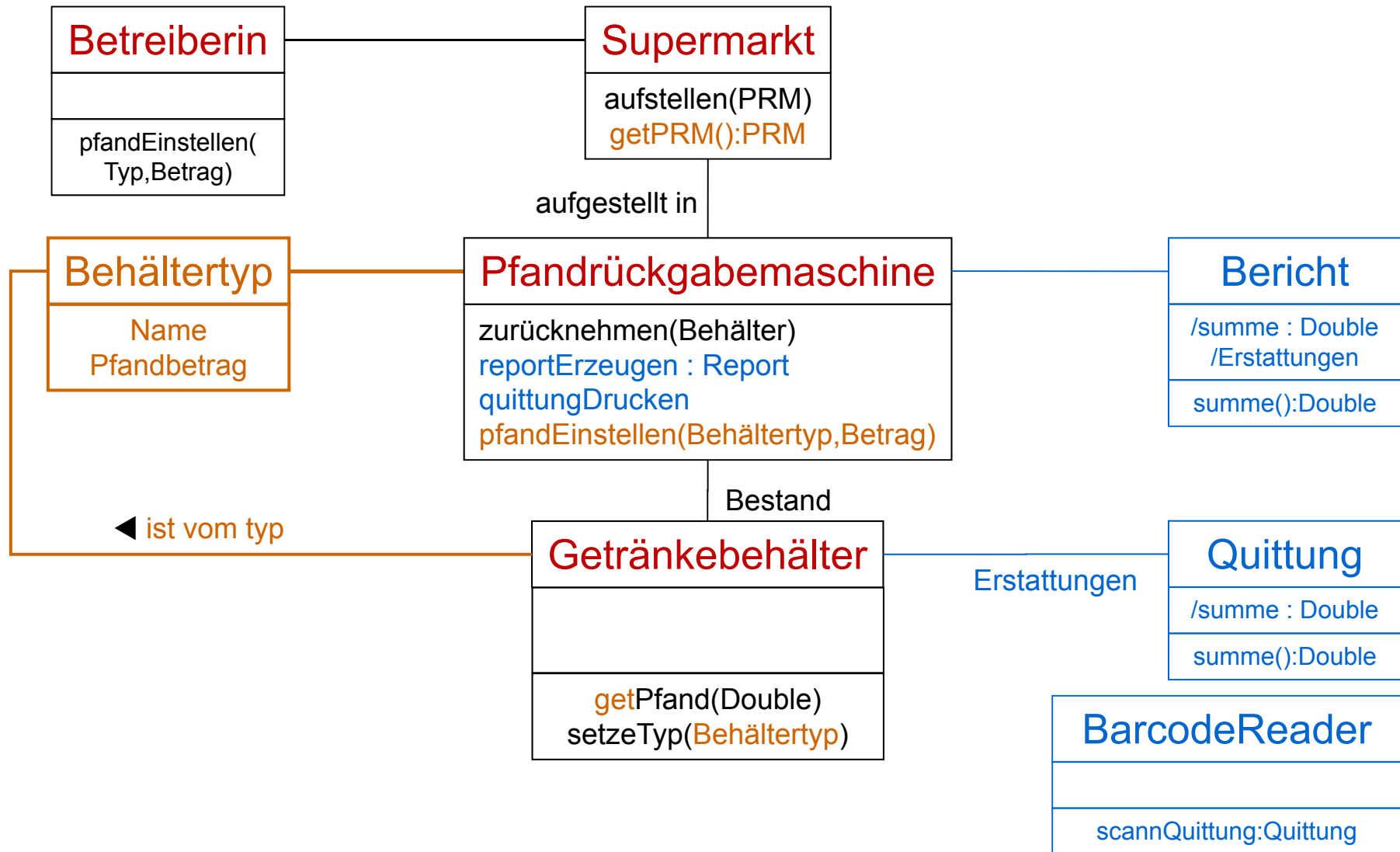
Anstelle der Rückgabe von Münzgeld druckt die PRM eine Quittung über die Summe der entstandenen Pfandbeträge. Diese Quittung kann durch einen Bar Code Scanner eingelöst werden.

Die PRM generiert für unsere Betreiberin täglich einen Bericht. Ein Bericht beinhaltet eine Liste aller an diesem Tag zurückgegangenen Behälter, sowie die Tagessumme an ausgezahltem Pfand-Geld.“

Beispiel ▶ Endergebnis für gesamten Text

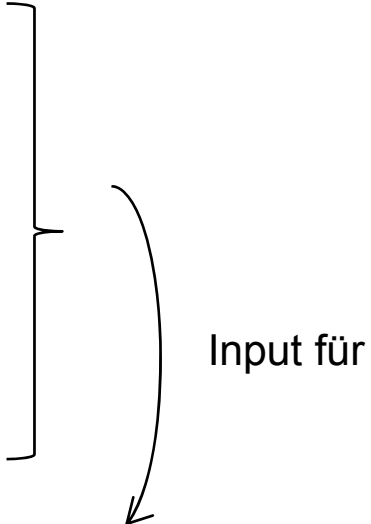


Beispiel ▶ Endergebnis weitergedacht



5.1.5 Dynamische Modellierung von Anwendungsfällen

Anforderungserhebung ▶ Gesamtüberblick

- ✓ 1. Analysiere die Problembeschreibung
 - ◆ Identifiziere funktionale Anforderungen
 - ◆ Identifiziere nichtfunktionale Anforderungen
 - ◆ Identifiziere Nebenbedingungen (Pseudo-Anforderungen)
 - ✓ 2. Entwickle das funktionale Modell
 - ◆ Entwickle Use Cases zur Illustration der funktionalen Anforderungen
 - ✓ 3. Entwickle das Objektmodell
 - ◆ Entwickle Klassendiagramme, die die Struktur des Systems beschreiben
 - 4. Entwickle das dynamische Modell
 - ◆ Aktivitätsdiagramme für Geschäftsprozesse
 - ◆ Interaktionsdiagramme für das Zusammenspiel von Objekten
 - ◆ Zustandsdiagramme für die internen Abläufe von Objekte mit interessantem Verhalten
- 
- Input für

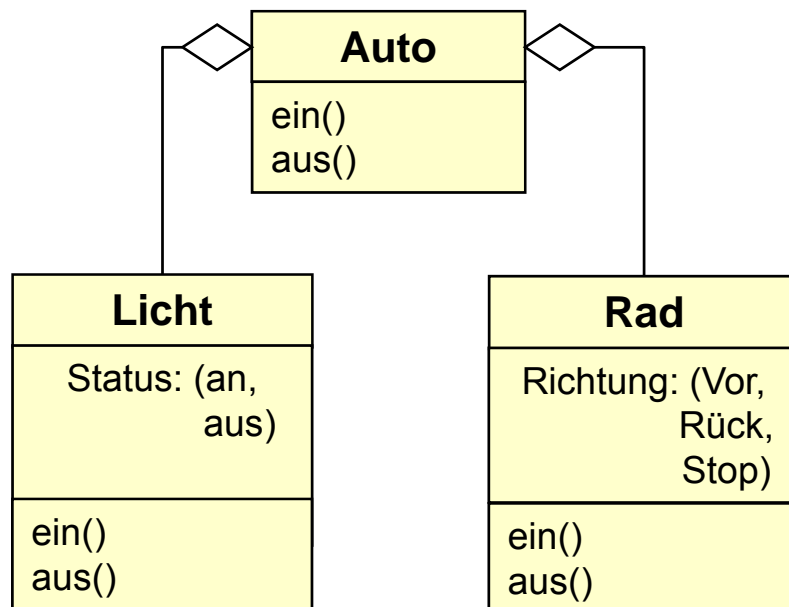
Spielzeugauto ▶ Problembeschreibung

- Strom wird angeschaltet
→ Auto fährt **vorwärts**
● Scheinwerfer leuchtet
- Strom wird ausgeschaltet
● STOP Auto hält an
● Scheinwerfer geht aus
- Strom wird angeschaltet
● Scheinwerfer leuchtet
- Strom wird ausgeschaltet
● Scheinwerfer geht aus

- Strom wird angeschaltet
← Auto fährt **rückwärts**
● Scheinwerfer leuchtet
- Strom wird ausgeschaltet
● STOP Auto hält an
● Scheinwerfer geht aus
- Strom wird angeschaltet
● Scheinwerfer leuchtet
- Strom wird ausgeschaltet
● Scheinwerfer geht aus

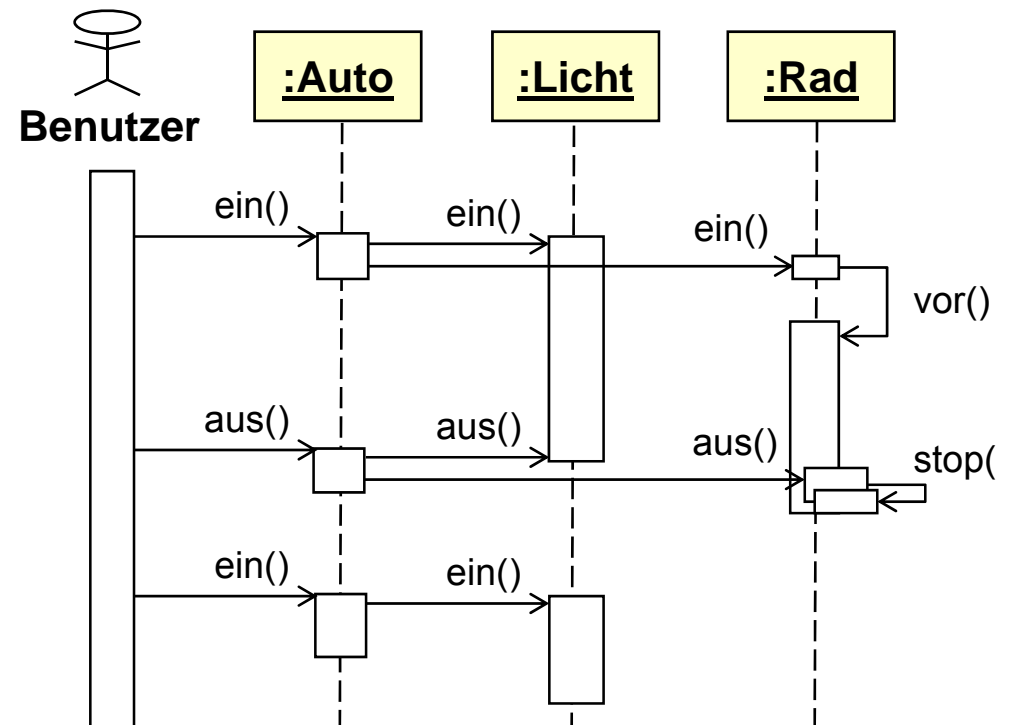
Spielzeugauto ▶ Verschiedene Modelle

Klassendiagramm



- Sagt nichts über das Verhalten
- Jedes "ein()" tut etwas anderes!

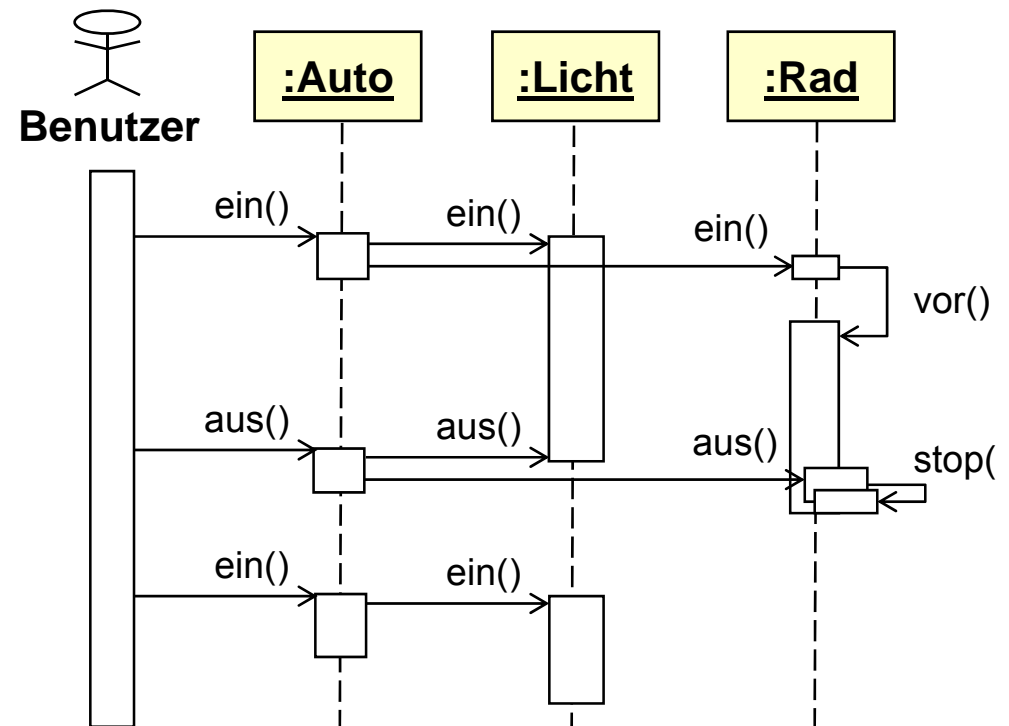
Sequenzdiagramm



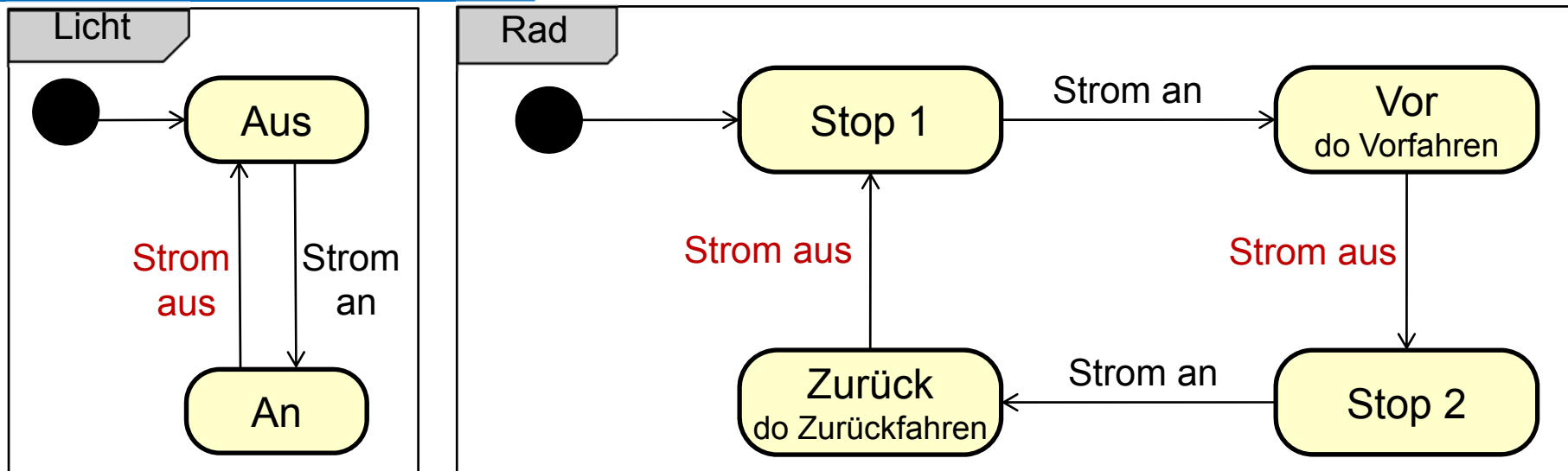
- Modelliert asynchrones Verhalten
- ... allerdings nur einen Ausschnitt

