

Kapitel 4

Anforderungserhebung („Requirements Elicitation“)

- Stand: 21.11.2018 ←

Umfangreiche Ergänzung
der Rahmen mit
Bezeichnen

- 4.1 Motivation und Übersicht
- 4.2 Anforderungen, Szenarien und Anwendungsfällen („Use Cases“)
- 4.3 Anwendungsfall-Diagramme
- 4.4 Statische Modellierung der Anwendungsdomäne („Domain Object Model“)
- 4.5 Dynamische Modellierung der Anwendungsfälle
- 4.6 Zusammenfassung

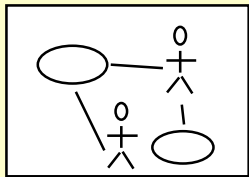
- **Bisher: Grundlagen**
 - ◆ Konfigurationsmanagement
 - ◆ UML
- **Ab jetzt: Arbeitsabläufe („Aktivitäten“ / „Workflows“)** und dazugehörige Technologien, Werkzeuge, etc.
 - ◆ Anforderungserhebung
 - ◆ Systementwurf
 - ◆ Objektentwurf
 - ◆ Implementierung
 - ◆ Testen
 - ◆ Refactoring
- **Danach: Prozessmodelle**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

Aktivitäten bei der Softwareentwicklung („Workflows“)

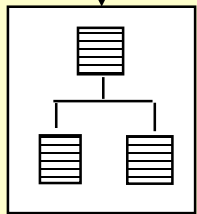
Requirements Engineering

Anforderungs-
erhebung

**Use Case
Model**



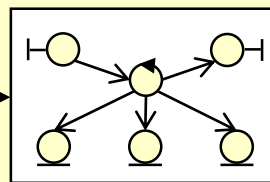
ergänzt
durch



**Domain
Object Model**

Anforderungs-
analyse

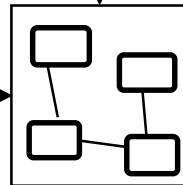
ausgedrückt
als



**Analysis
Model**

System
Design

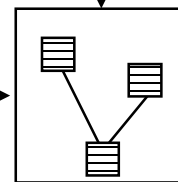
strukturiert
durch



Subsysteme

Object
Design

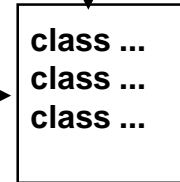
realisiert
durch



Entwurf

Implemen-
tation

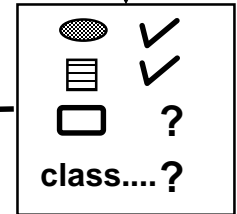
implementiert
von



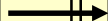
Quellcode

Testen

verifiziert
durch



Testfälle

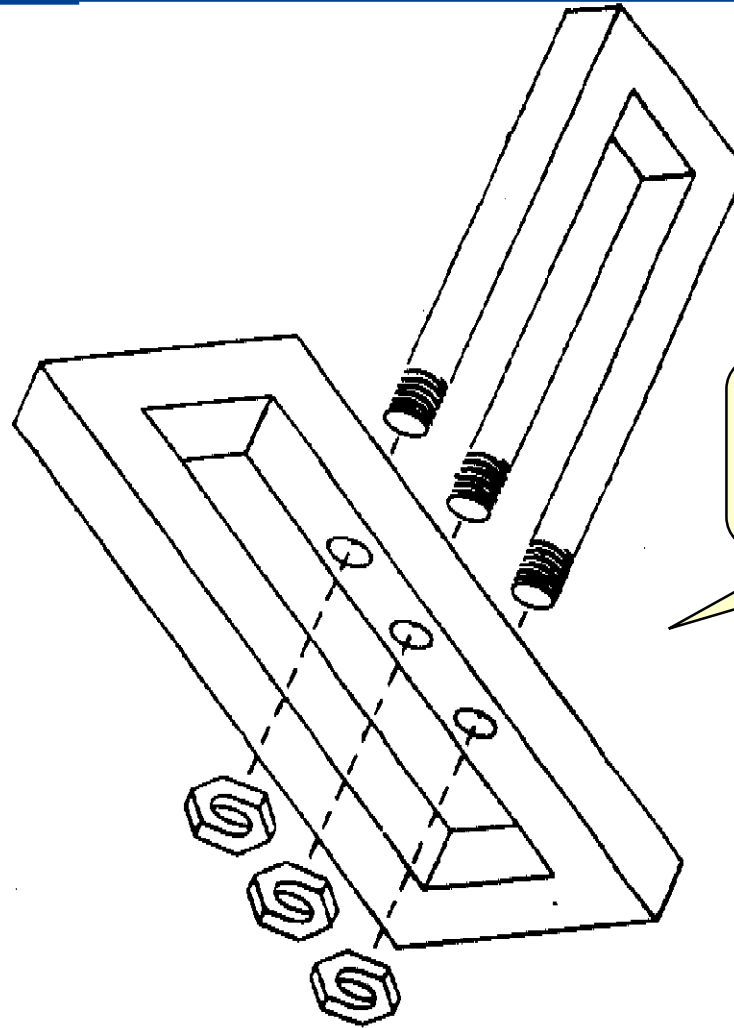


4. 1 Anforderungserhebung

– Was und warum? –

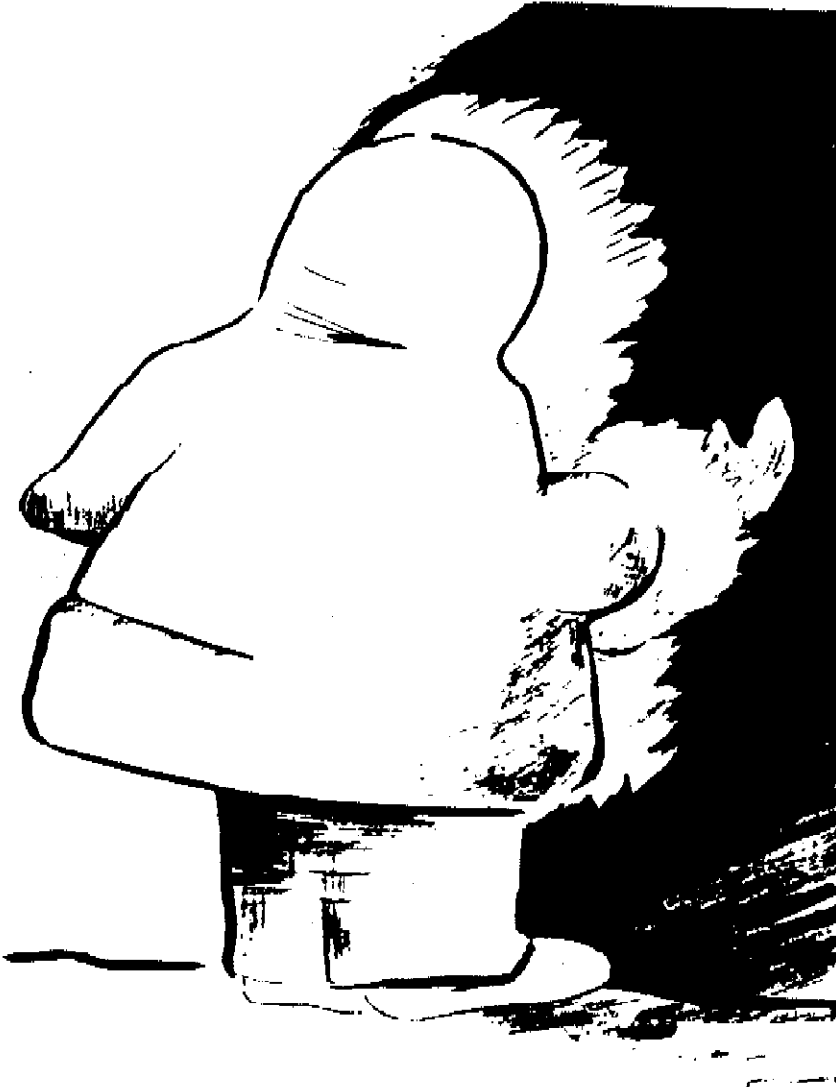
Motivation und Übersicht

Können Sie so etwas bauen?