Spielzeugauto ▶ Bewertung des Sequenzdiagramms

- Drückt nicht aus, dass sich die Vorgänge beliebig wiederholen können
 - ◆ keine feste Wiederholungszahl
 - ◆ keine feste Endbedingung
- Dass beim nächsten Einschalten des Stroms nach dem Stop die Räder still stehen wurde im Auto modelliert
 - ◆ Das Auto schickt in diesem Fall keine Nachricht an die Räder
 - Das Auto ist für die Modellierung des Verhaltens der Räder mit zuständig
 - ◆ Ist das gut oder schlecht?



Spielzeugauto ▶ Zustandsdiagramm



- Hier wird das Verhalten von Licht und Rädern komplett und kompakt ausgedrückt – einschließlich der Unabhängigkeit der beiden Aspekte
 - ◆ „Die Moral von der Geschichte“ ▶ Oft ist die Wahl der Wahl der richtigen Notation schon der wichtigste Teil der Lösung geleistet.
- Denksport
 - ◆ Welche Annahme steckt im Automat für die Räder?
 - ⇒ Tip: siehe Problembeschreibung und Bewertung des Sequenzdiagramms
 - ◆ Wie müsste der Automat verändert werden, wenn sie nicht mehr gilt?

5.1.6 Zusammenfassung

Was Sie sich zur Anforderungserhebung mindestens merken sollten

Anforderungserhebung ▶ Wichtigste Konzepte

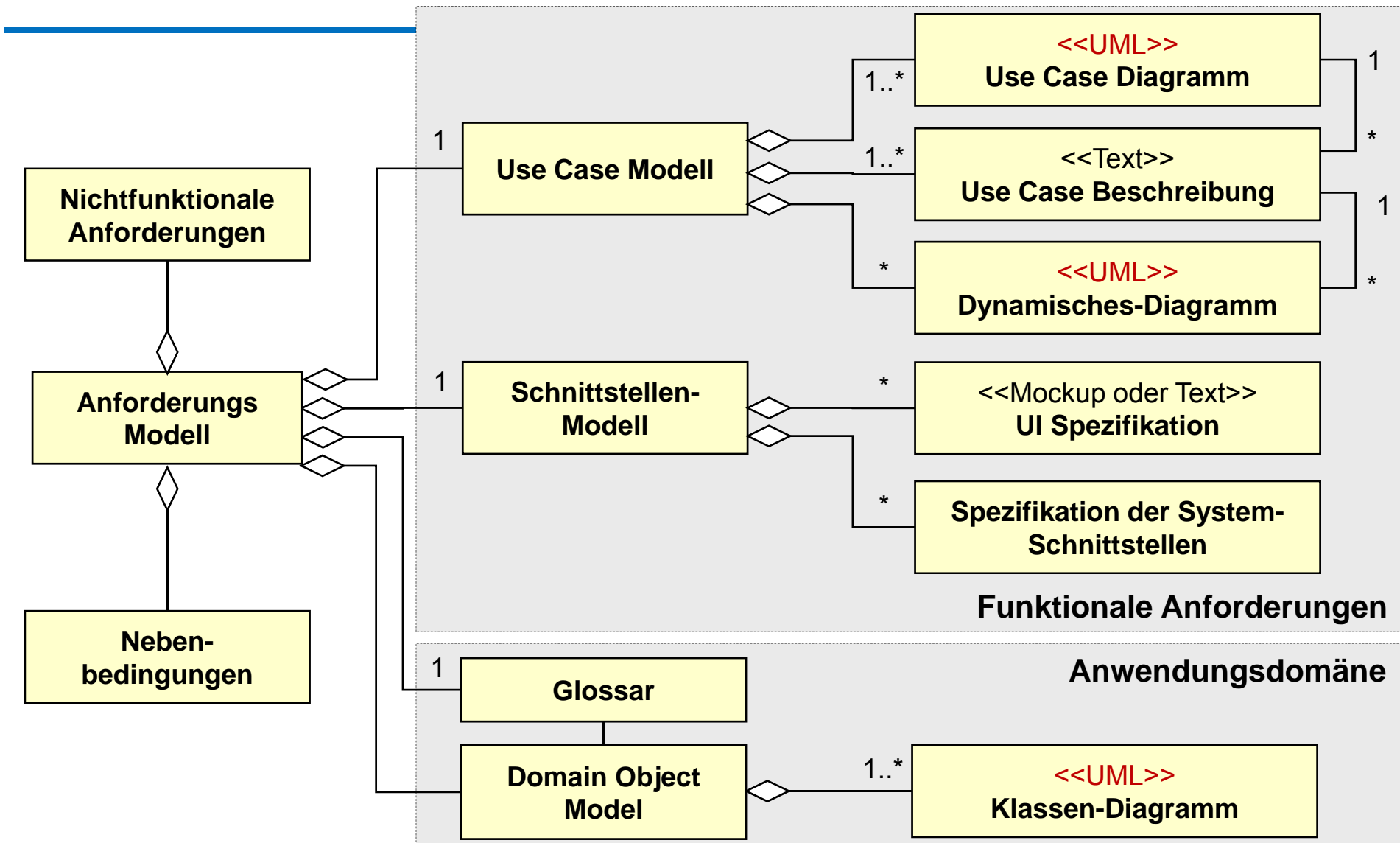
- Szenarien
 - ◆ Großartiger Weg zur Kommunikation mit dem Kunden
 - ◆ As-Is Szenarien, Visionary Szenarien, Evaluation Szenarien, Training Szenarien
- Use Cases
 - ◆ Abstraktion von Szenarien
 - ◆ Use Case Modell erfasst vor allem funktionale Anforderungen
- Domain Object Model
 - ◆ Klassendiagramme erfassen Anwendungsdomäne in strukturierter Form
- Verfahren von Abbott
 - ◆ Textanalyse als Hilfe zur Objektidentifikation
 - ◆ Strukturierung / Verfeinerung des Objektmodells anhand simpler Prinzipien

Anforderungserhebung ▶ Aktivitäten

- Aktivitäten der Anforderungserhebung
 - ◆ Erhebung von Szenarien
 - ◆ Abstraktion und Strukturierung von Use Cases
 - ◆ Identifikation beteiligter Objekte (Domain Object Model)
 - ◆ Spezifikation von elementarem Verhalten

- Sie dienen der
 - ◆ Festlegung der Intention und des Umfangs eines Systems
 - ◆ Erfassung der Anforderungen aus Anwender-Sicht in einer für den Anwender noch nachvollziehbaren Form

Anforderungserhebung ▶ Produkte



Artefakte des Anforderungs-Modells

- Glossar
 - ◆ Stichwortverzeichnis der Anwendungsdomäne
 - ◆ Begriffe sind durch freien Text definiert
- Domain Object Model
 - ◆ Erfassung der wichtigsten Konzepte des Glossars und ihrer Beziehungen durch ein Klassendiagramm / Klassendiagramme
- Interface Model
 - ◆ Mockups (= mit Papier, Farben, etc. simulierte Benutzeroberflächen)
 - ◆ System-Schnittstellen-Beschreibung (textuell oder Klassendiagramm)
- Use Case Modell
 - ◆ Beschreibung aller Anwendungsfälle durch ein oder mehrere Diagramme
 - ◆ Beschreibung jedes Anwendungsfalls durch strukturierten Text
 - ◆ Evtl. Beschreibung der Abläufe komplexer Anwendungsfälle durch dynamische Diagramme
 - ◆ Evtl. zum zusätzlichen Verständnis komplexer Geschäftsprozesse in die die Anwendungsfälle eingebettet sind, Beschreibung der Geschäftsprozesse durch dynamische Diagramme.

Hier geht's weiter am Di, 16.11.2010

5.2 Anforderungsanalyse (Requirements Analysis)

- 5.2.1 Das Analyse-Modell
- 5.2.2 Objektmodellierung im Analyseworkflow
 - 5.2.3 Dynamische Modellierung
 - 5.2.4 Der Analyse-Workflow, Beispiele
 - 5.2.5 Konsolidierung der Analyse

Inhaltsübersicht

- 1) Das Analyse-Modell
 - ◆ Unterschiede Use Case Modell ↔ Analysemodell
- 2) Objektmodellierung im Analyse-Workflow
 - ◆ Entities, Boundaries und Controller
- 3) Dynamische Modellierung
 - ◆ Von Use Cases zum Objektverhalten
- 4) Der Analyse-Workflow
 - ◆ Beispiel: Von der Objektstruktur zum Objektverhalten
 - ◆ Beispiel: Kompletter Analyse-Workflow
- 5) Konsolidierung der Analyseergebnisse
 - ◆ Redundanzen, Mehrdeutigkeiten, Widersprüche, ... eliminieren

Aktivitäten bei der Anforderungserhebung und -analyse

Anforderungserhebung

- Identifiziere Akteure
- Identifiziere Szenarien
- Identifiziere Use Cases
- Verfeinere Use Cases
- Identif. Beziehungen zwischen Use Cases
- Identif. nichtfunktionale Anforderungen
- Identifiziere Nebenbedingungen
- Identifiziere beteiligte Objekte
- Erstelle Glossar

Anforderungsanalyse

- Identifiziere Objekte/Klassen
 - ◆ boundary, controller, entity
- Identifiziere Operationen und Methoden
- Identifiziere Assoziationen
- Identifiziere Attribute
- Modelliere Objektinteraktionen
 - ◆ Kommunikationsdiagramme
 - ◆ Sequenzdiagramme
- Modelliere nichttriviales internes Verhalten
 - ◆ Zustandsdiagramme

Use Cases in Objektstruktur übersetzen

Use Cases in Objektverhalten übersetzen

Von beobachteter

Mehrdeutigkeit, Unkorrektheit, Unvollständigkeit, Inkonsistenz, Undurchführbarkeit
zur Verfeinerung der Anforderungen

5.2.1 Das Analyse-Modell

Abgrenzung Anforderungserhebung zu Anforderungsanalyse

Use Case vs. Analyse Modell (1)

Use Case Modell

- in der Terminologie der Anwender verfasst

- externe Sicht des Systems (was tut es?)

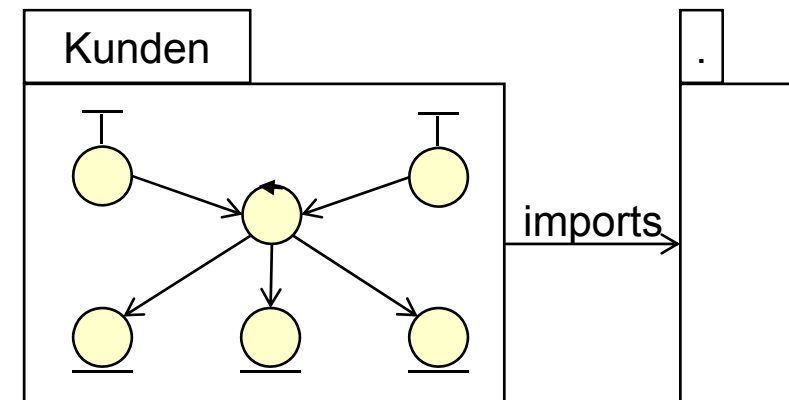
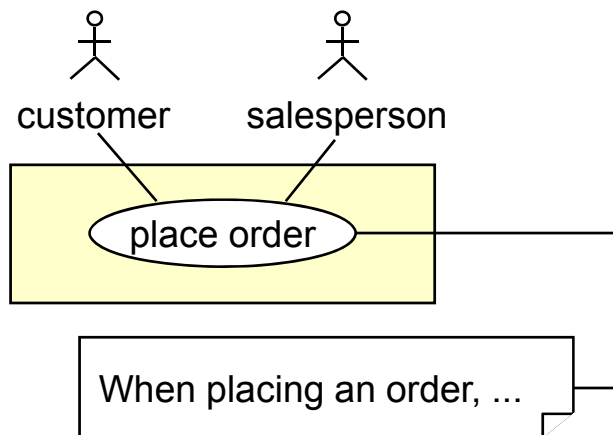
- in „use cases“ strukturiert

Analyse Modell

- in der Terminologie der Entwickler verfasst

- interne Sicht des Systems (wie tut es das?)

- in Module und Klassen-Stereotypen strukturiert



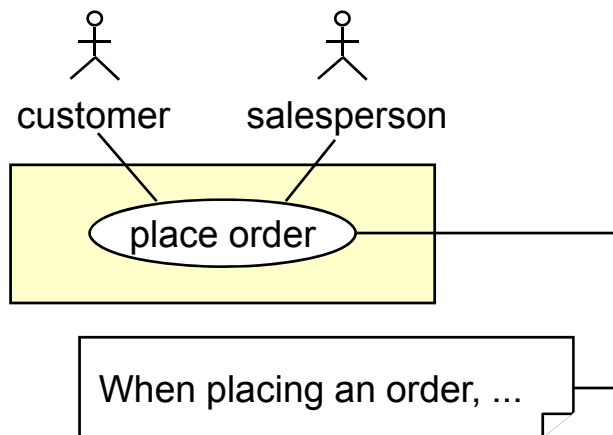
Use Case vs. Analyse Modell (2)

Use Case Modell

- dient vorwiegend als Kontrakt zwischen Auftraggeber und Entwickler hinsichtlich der Anforderungen an das System

- kann redundante / inkonsistente Anforderungen enthalten

- definiert Use Cases, die im Analyse-Modell weiter vertieft werden

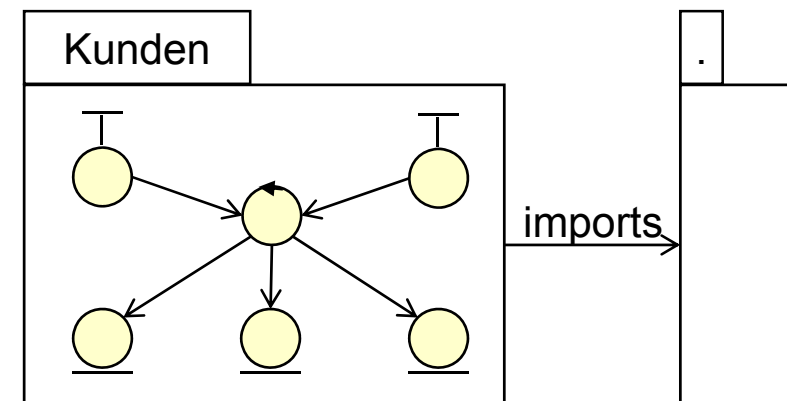


Analyse Modell

- dient den Entwicklern zum Verständnis der System-Struktur

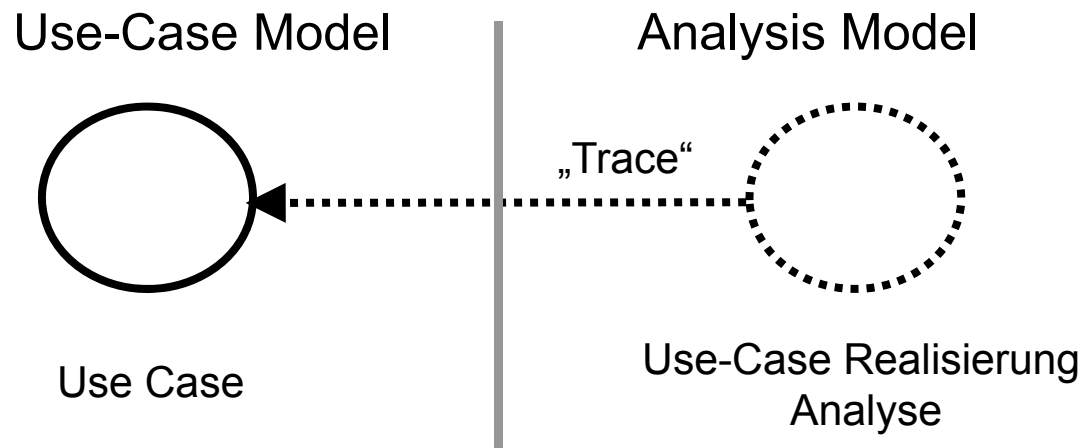
- darf keine Redundanzen / Inkonsistenzen mehr enthalten

- definiert die Use-Case-Realisierungen



Use-Case Realisierung – Analyse

- Beschreibt die Umsetzung eines Use-Case im Analyse-Modell
 - ◆ auf hoher Abstraktionsebene
 - ◆ enthält keine Implementierungsdetails (verwendete Bibliotheken, Application Server, ...)
- Besteht aus
 - ◆ Textuelle Beschreibung des Ereignisflusses
 - ◆ Besondere Anforderungen
 - ◆ **Klassendiagrammen** ← Verfeinerung des DOM
 - ◆ **Interaktions- und Aktivitätsdiagrammen** ← Verfeinert oder neu



5.2.2 Objektmodellierung im Analyseworkflow

Die Besonderheit der Objektmodellierung im Analyseworkflow
→ Die Aufteilung in Boundary, Controller und Entity Stereotypen

Analysetyp (Klasse oder Interface)

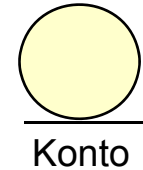
- Abstrahiert eine oder mehrere Konzepte und / oder Subsysteme
 - ◆ Definiert konzeptuelle Attribute die evtl. später zu eigenen Klassen werden
 - ◆ Beschreibt noch relativ wenige Operationen
- Realisiert essentielle funktionale Anforderungen
 - ◆ Nicht-funktionale Anforderungen werden erst im Systemdesign behandelt

Kategorien von Analysetypen

Ein Analysetyp ist stets einer der drei folgenden Stereotypen:

- **Boundary Typ** = Schnittstelle zu Akteur
- **Control Typ** = Ereignisfluss eines Use Cases
- **Entity Typ** = Anwendungskonzept („Business object“)

Analysetyp ▶ Entity



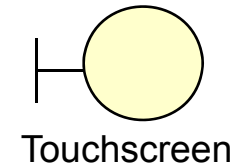
- Repräsentiert ein für das Anwendungsgebiet relevantes Konzept („Business object“)
 - ◆ Heuristik ▶ Jeder Typ im Domain Object Model ist ein Entity-Kandidat
 - ◆ Heuristik ▶ Jeder Begriff im Glossar ist ein Entity-Kandidat
- Hat oft Use-Case-übergreifende Bedeutung
 - ◆ Heuristik ▶ In verschiedenen Use Cases wiederkehrende Substantive identifizieren

Lebensdauer

- Entspricht oft den persistenten Informationen der Anwendung
 - ◆ Das ergibt sich aus der Use Case übergreifenden Bedeutung
 - ◆ Entities sind aber keine reinen Daten-Klassen
 - ◆ Sie können komplexes Verhalten besitzen (z.B. Zinsberechnung)

Analysetyp ▶ Boundary

«boundary»
Touchscreen

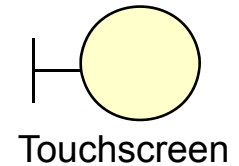


- Beschreibt Interaktionen zwischen dem System und den Akteuren
 - ◆ Heuristik ▶ Jeder Akteur sollte **nur** mit Boundaries verknüpft sein
 - ◆ Konsistenzcheck ▶ Jedes Boundary sollte mit **mindestens einem** Akteur verknüpft sein
- Typische Boundaries
 - ◆ Dateneingabeformulare und Fenster: NotfallberichtBoundary
 - ◆ Fragen und Nachrichten an den Nutzer: BestätigungsBoundary

Benennung

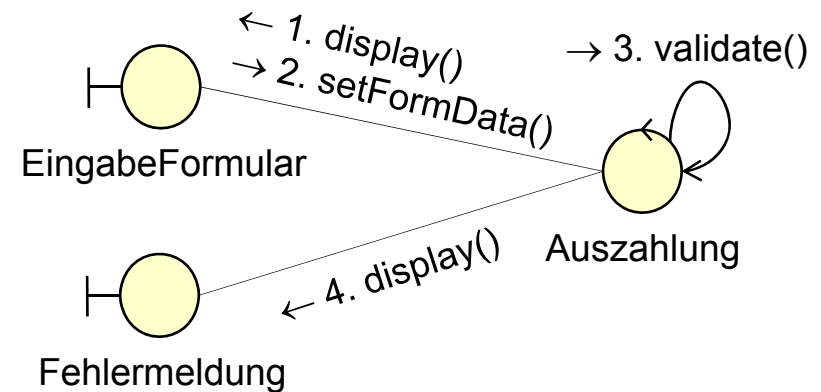
- Verwende Termini der Anwendungsdomäne plus Suffix Boundary zwecks Unterscheidung von Entities
 - ◆ Notfallbericht bezeichnet ein Entity das den Bericht darstellt
 - ◆ NotfallberichtBoundary bezeichnet das Eingabeformular dafür
- Verwende keine Implementierungsbegriffe um keine voreiligen Implementierungsentscheidungen zu suggerieren
 - ◆ Nicht NotfallberichtTabelle oder BestätigungsPopUp

Analysetyp ▶ Boundary ▶ Granularität



- So fein wie nötig ▶ Einheiten trennen, die im Ereignisfluss als unterschiedlich erkennbar sein müssen.

- ◆ EingabeformularBoundary von EingabefehlermeldungBoundary trennen, um modellieren zu können, dass letzteres vom Controller als Reaktion auf eine Fehleingabe in ersterem geöffnet wird

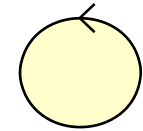


- So grob wie möglich ▶ Möglichst große logische Einheiten zusammenfassen.

- ◆ Nicht jeder Knopf ist ein Boundary!
- ◆ Eine zu feine Aufteilung würde
 - ⇒ die Modellierung unnötig komplex machen (sich in Details verlieren)
 - ⇒ dauernde Änderungen des Analysemodells für jede Änderung der graphischen Oberfläche nach sich ziehen

Analysetyp ▶ Controller

«control»
Abheben



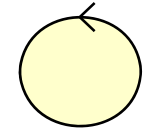
Abheben

- Kapselt den Ereignisfluss eines Use-Case
 - ◆ Vermittlungsstelle zwischen Entities und Boundaries
 - ◆ Komplexe Berechnungen oder logische Zusammenhänge, die keiner Entity zugeordnet werden können
- Typische Aufgaben von Kontrollobjekten
 - ◆ Ablaufsteuerung in Formularen (z.B. „wizards“)
 - ◆ Command History und Undo-Funktionalität
 - ◆ Verteilung und Weiterleitung von Informationen

Benennung

- Name des Use-Case plus Suffix Control, Controller oder Command

Analysetyp ▶ Controller ▶ Granularität



Abheben

- **Standard** ▶ Ein Kontrollobjekt pro Use Case
- **Ausnahme** ▶ Mehr als ein Kontrollobjekt pro Use Case
 - ◆ um problemgebundene physikalische Verteilung auszudrücken
 - ⇒ NotfallberichtControl ▶ im Einsatzfahrzeug
 - ⇒ NotfallerkennungControl ▶ in der Notfallzentrale
 - ◆ oder für sehr komplexe Use Cases

Lebensdauer

- Die Lebensdauer eines Kontrollobjektes sollte der des zugehörigen Use Case entsprechen

Warum genau diese drei Stereotypen?

Die Trennung der drei Kategorien ermöglicht

- **Stabilität** ▶ Modelle, die **weniger änderungsanfällig** sind
 - ◆ Die Grenzen eines System (boundaries) ändern sich häufig
 - ⇒ Darstellung auf zeilenorientiertem Terminal, im Webbrowser, ...
 - ◆ Die Geschäftsabläufe (controller) ändern sich häufig
 - ⇒ Ablauf der Kreditvergabe, Kreditwürdigkeitskriterien, ...
 - ◆ Jede dieser Änderungen soll der Rest des Systems nicht beeinflussen
 - ⇒ Insbesondere sollte die Anwendungsdomäne (entities) möglichst stabil bleiben
 - ⇒ Konten, Kredite, Zinsen, Tilgung, etc.
- **Flexibilität** ▶ Modelle die **flexibler kombinierbar** sind
 - ◆ Verschiedene Geschäftsabläufe und Oberflächen sollten möglichst frei miteinander kombinierbar sein
 - ⇒ Beispiel: Kreditwürdigkeitsprüfung via Webbrowser oder Java-Oberfläche

Herkunft ▶ MVC Framework in Smalltalk 76

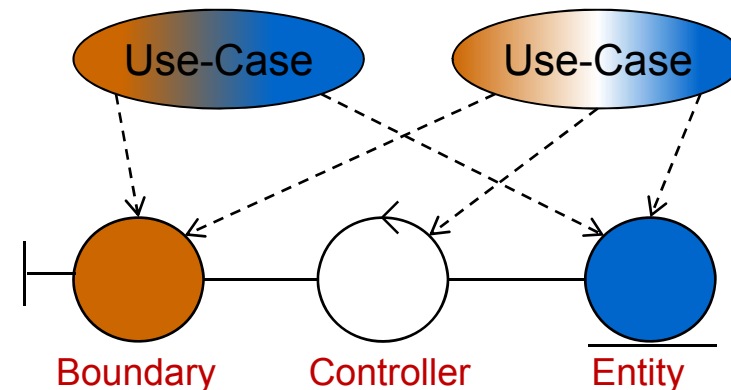
- „Entity, Boundary, Controller“ = „Model, View, Controller“ (MVC)