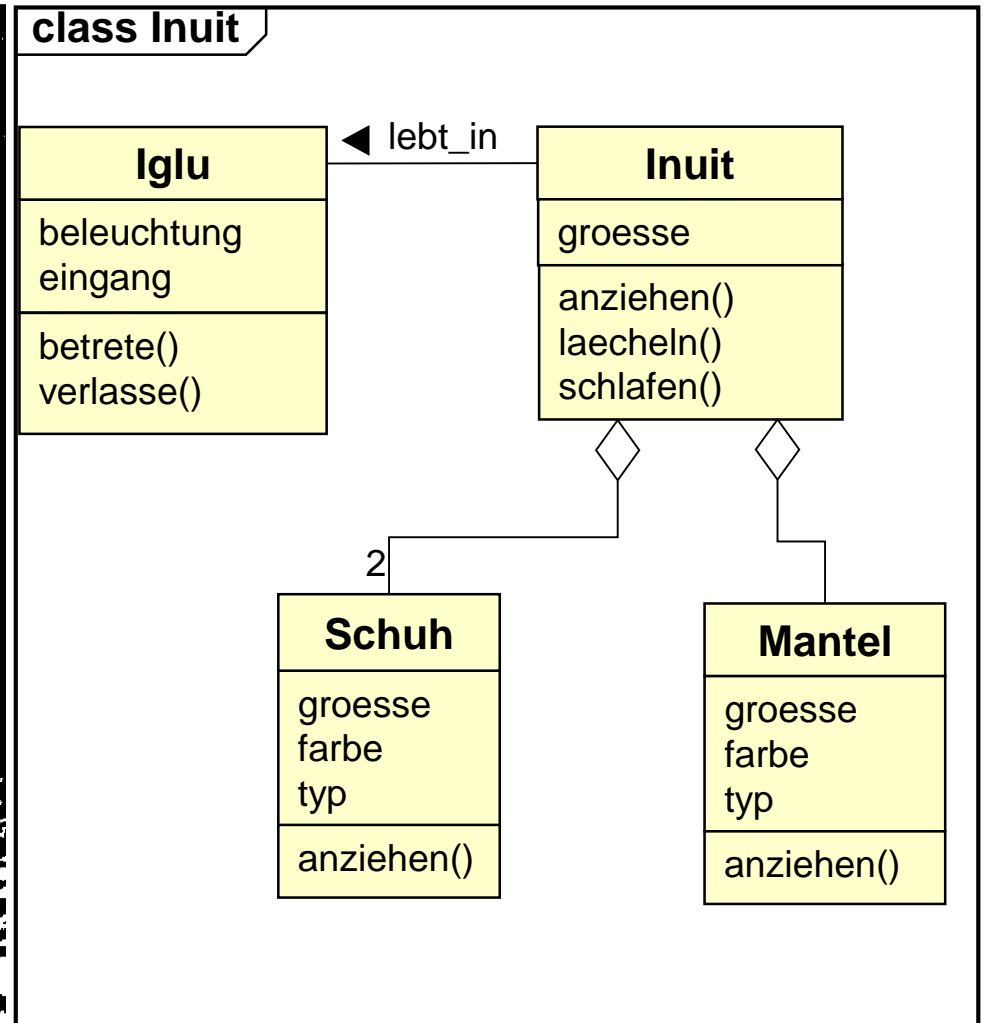
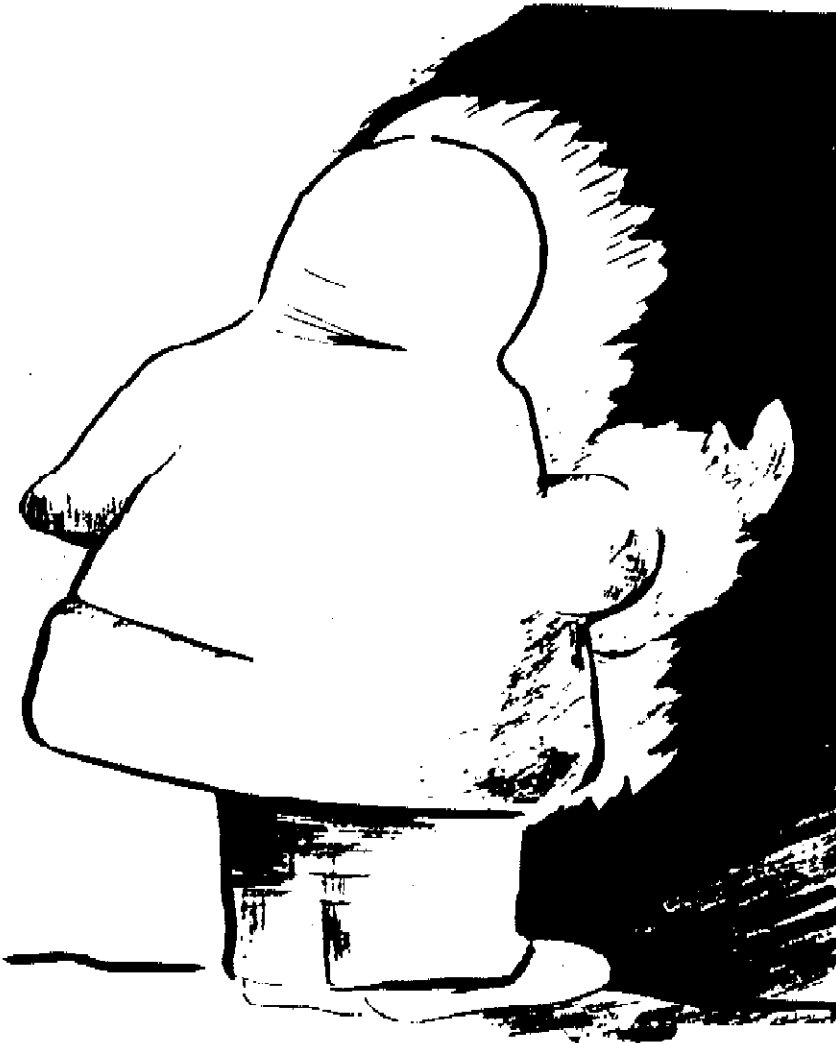
Widersprüchliche
Anforderungen

Was ist das?

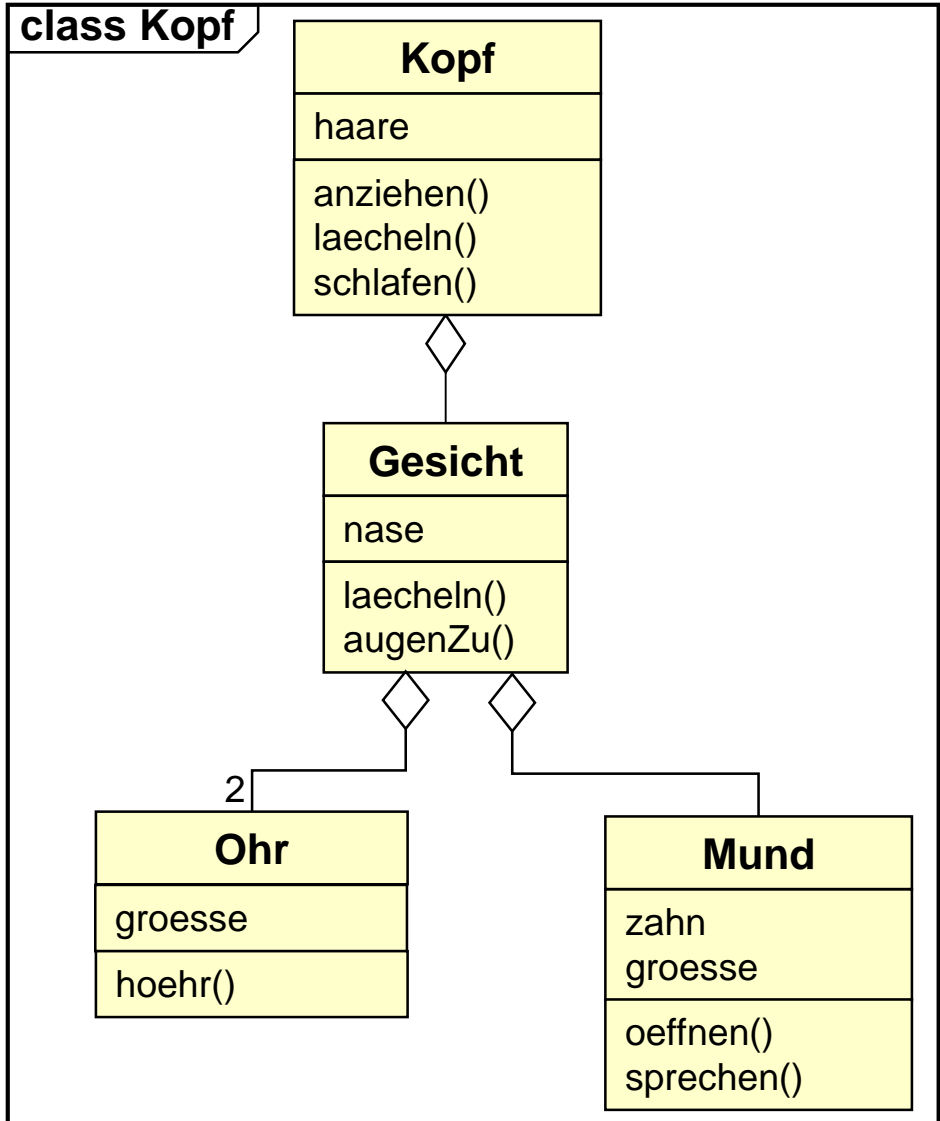
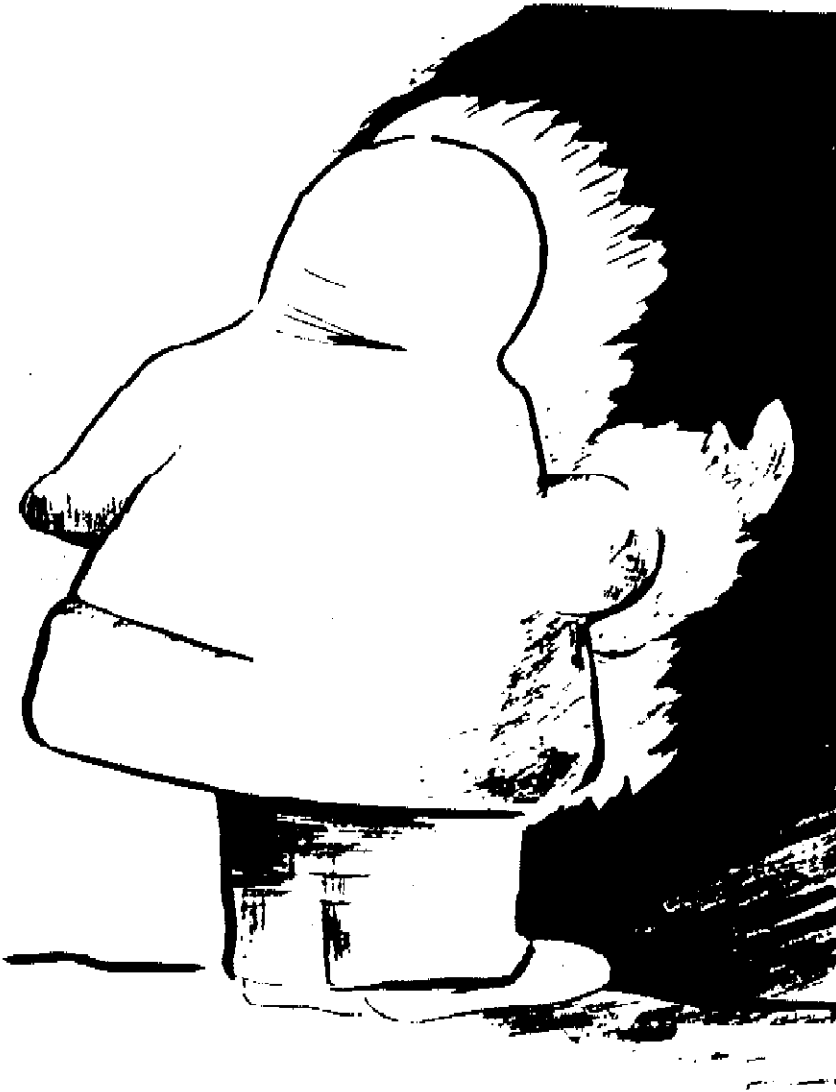