Analyse-Objektmodell

- Weiterentwicklung des "Domain Object Model"
 - ◆ Entity-Typen ▶ Aus DOM übernommen (evtl. auch ein paar Neue)
 - ◆ Kontroll- und Boundary-Typen ▶ Während der Analyse hinzugefügt

Aufteilung von Use-Cases auf Typen im Analyse-Objektmodell

- Jeder Use-Case verteilt sich auf mehrere Analysetypen
 - ◆ Boundary, Controller, Entity
- Gleicher Analysetyp kann Aspekte mehrerer Use Cases beinhalten

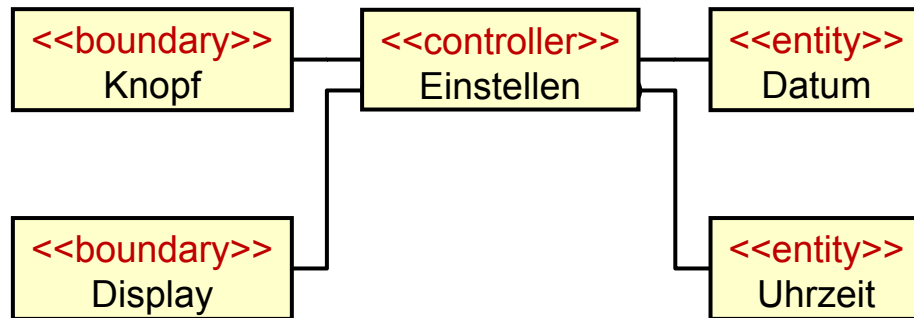


Layout von Analyse-Klassendiagrammen

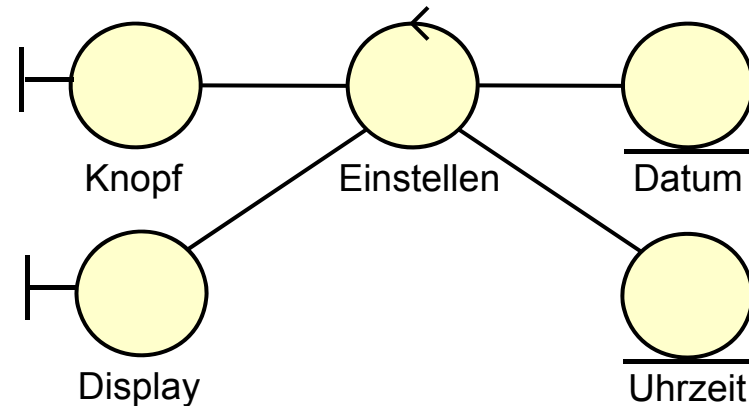
- Boundaries links, Controller in der Mitte, Entities rechts

Analyse-Objektmodell „Digitaluhr“

Textuelle Notation*



Graphische Notation*



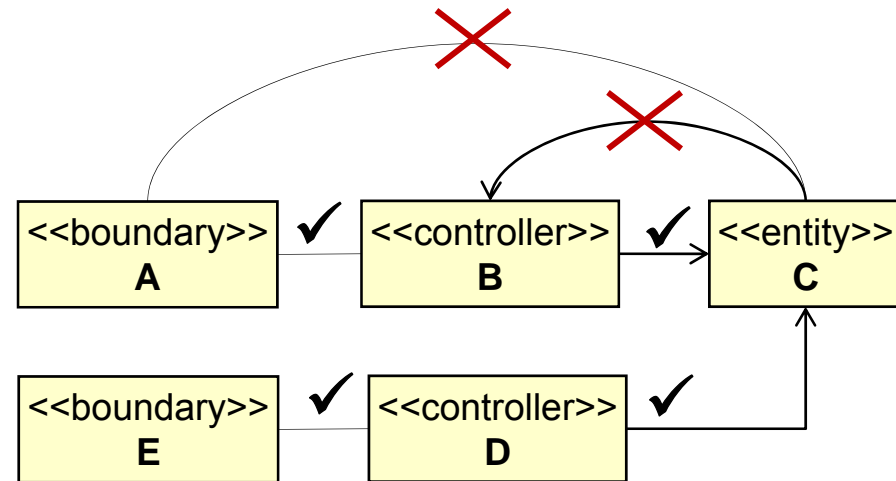
* = Beziehungsnamen, Rollen, Kardinalitäten, Attribute und Operationen aus Platzmangel unterschlagen.

- Man kann die Stereotypen für Analyse-Klassen graphisch oder textuell notieren
- Klassendiagramme des Analysemodells (die die drei Stereotypen nutzen) heißen auch „**Robustness Diagrams**“ → Warum wohl?
- Vergleichen Sie obiges Diagramm mit dem Klassendiagramm der Digitaluhr im UML-Foliensatz und diskutieren Sie die Unterschiede!

Analyse-Objektmodell ▶ Verbotene Abhängigkeiten

- Verbotene Abhängigkeiten zwischen Analyse-Typen

- ◆ a) Boundary zu Entity
- ◆ b) Entity zu Boundary
- ◆ c) Entity zu Controller



- Sinn des Verbots a)

- ◆ Boundaries sollen keine Kontrollaufgaben übernehmen
- ◆ Wartbarkeit und automatische Codeerzeugung für graph. Oberflächen

- Sinn der Verbote b,c)

- ◆ Entity-Typen sind unempfindlich gegen Änderungen in anderen Typen
 - ⇒ C muss nicht geändert werden wenn sich A, B, D oder E ändert
- ◆ Nutzung eines Entity-Typs in mehreren Use Cases (= Controller-Typen) möglich
 - ⇒ Wenn C auf Interaktionen mit A und B ausgelegt wäre, könnte es nicht von D genutzt werden

5.2.3 Dynamische Modellierung und Beispiel zum kompletten Analyseworkflow

Erstellung des Analyse-Verhaltensmodells

Beispiel zum Analyseworkflow

1. **Strukturmodell:** Von Use Cases zu Boundaries, Controllern und Entities
2. **Verhaltensmodell:** Interaktionen der Analysetypen

Weitere Hinweise zur dynamischen Modellierung

Dynamische Modellierung von Benutzerinterfaces

Vom Analyse-Objektmodell zum Analyse-Verhaltensmodell

- Analyse-Verhaltensmodell
 - ◆ Eine Sammlung von Interaktionsdiagrammen und Zustandsdiagrammen
 - ◆ Je ein Interaktionsdiagramm für den Ereignisfluss jedes wichtigen Use Case
 - ◆ Je ein Zustandsdiagramm für Typen mit komplexen Verhalten
- Zweck
 - ◆ Identifikation von Methoden für das Objektmodell
 - ◆ Präzisierung von Methodenimplementierungen
 - ◆ Verfeinerung des Objektmodells
- Vorgehensweise
 - ◆ Erstelle Analyse-Objektmodell zu einer Gruppe von Use Cases
 - ⇒ Abschnitt 5.2.2
 - ◆ Iteriere für alle ausgewählten Use Cases nachfolgendes Verfahren
 - ⇒ nächste Seite

Verhaltensmodellierung eines Use Case

Ausgangspunkt

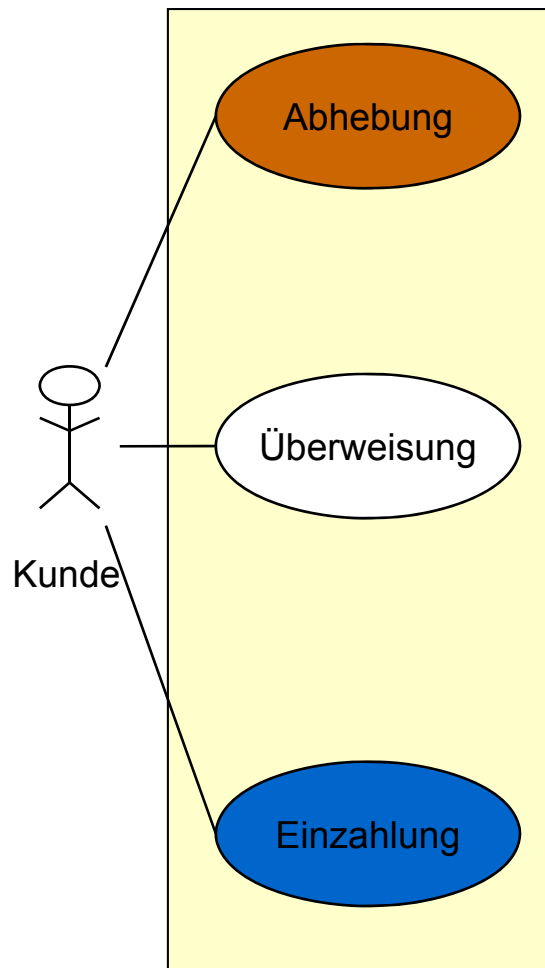
- (0) Vorhandenes Analyse-Objektmodell

Verfahren (pro Use Case)

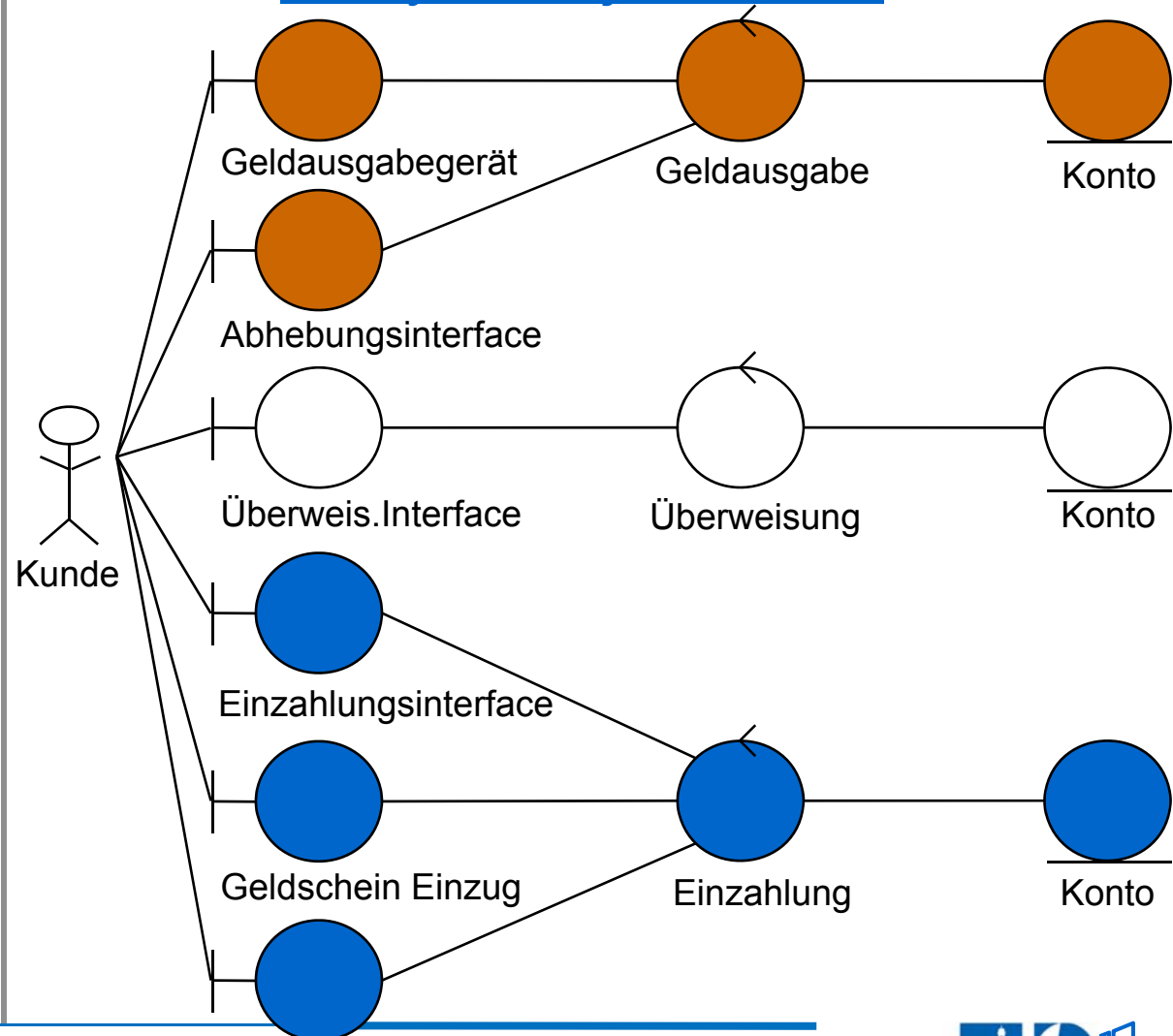
- (1) Konkretisiere Ereignisfluss
 - ◆ Bilde Schritte im Ereignisfluss auf Methoden oder Ereignisse („events“) ab
- (2) Ergänze Klassendiagramm
 - ◆ Erweitere Klassen um evtl. fehlende Methoden, Parameter und Attribute
- (3) Modelliere Ereignisfluss
 - ◆ Zusammenspiel von Objekten → Interaktionsdiagramm(e)
 - ◆ Verhalten einzelner Objekte → Zustandsdiagramm(e)
- (4) Verfeinere dynamisches Modell und Objektmodell
 - ◆ mehr Zwischenschritte (neue Methoden und Nachrichten)
 - ◆ mehr Strukturdetails (neue Attribute, Parameter, Typen)