Mehrdeutige
Anforderungen

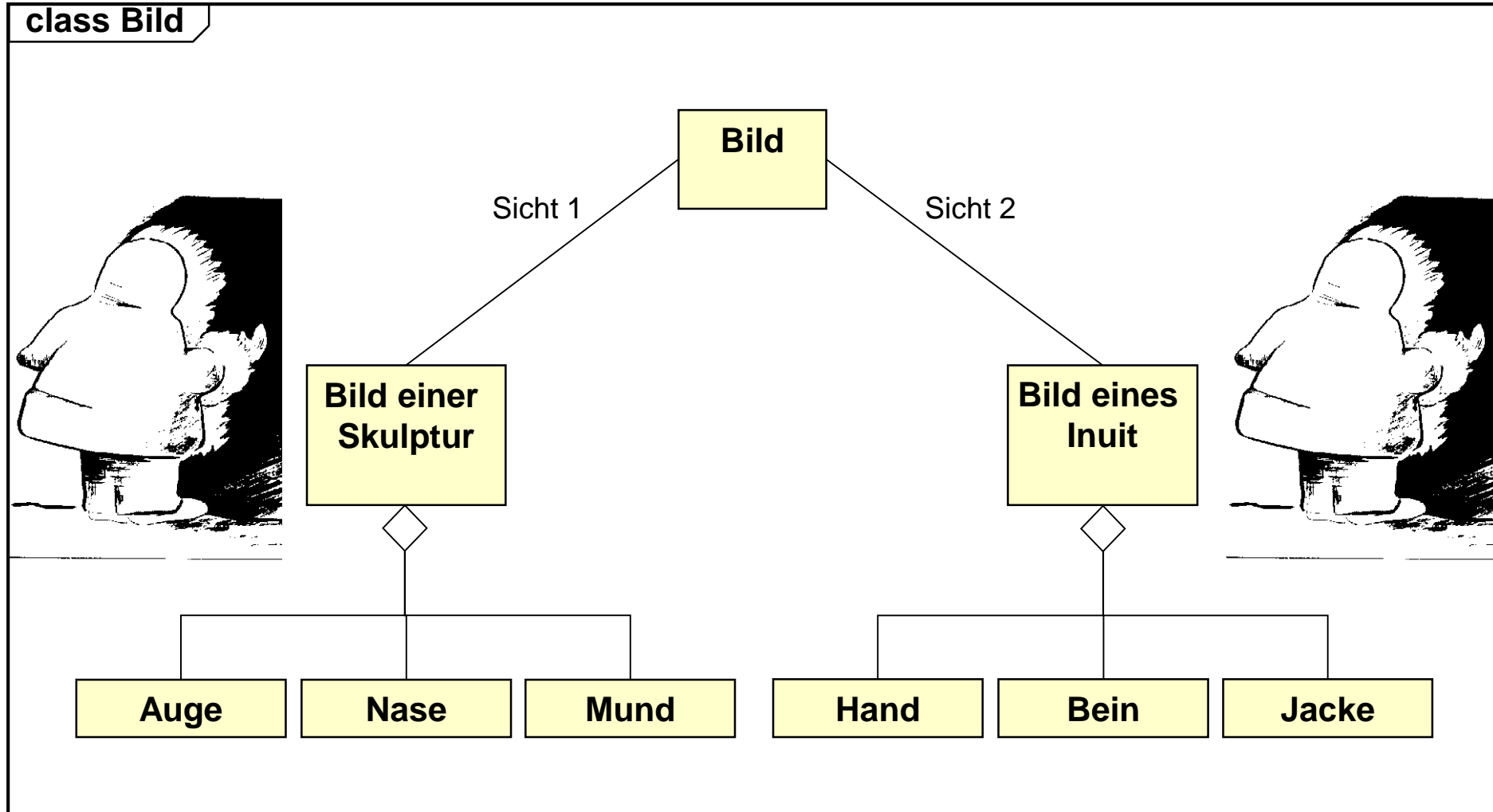
Mögliches Objektmodell ▶ Inuit



Genauso mögliches Objektmodell ▶ Kopf



Objektmodell aus der Sicht des Künstlers



Welches System sollen Sie bauen?



Unscharfe
Anforderungen

Was ist Requirements Engineering (RE)?

- Ziele

- ◆ Bestimmung der **Anforderungen an das System**

- ⇒ Identifikation der **Geschäftsprozesse**, die das System unterstützen soll
 - ⇒ Identifikation der **Funktionen** die das System dafür bieten soll

- ◆ Kennen lernen der **Anwendungsdomäne**

- ⇒ Identifikation von Konzepten, Beziehungen, grundlegendem Verhalten

- Aktivitäten („Workflows“)

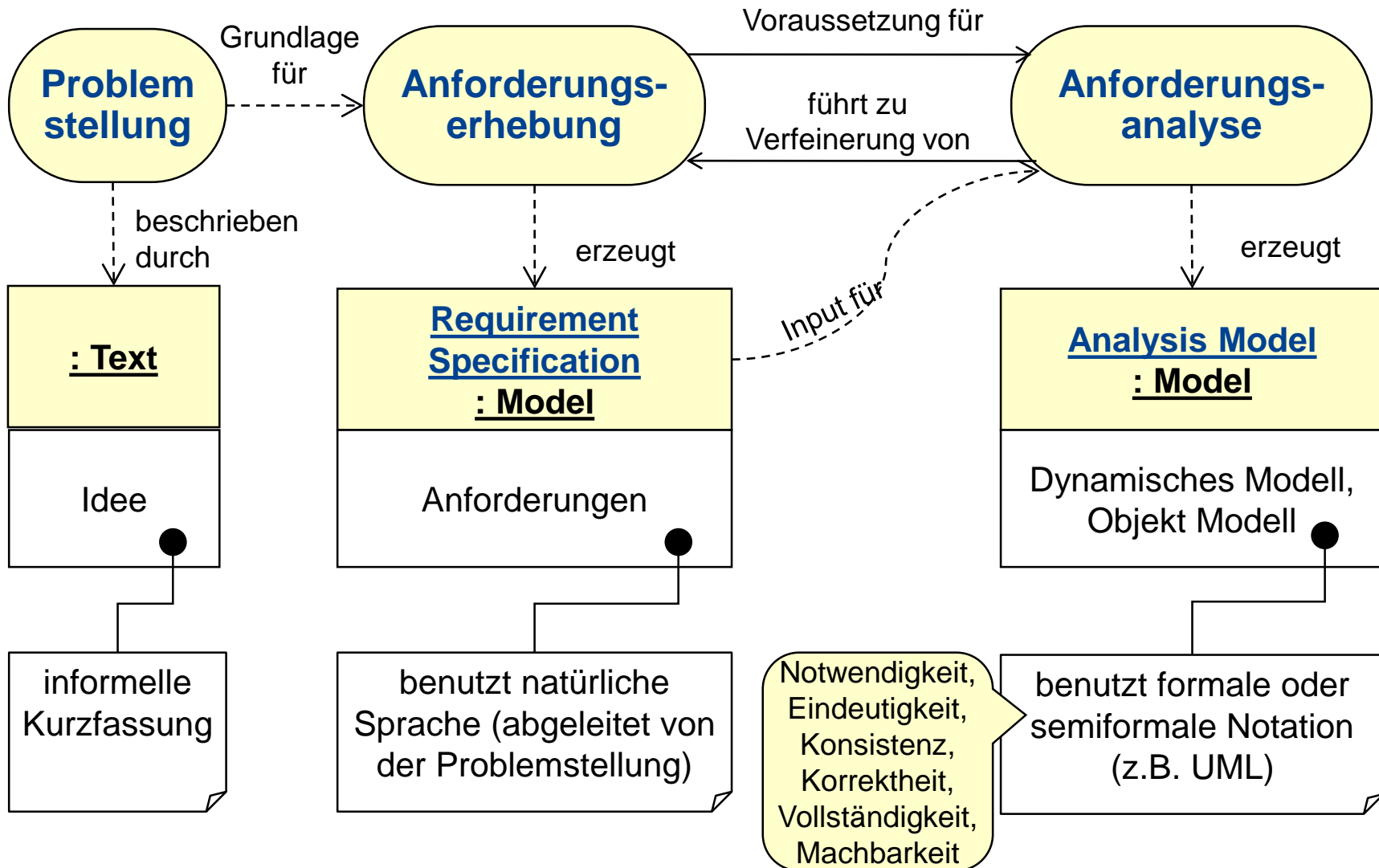
- ◆ **Anforderungserhebung** („Requirements Elicitation“) → Dieses Kapitel:

- ⇒ Definition des Systems in einer Form, die Kunden und Entwickler verstehen

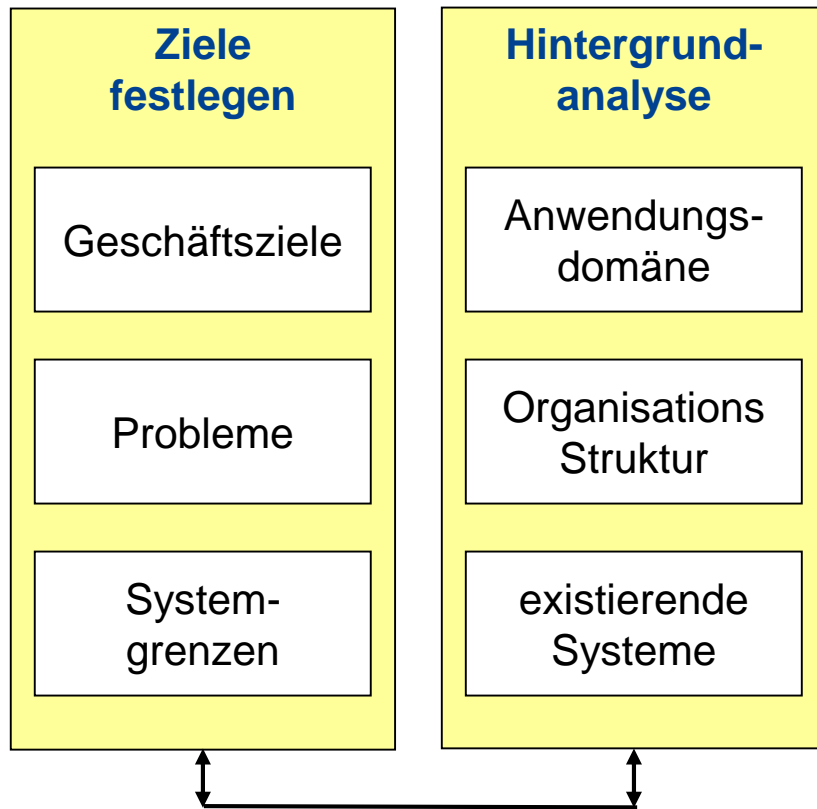
- ◆ **Anforderungsanalyse** („Requirements Analysis“) → Kapitel 6:

- ⇒ Technische Spezifikation des Systems in einer für die Entwickler verständlichen und handhabbaren Form.

Der Requirements Engineering Prozess: Aktivitäten und Produkte

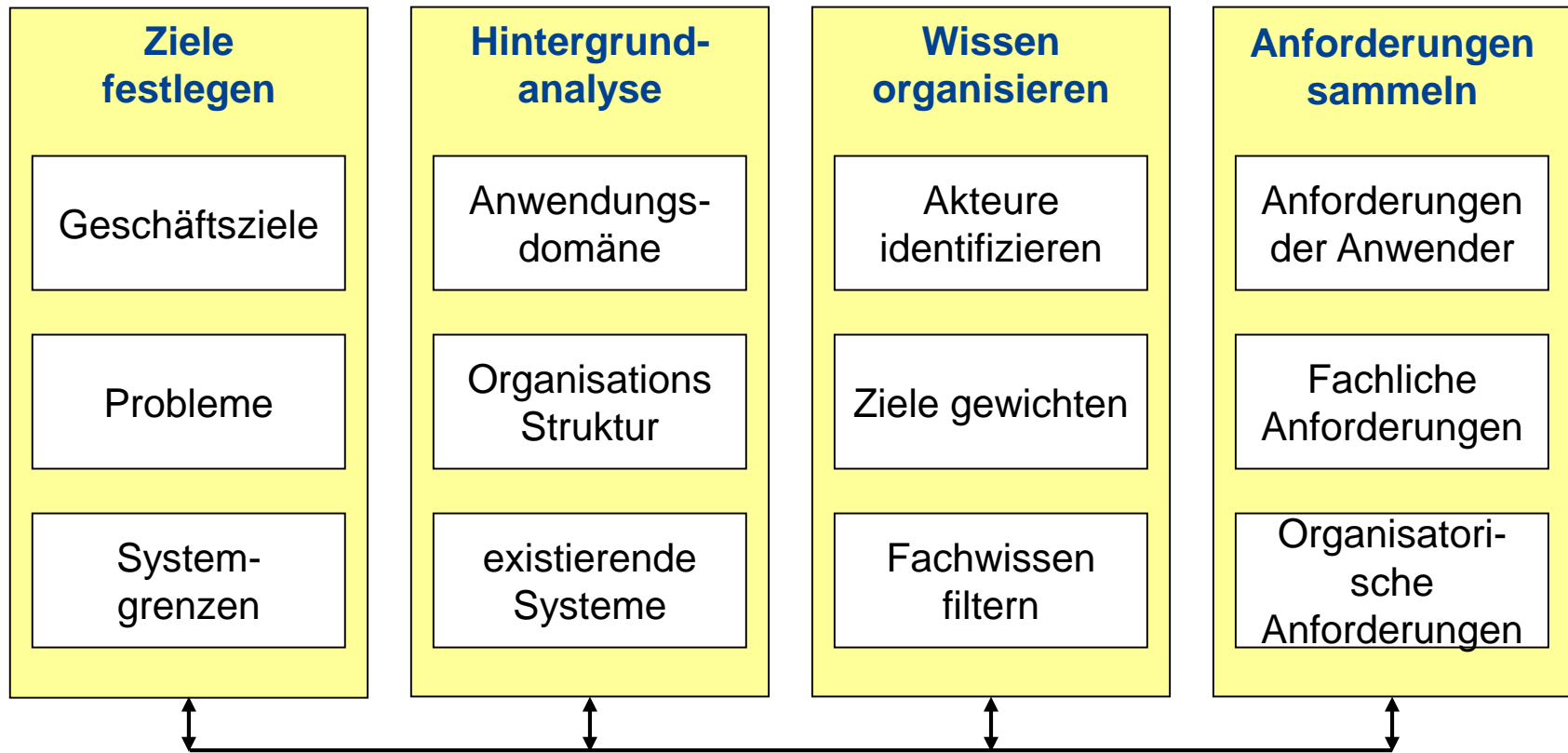


Anforderungserhebung ▶ Erhebungsstufen (1)