Use-Case Modell → Analyse-Objektmodell

Use-Case Modell

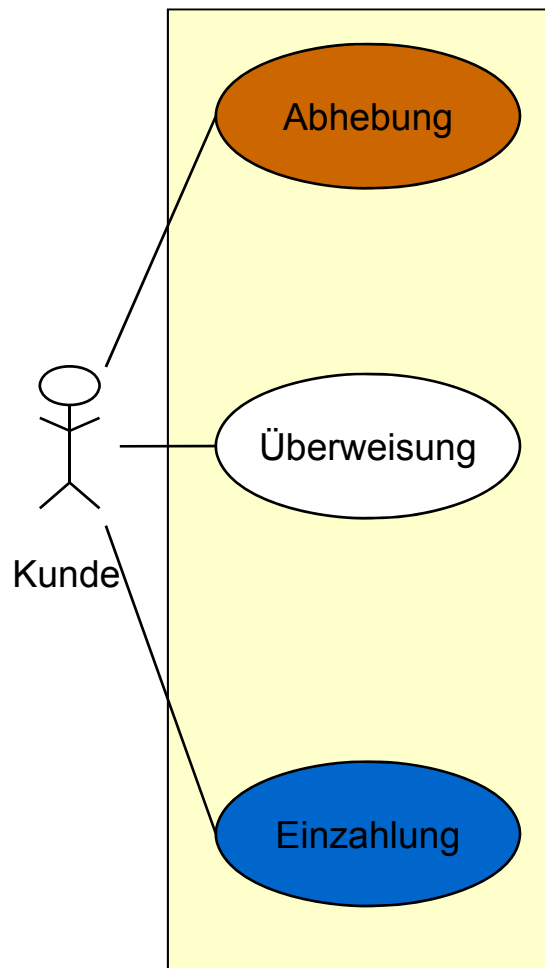


Analyse-Objektmodell

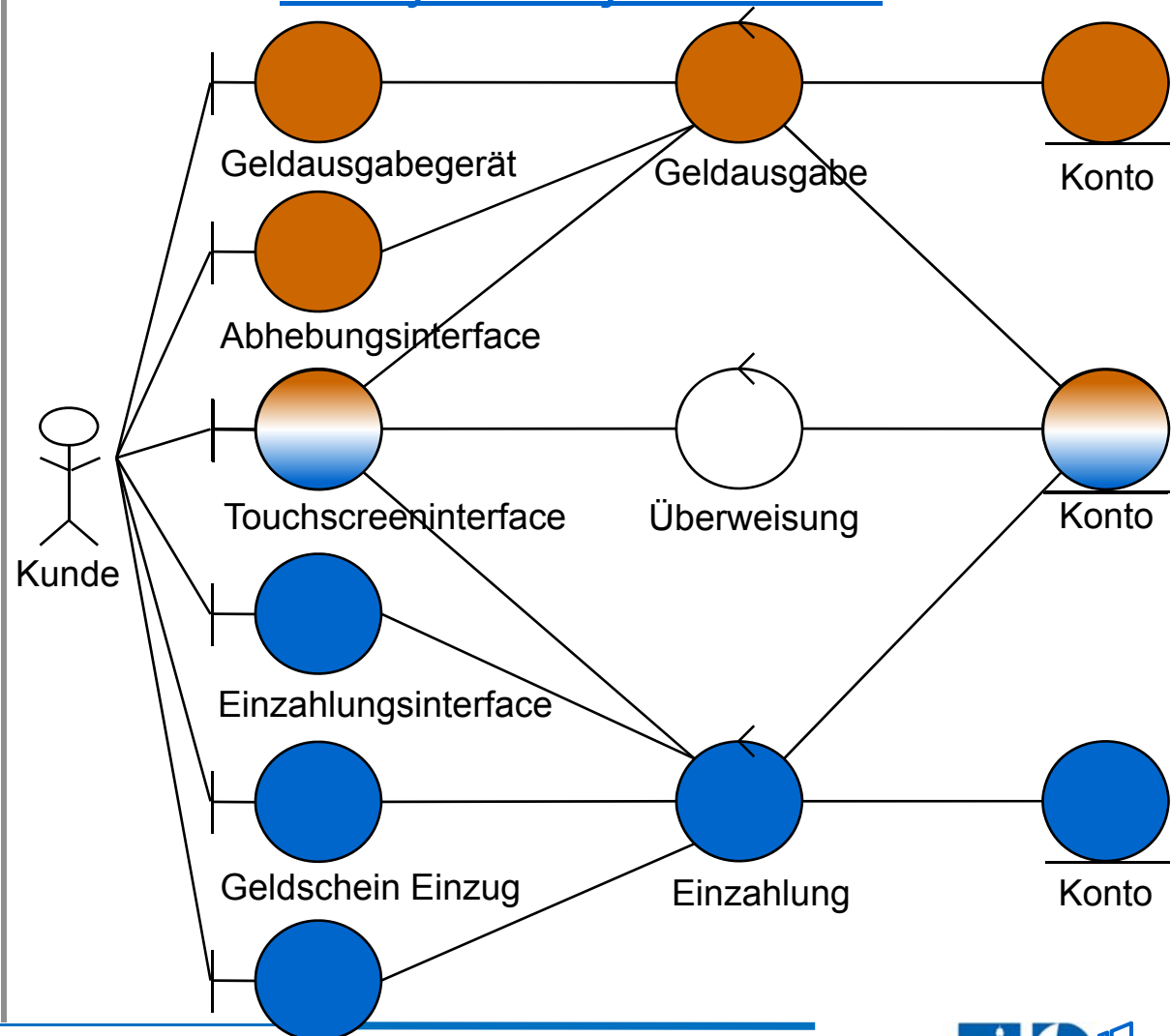


Use-Case Modell → Analyse-Objektmodell

Use-Case Modell

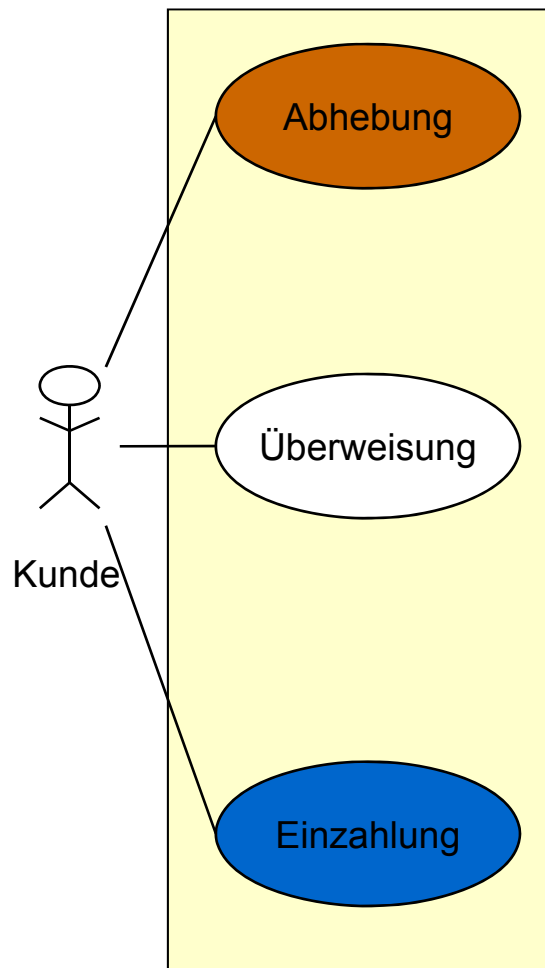


Analyse-Objektmodell

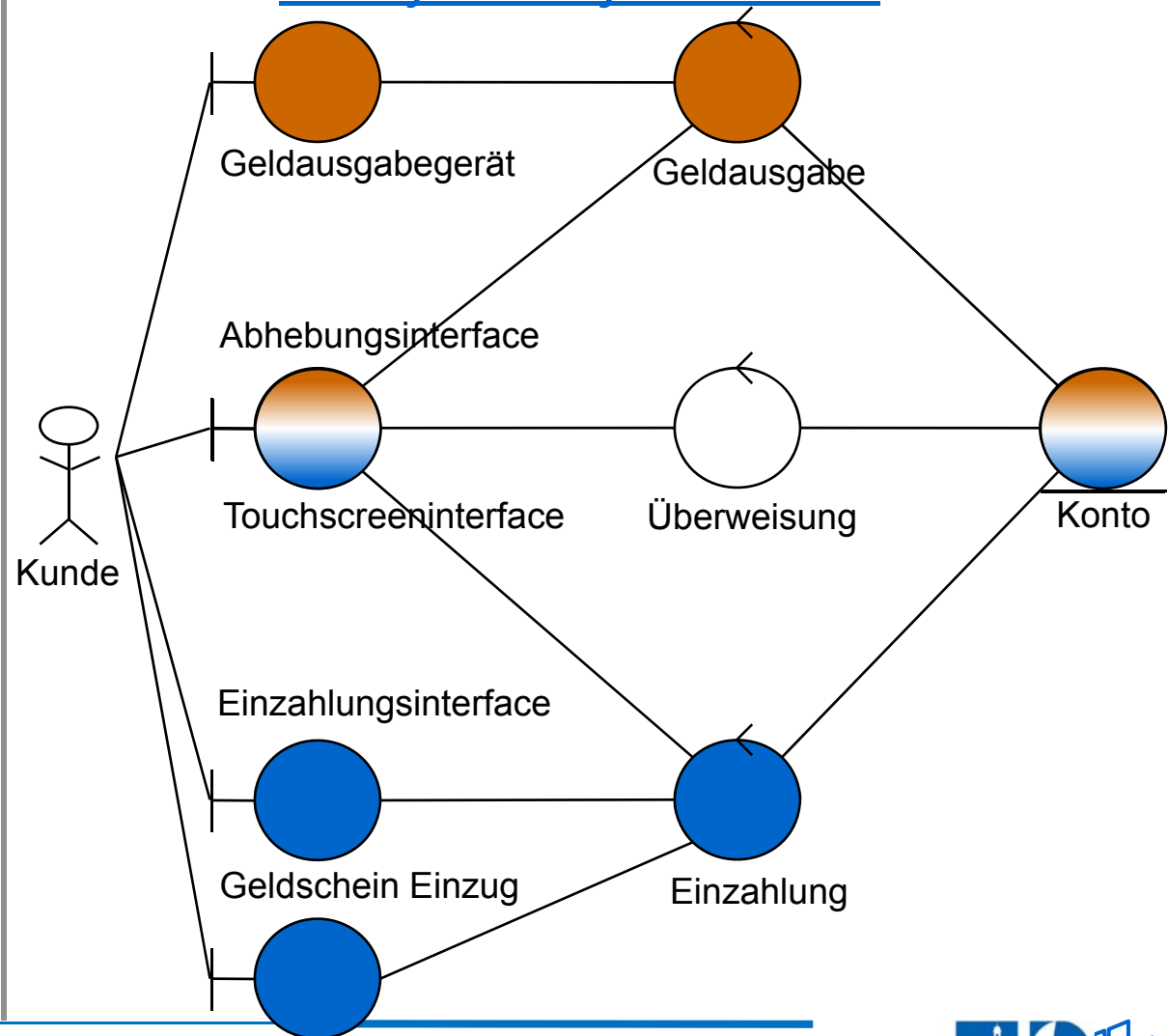


Use-Case Modell → Analyse-Objektmodell

Use-Case Modell

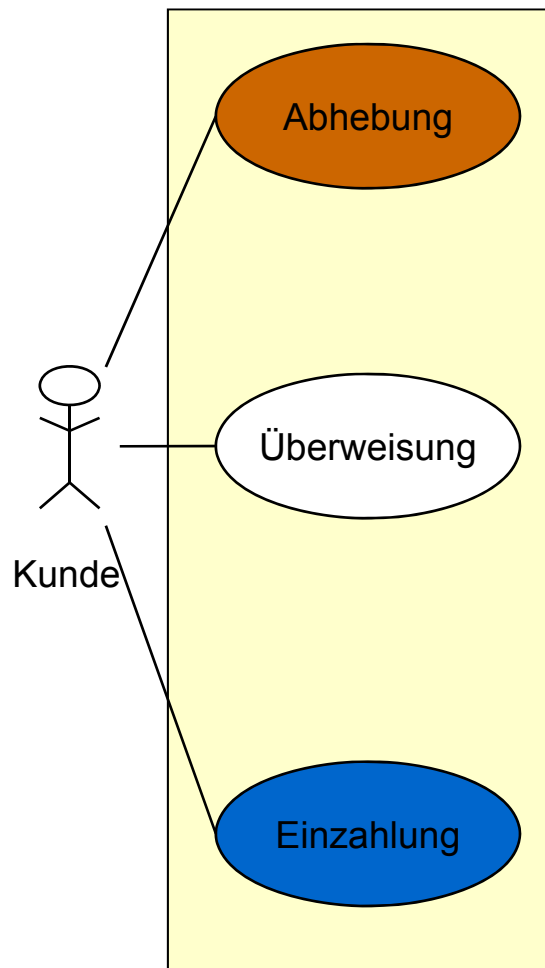


Analyse-Objektmodell

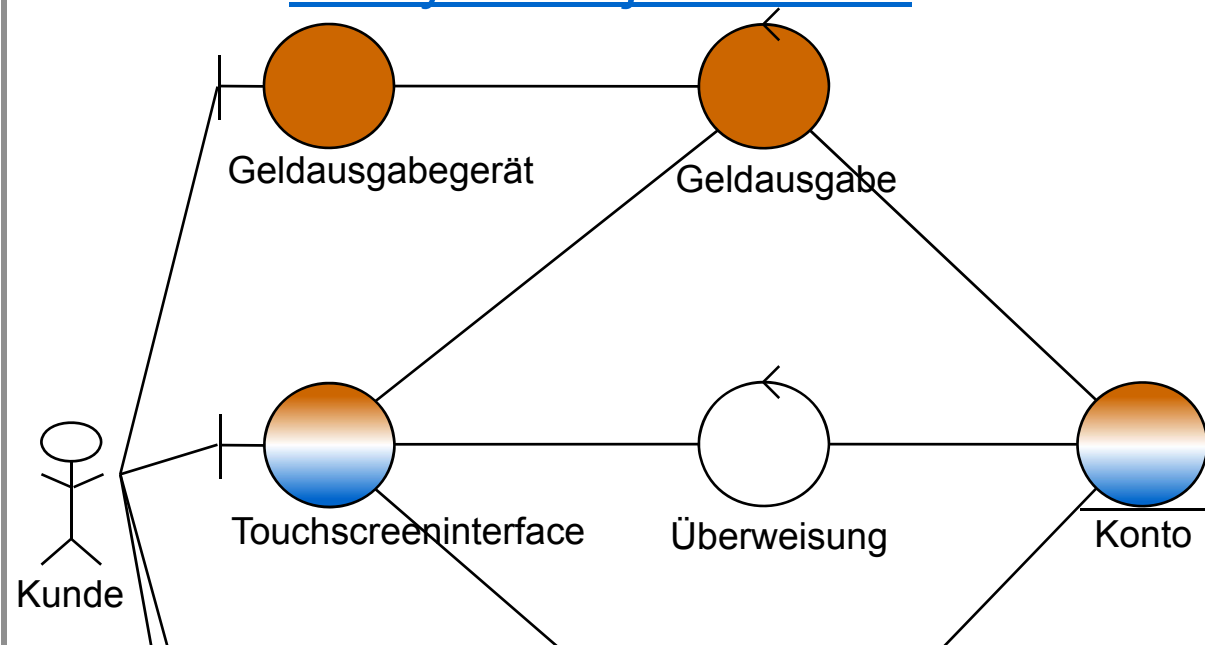


Use-Case Modell → Analyse-Objektmodell

Use-Case Modell



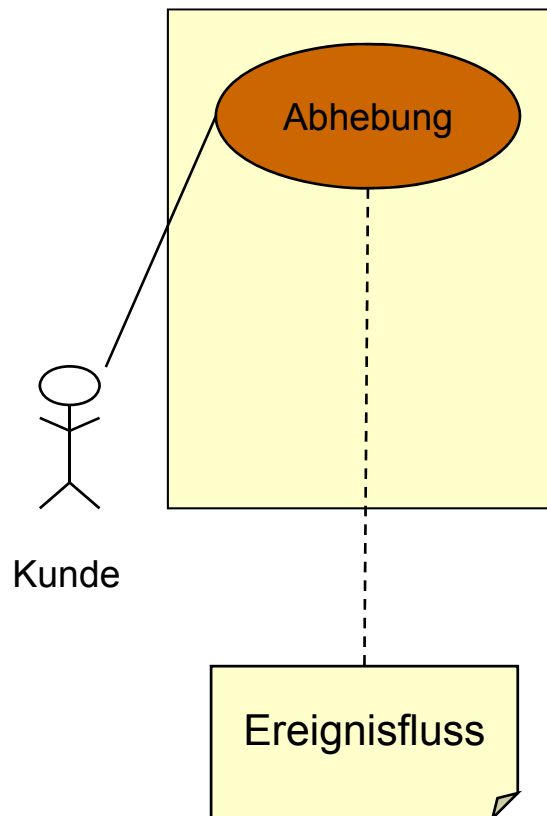
Analyse-Objektmodell



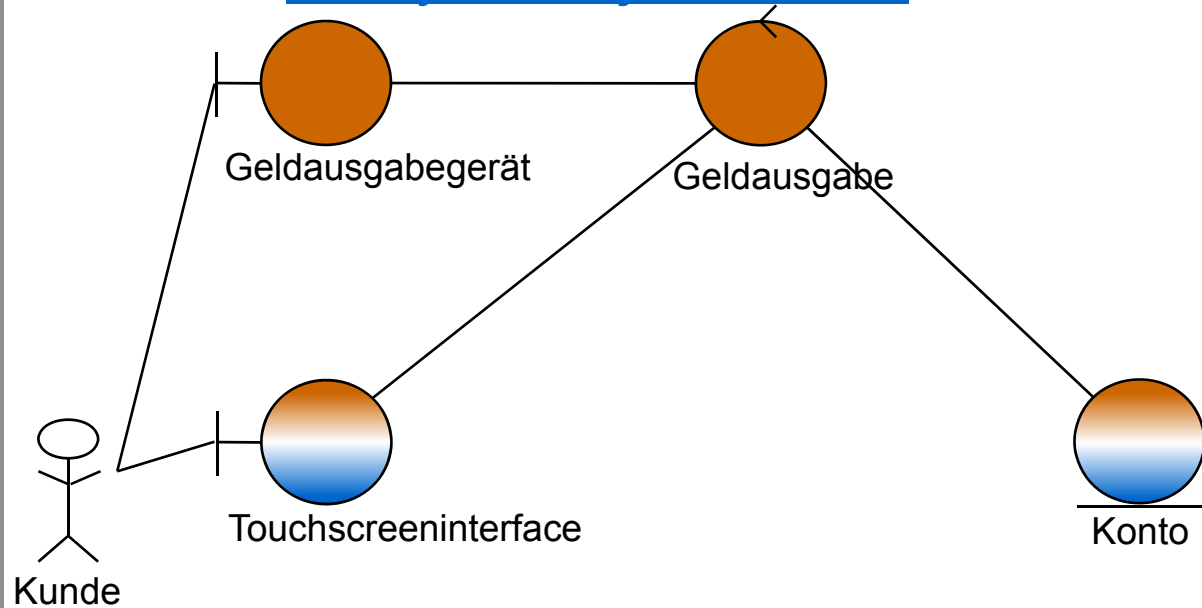
- Zusammenfassung von Objekten aus verschiedenen Use-Case-Realisierungen, die ...
- ... ähnliches tun
 - ◆ TouchScreen statt getrenntes Abhebungs- und EinzahlungsInterface
- ... konzeptuell identisch sind
 - ◆ Konto

Use-Case Modell → Verhaltensmodell der Analyse

Use-Case Modell



Analyse-Objektmodell

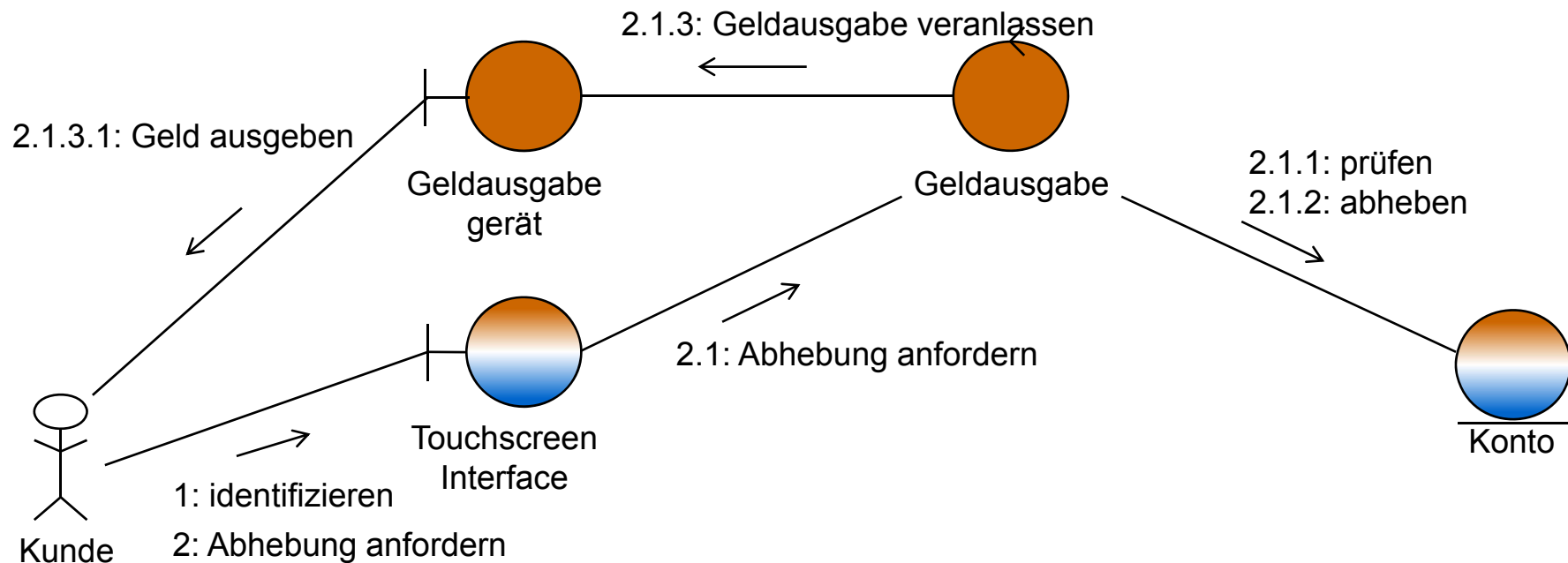


Nächster Schritt: Ereignisfluss des Use Case „Abhebung“ auf Interaktionen der betroffenen Analyse-Objekte abbilden!

→ **Kommunikationsdiagramm**

Objektmodell → Verhaltensmodell

Ereignisfluss "Abhebung" auf Kommunikationsdiagramm abbilden

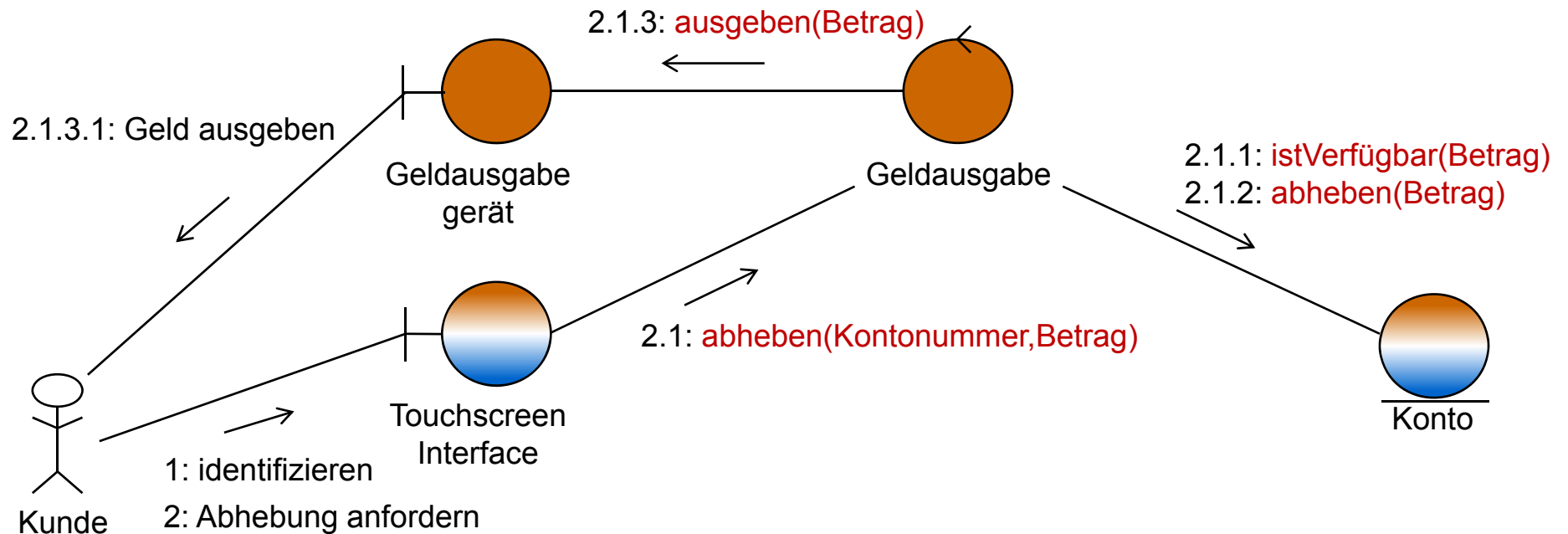


- Fortsetzung (1)