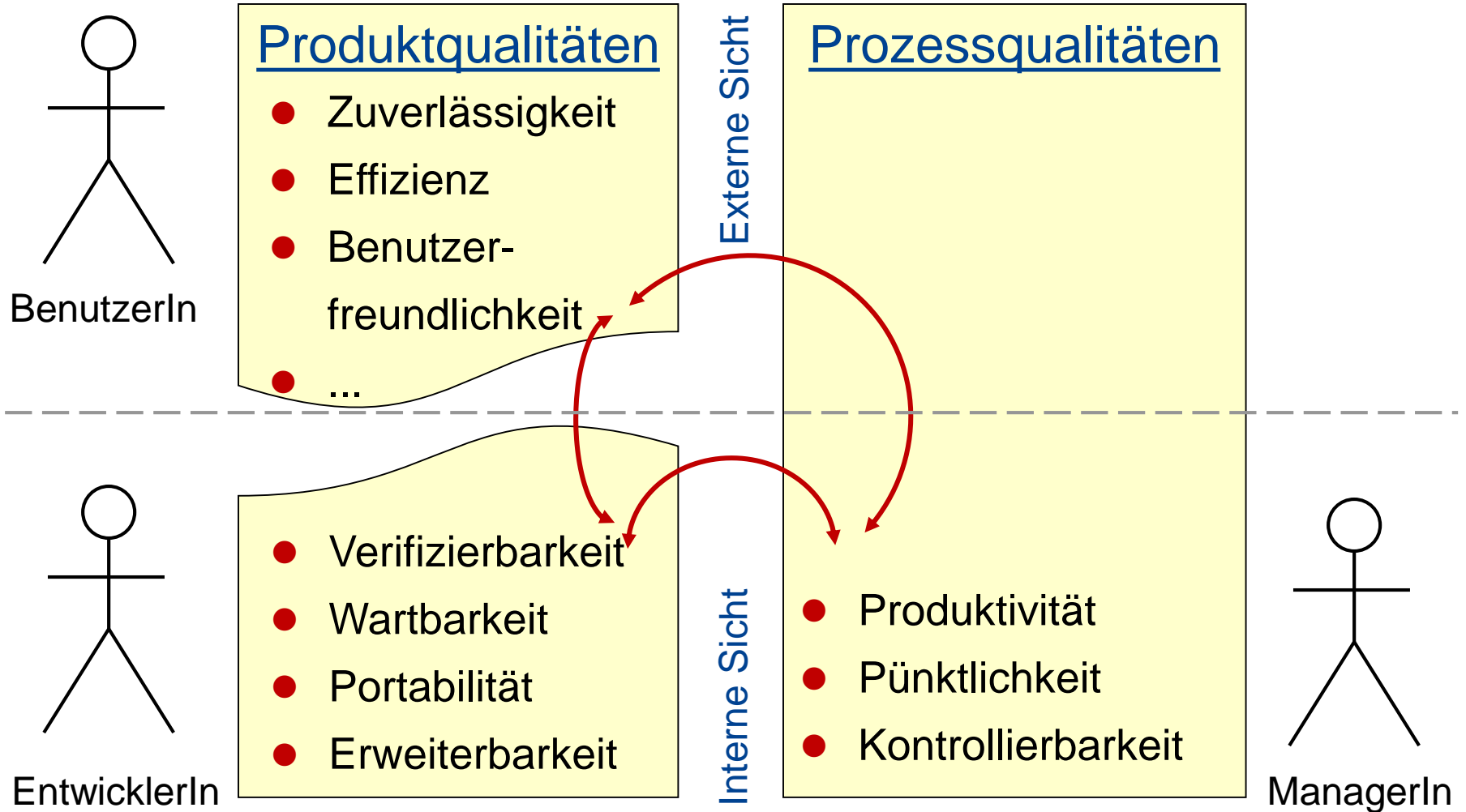
- **Zielsetzung:** Unternehmensziele identifizieren: Warum wird das System gebraucht
- **Hintergrundanalyse:** Sammeln und verstehen von Hintergrundinformationen über das System

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

Qualität ▶ Verschiedene Sichten



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

Arten von Anforderungen

Arten der Anforderungserhebung

Szenarien und Anwendungsfälle („Use Cases“)

Anforderungstypen

- **WAS: Funktionale Anforderungen**

- ◆ Beschreiben die Interaktionen zwischen dem System und seiner Umgebung unabhängig von der Implementierung
 - ⇒ Die Uhr muss die Zeit ortsabhängig anzeigen

- **WIE: 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”)**

- ◆ Alles was uns in der Umsetzung des Was und Wie einschränkt
 - ⇒ Die Implementierung muss in COBOL erfolgen unter Verwendung von DB2.
 - ⇒ Muss mit dem Abfertigungssystem von 1956 zusammenarbeiten.
- ◆ Bedingt durch den Kunden oder der Einsatzumgebung des Systems

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

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.

Arten von Szenarien

Szenarien können während eines Software-Lebenszyklus an vielen Stellen Anwendung finden.

- 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?

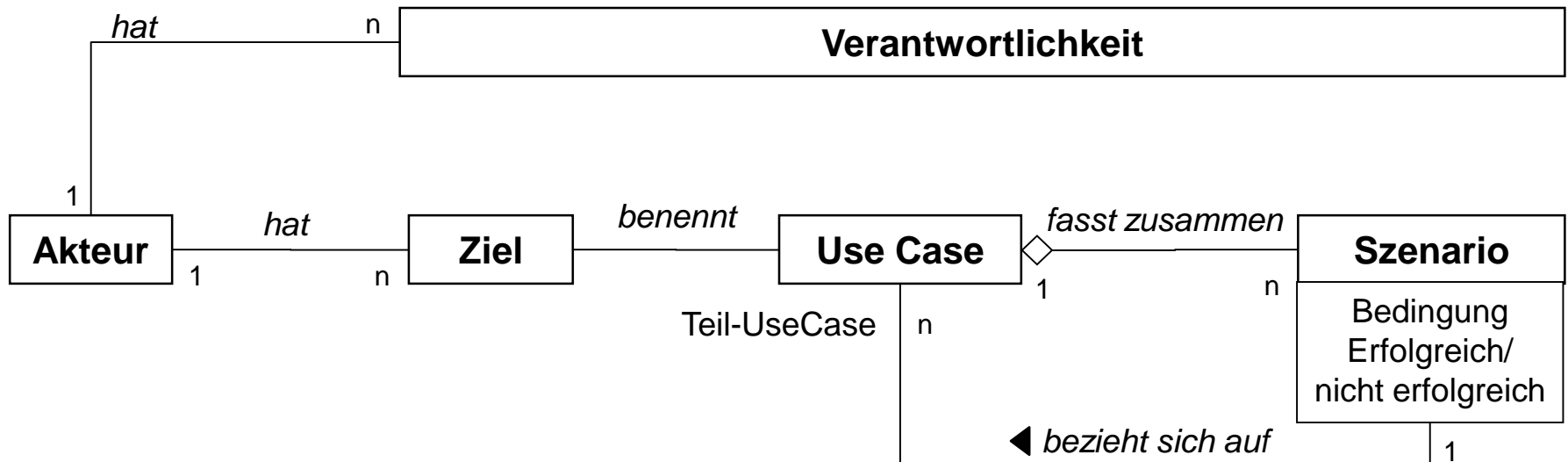
▶ Interview-Techniken

Siehe Exkurs in Ergänzungsfoliensatz „05-exkurs-interviews“.

Beziehungen ▶

Akteur-Ziel-UseCase-Szenario

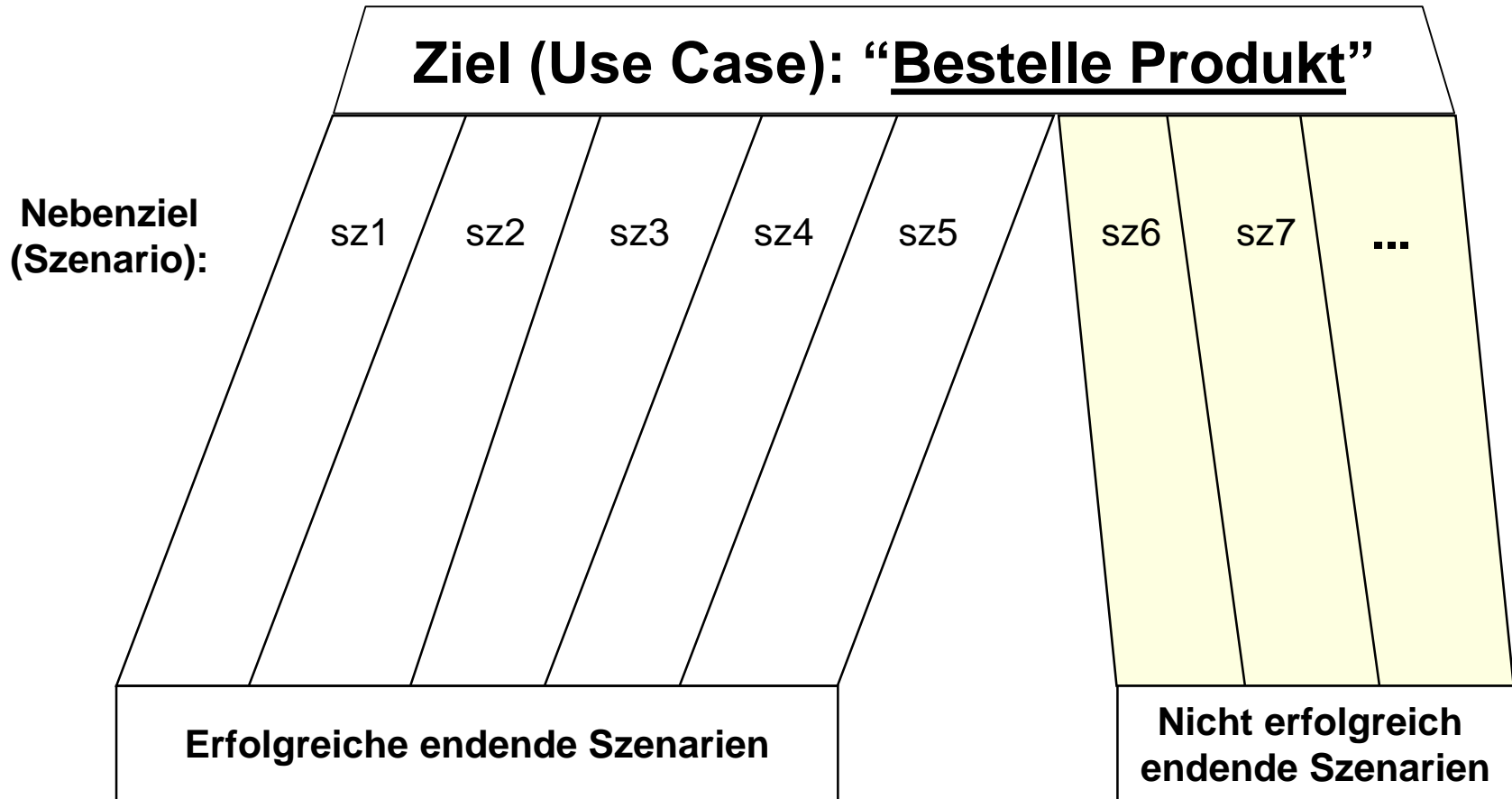
- Ein Akteur hat **Verantwortlichkeiten** aus denen sich **Ziele** ergeben
- Im System umgesetzte **Use Cases** dienen dem Erreichen der **Ziele**
 - ◆ Die Use Cases sind nach den Zielen benannt
- Jeder Use Case fasst eine Menge von **Szenarien** zusammen
- Ein Szenario kann sich auf Teil-Use Cases beziehen.



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)

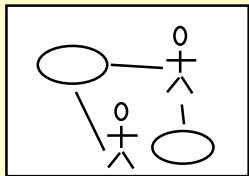


Use-Case-getriebener Softwareentwicklungsprozeß

Requirements Engineering

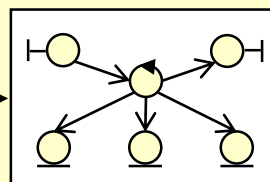
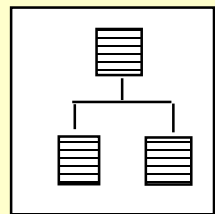
Anforderungs-
erhebung