- ◆ informelle Aktionen durch konkrete Nachrichten ersetzen

Objektmodell → Verhaltensmodell

Ereignisfluss "Abhebung" auf Kommunikationsdiagramm abbilden

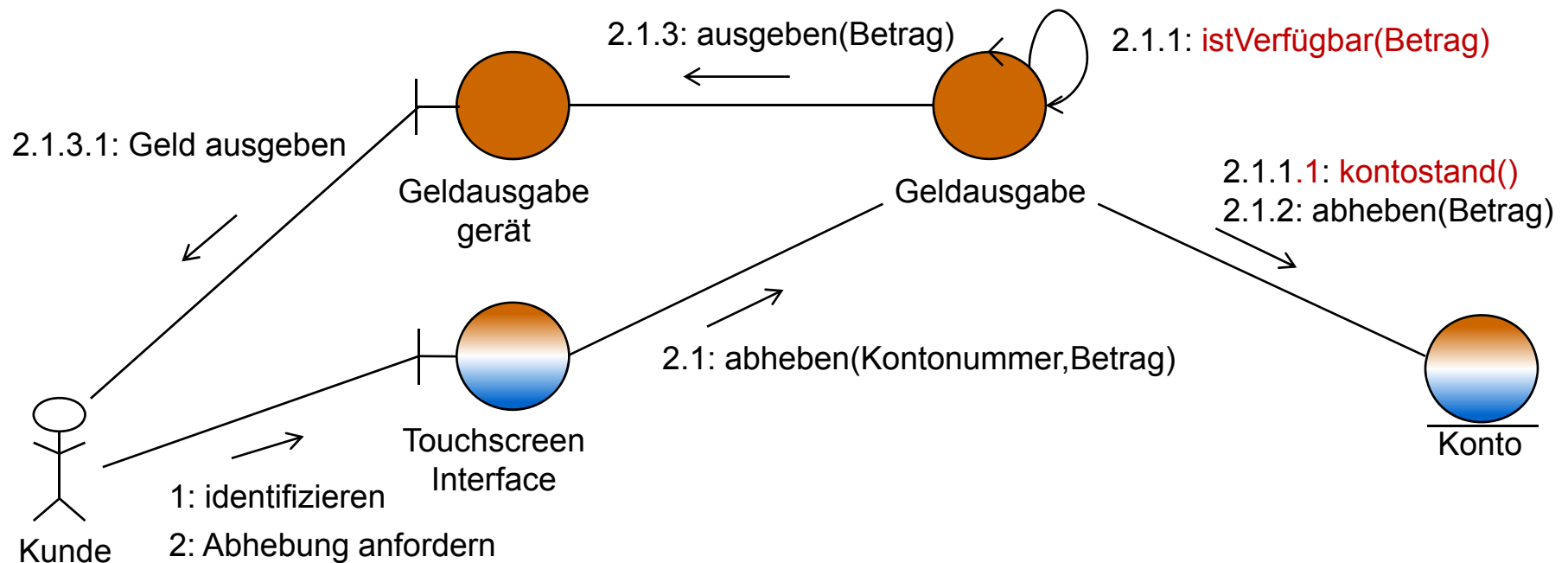


- Fortsetzung (1)

- ◆ informelle Aktionen durch konkrete Nachrichten ersetzen

Objektmodell → Verhaltensmodell

Ereignisfluss "Abhebung" auf Kommunikationsdiagramm abbilden

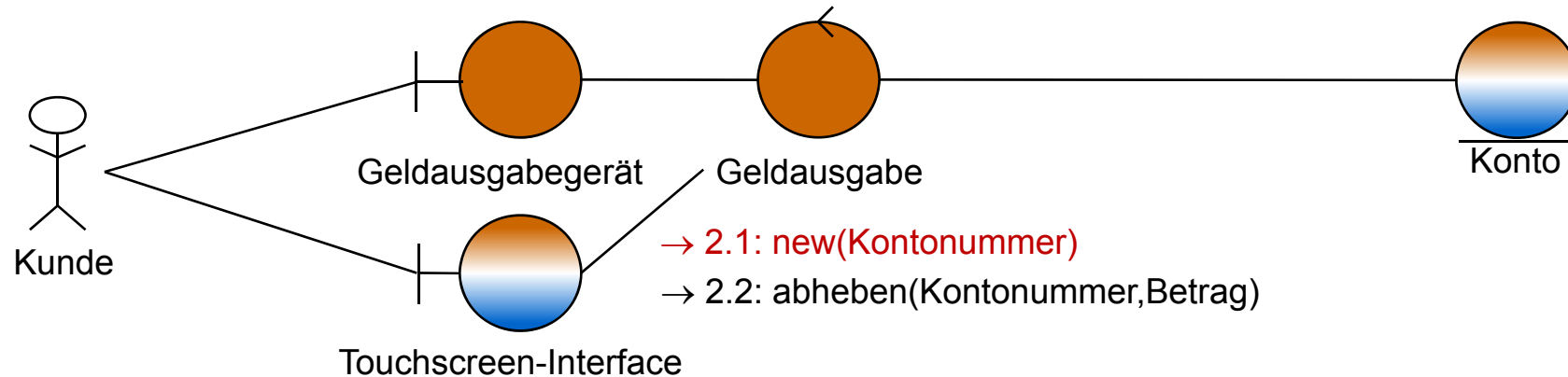


● Fortsetzung (2)

- ◆ Verantwortlichkeiten der Objekte überdenken → Zuordnung von Nachrichten anpassen (Bsp: istVerfügbar() in Controller statt in Entität)
- ◆ Methoden der Objekte ergänzen: Was braucht man noch für „istVerfügbar()“?

Analyse-Verhaltensmodell

► Objekt-Erzeugung

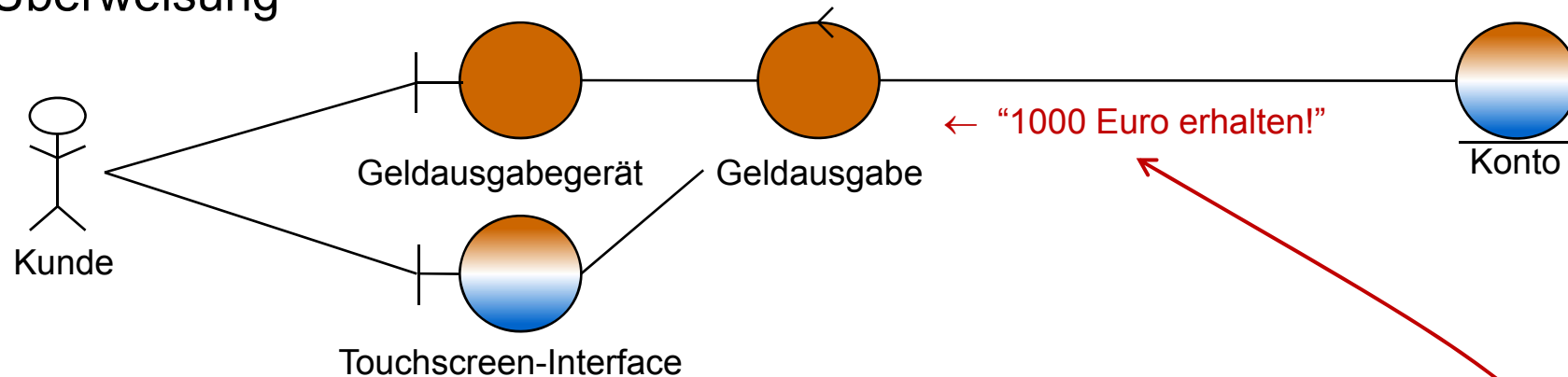


- **Controller-Objekte** werden bei der Initiierung des Use Case erschaffen
 - ◆ **A)** In einem Boundary das der Initiierung des Use Case dient
 - ◆ **B)** In Controller eines Use Case der die Quelle einer <<include>> oder das Ziel einer <<extend>>-Beziehung zum eigenen Use Case ist
- **Boundary-Objekte** werden von Controller-Objekten erschaffen
 - ◆ Im obigen Beispiel ausnahmsweise nicht! → **Denksport: Warum???**
- **Entity-Objekte** existieren schon (meist persistent) oder werden von Controller-Objekten erschaffen

Analyse-Verhaltensmodell

► Objekt-Zugriff

Szenario: „Sofortige Kontostandsaktualisierung bei eintreffender Überweisung“



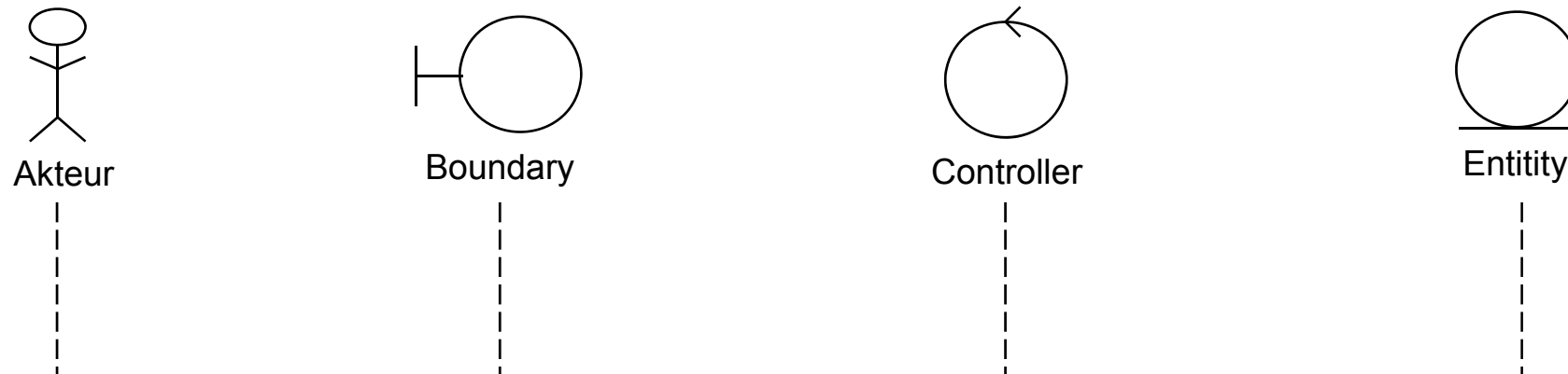
● Objekt-Zugriff

- ◆ Trotz der Einschränkungen auf Typenebene (s. Analyse-Objektmodell „Verbotene Abhängigkeiten“) dürfen Entity-, Boundary- und Controller-**Objekte** beliebig miteinander interagieren!
- ◆ Interaktionen, die scheinbar „schlechten Abhängigkeiten“ entsprechen erfordern jedoch die Ergänzung des Klassendiagramms um das Entwurfsmuster „Observer“

⇒ Siehe Kapitel „Entwurfsmuster“ und „Objektentwurf“

Analyse-Verhaltensmodell ▶

Sequenzdiagramme für Analyseobjekte



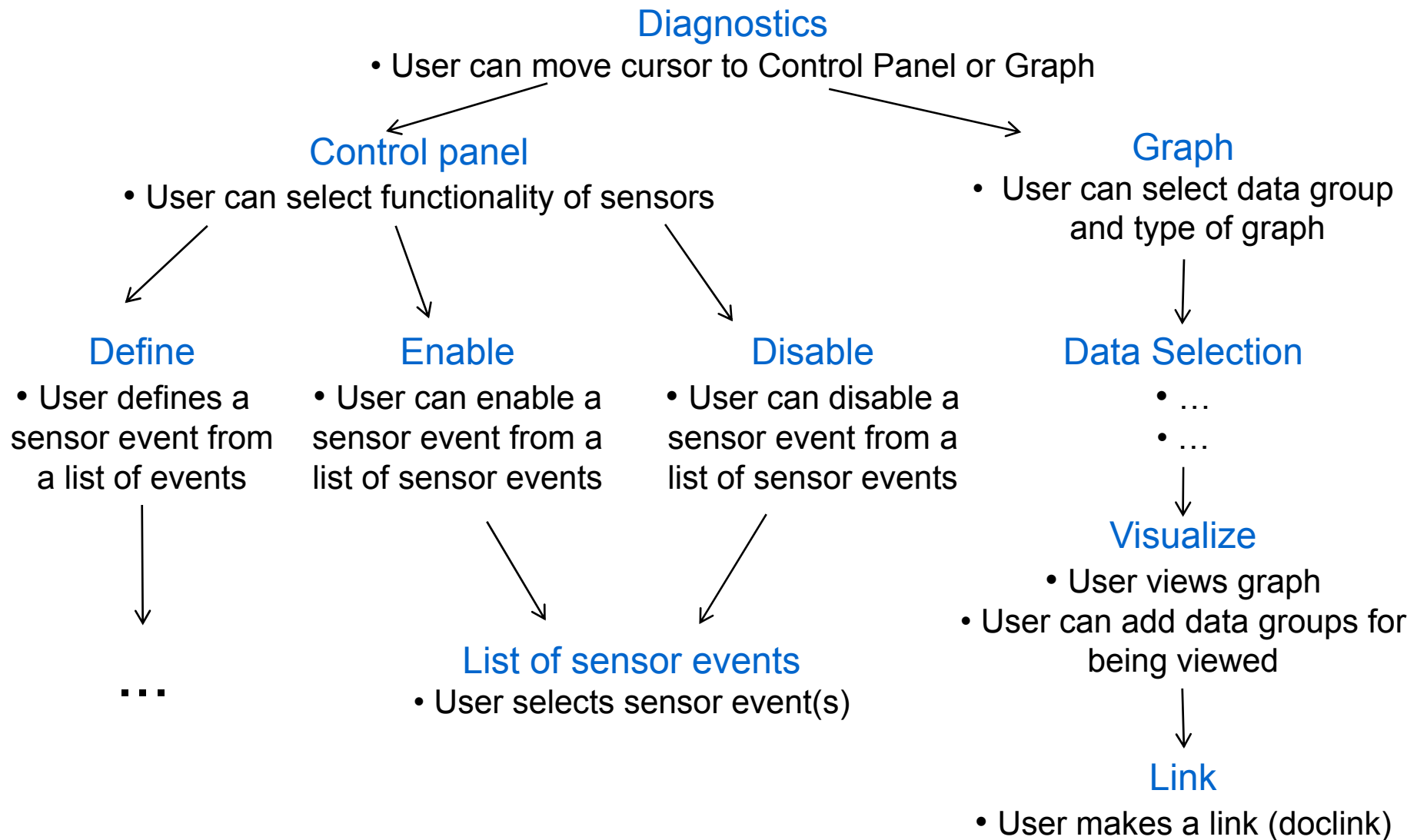
- **Layout-Konventionen**

- ◆ **Spalte 1.** ▶ **Akteur** der den Use Case initiiert hat
- ◆ **Spalte 2.** ▶ **Boundary-Objekt** mit dem der Akteur als erstes interagiert
- ◆ ... ▶ Weitere Boundaries
- ◆ **Danach** ▶ **Kontroll-Objekt** das den Use Case steuert
- ◆ **Danach** ▶ **Entities**
- ◆ **Danach** ▶ Evtl. weitere Kontroll-Objekte ← **Erinnern Sie sich, wann es weitere Kontrollobjekte geben kann?**

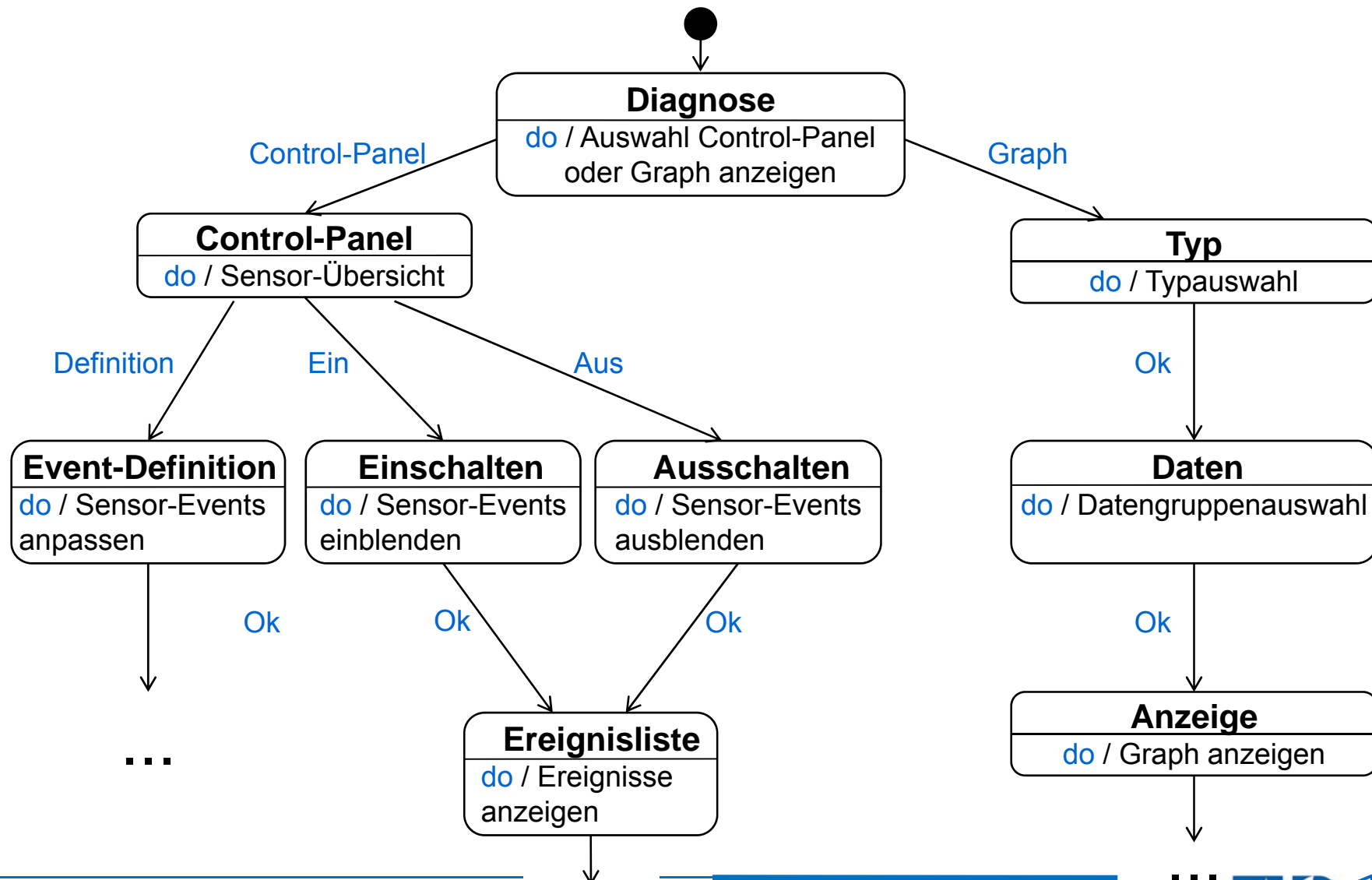
Dynamische Modellierung von Benutzerschnittstellen

- Navigationspfade
 - ◆ Eine Variation von Zustandsdiagrammen
 - ◆ Werden zum Design von Benutzerschnittstellen benutzt
- Zustand ▶ Einzelne Bildschirmansicht („Screen“)
 - ◆ Ein grafisches Layout der mit den Zuständen assoziierten Screens hilft bei der Vorstellung des dynamischen Modells eines Benutzerinterfaces
 - ◆ Aktivitäten/Aktionen sind als Unterpunkte unter dem Screen-Namen aufgeführt
 - ◆ Oft wird nur eine Exit-Aktion angegeben
- Zustandsübergang ▶ Ergebnis einer Exit-Aktion
 - ◆ Klick auf Schaltfläche
 - ◆ Menüauswahl
 - ◆ Cursorbewegungen

Navigationspfade als Graph



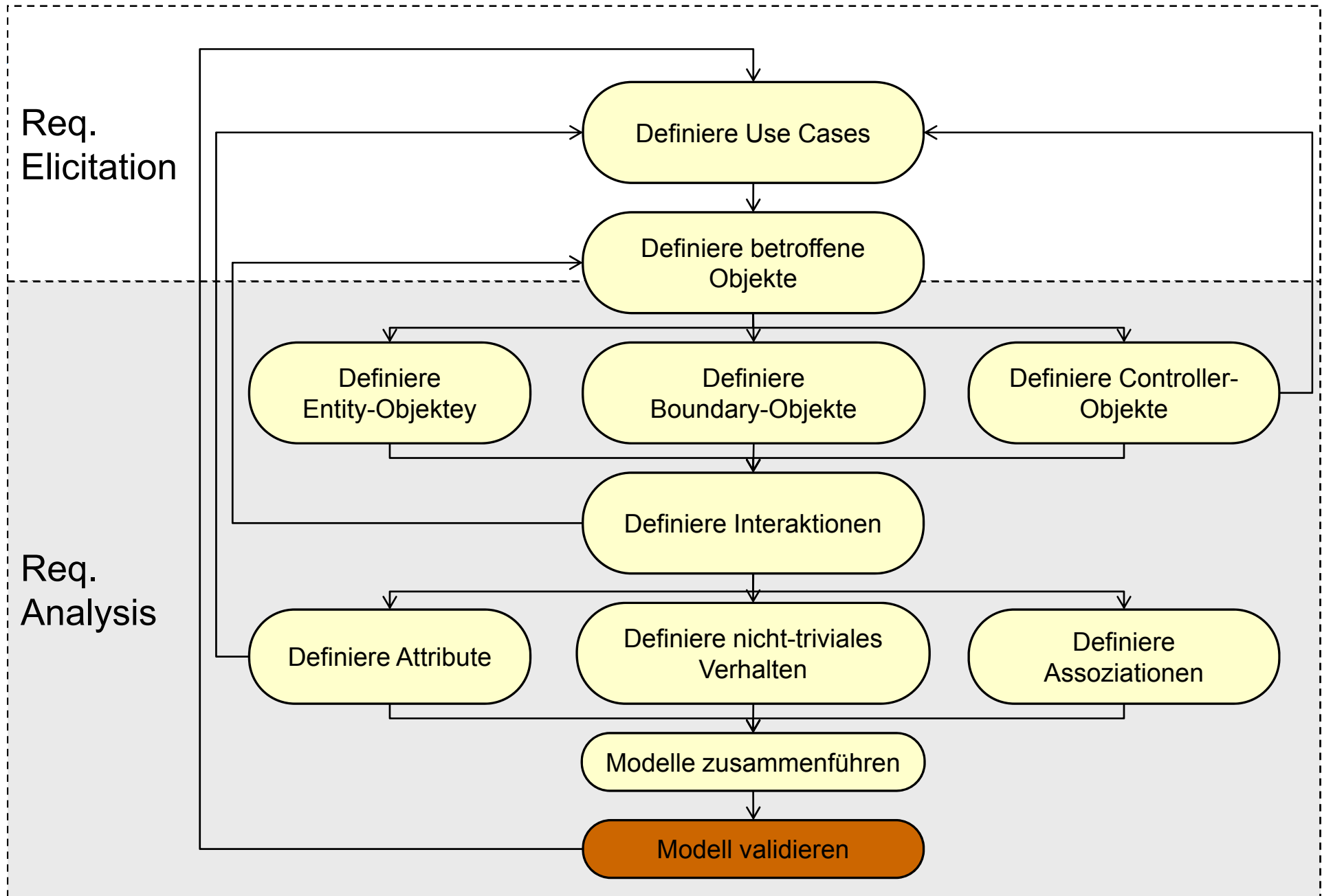
Navigationspfade als Zustandsdiagramm



5.2.5 Konsolidierung der Analyse

(aus Brügge und Dutoit)

Aktivitätsdiagramm des Analyse-Workflow



Kriterien der Anforderungsvalidierung

- Korrektheit
 - ◆ Die Anforderungen repräsentieren die Sicht des Kunden.
- Vollständigkeit
 - ◆ Alle im System möglichen Szenarien sind beschrieben, inklusive Ausnahmeverhalten von System und Benutzer
- Konsistenz
 - ◆ Es gibt keine funktionalen oder nichtfunktionalen Anforderungen die sich widersprechen
- Klarheit
 - ◆ Es gibt keine Zweideutigkeiten bei den Anforderungen.
- Realismus
 - ◆ Anforderungen können implementiert und ausgeliefert werden
- Zurückverfolgbarkeit
 - ◆ Jede Funktion des Systems kann auf einen Satz entsprechender funktionaler Anforderungen zurückgeführt werden

Konsistenz, Vollständigkeit, Mehrdeutigkeit

- Vollständigkeit
 - ◆ Identifikation von fehlenden Klassen (von einem Subsystem referenziert, aber nirgends definiert)
 - ◆ Identifikation von Assoziationen mit losen Enden (Assoziationen, die nirgends hinzeigen)
- Konsistenz
 - ◆ Identifikation von vertauschten „Verdrahtungen“ zwischen Klassen
 - ◆ Identifikation von doppelt definierten Klassen
 - ◆ Benennung von Klassen, Attributen, Methoden
 - ⇒ Keine Synonyme, d.h. keine verschiedene Namen für gleiche Bedeutung
- Mehrdeutigkeiten
 - ◆ Rechtschreibfehler in Namen
 - ◆ Benennung von Klassen, Attributen, Methoden
 - ⇒ Keine Homonyme, d.h. keine gleichen Namen für unterschiedliche Bedeutungen

Anforderungsevolution

- Anforderungen ändern sich rapide während der Anforderungserhebung
- Tools zur Verwaltung von Anforderungen
 - ◆ Speichern Anforderungen in einem Repository
 - ◆ Erstellen automatisch Anforderungsdokumente aus dem Repository
 - ◆ Unterstützen Zurückverfolgbarkeit und Änderungsmanagement für den ganzen Lebenszyklus des Projektes
 - ◆ Unterstützen Multi-user Zugriff
- Beispiele
 - ◆ Requisite Pro (Rational / IBM)
 - ⇒ <http://www.ibm.com/developerworks/rational/products/requisitepro/>
 - ◆ Jira
 - ⇒ <http://www.jira.com>

Formatvorlage Anforderungsanalyse-Dokument




- 1. Einführung
- 2. Momentanes System
- 3. Beabsichtigtes System
 - 3.1 Überblick
 - 3.2 Funktionale Anforderungen
 - 3.3 Nichtfunktionale Anforderungen
 - 3.4 Nebenbedingungen (“Pseudoanforderungen”)
 - 3.5 Systemmodelle
- 4. Glossar

Exkurs „Analyseformatvorlage“

Projektvereinbarung

- Die Projektvereinbarung steht für die Akzeptanz des Analysemodells (dokumentiert durch das Anforderungsanalyse-Dokument) durch den Kunden.
- Kunde und Entwickler einigen sich auf eine Idee, Funktionen und Features des Systems. Zusätzlich einigen sie sich auf:
 - ◆ Eine Liste der Prioritäten
 - ◆ Einen Revisionsprozess
 - ◆ Eine Liste von Kriterien zur Annahme des Systems
 - ◆ Einen Zeitplan und ein Budget

Zusammenfassung: Anforderungsanalyse

- 0. Was sind die Funktionen des Systems?  **Functional Modeling**
 - ◆ Erstelle Szenarios und Use Case Diagramme
 - ⇒ Spreche mit dem Kunden, beobachte, betrachte historische Aufzeichnungen (Protokolle, Berichte), führe Gedankenexperimente durch
- 1. Was ist die Struktur des Systems?  **Object Modeling**
 - ◆ Erstelle Klassendiagramme
 - ⇒ Identifiziere Objekte. Welche Beziehungen zwischen ihnen gibt es? Wie sind die Multiplizitäten?
 - ⇒ Was sind die Attribute der Objekte?
 - ⇒ Welche Operationen sind auf den Objekten definiert?
- 2. Welches sind die Abläufe im System?  **Dynamic Modeling**
 - ◆ Erstelle Sequenzdiagramme
 - ⇒ Identifiziere Sender und Empfänger
 - ⇒ Zeige Abfolge der zwischen Objekten ausgetauschten Events. Finde Abhängigkeiten und Nebenläufigkeiten.
 - ◆ Erstelle Zustandsdiagramme
 - ⇒ Nur für dynamisch interessante Objekte.

Zusammenfassung: Anforderungsanalyse

In diesem Unterkapitel haben wir uns mit folgenden Themen befasst:

- Konstruktion des statischen und dynamischen Analysemodells aus dem Use Case und Domain Object Model.
- Konsolidierung der Analyse
 - ◆ Integration der Modelle verschiedener Subsysteme
 - ◆ Konsistenz, Vollständigkeit, Mehrdeutigkeit