**Use Case
Model**



Anforderungs-
analyse

ausgedrückt
als

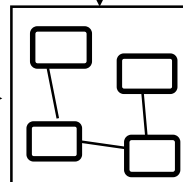


**Domain
Object Model**

**Analysis
Model**

System
Design

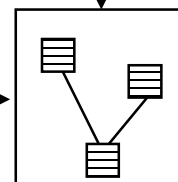
strukturiert
durch



Subsysteme

Object
Design

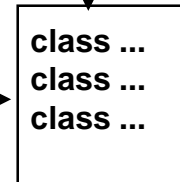
realisiert
durch



**Implementa-
tion Domain
Objects**

Implemen-
tation

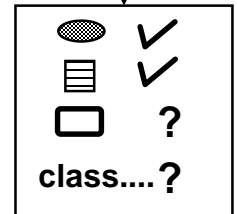
implementiert
von



**Quell
Code**

Testen

verifiziert
durch



**Test
Fälle**

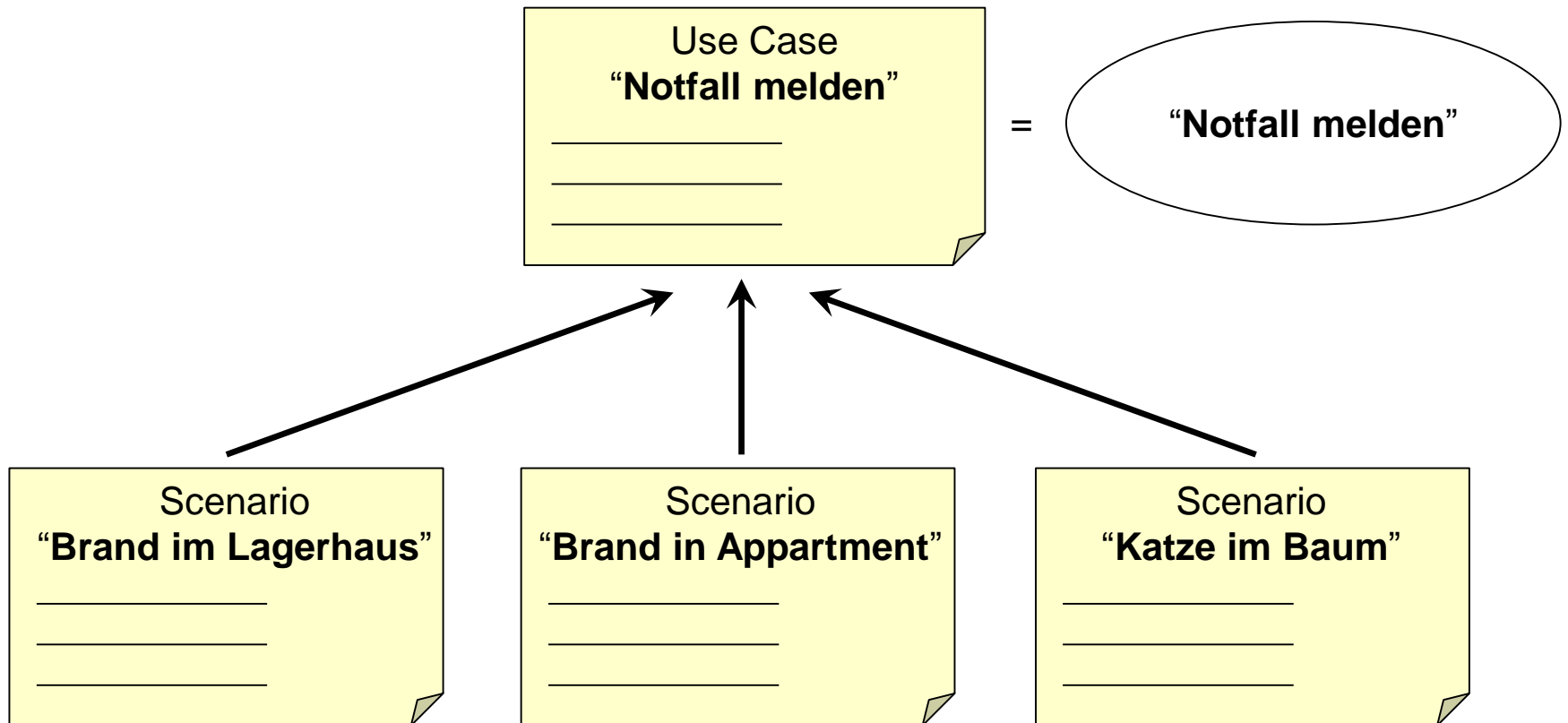
* Die erzeugten dynamischen Modelle (und Andere) sind hier aus Platzgründen nicht explizit dargestellt.

4.2.1 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 **NeuerNotfall** 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“

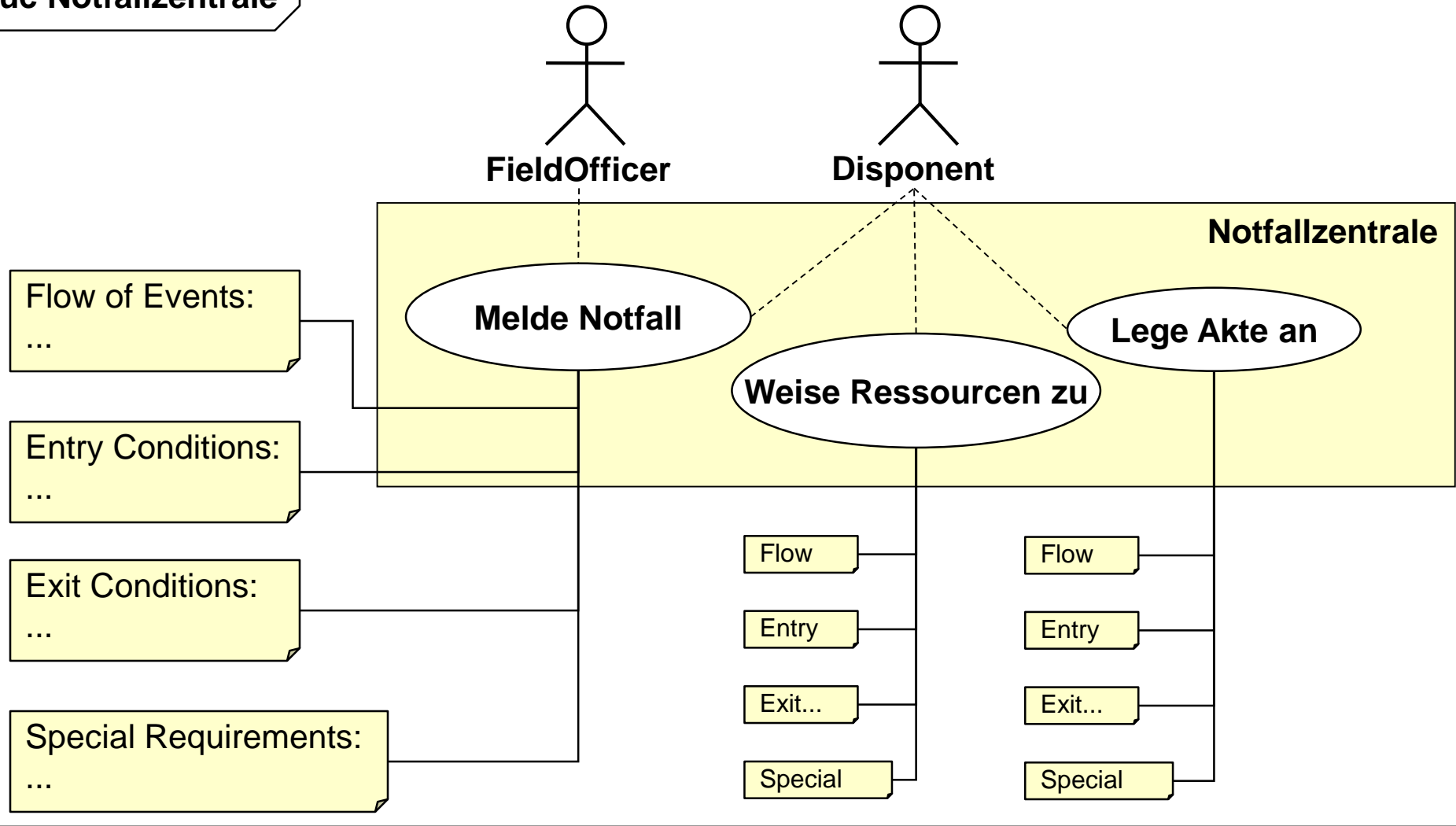
4.3. Anwendungsfalldiagramme ("Use Case Diagrams")

Wir haben nun Use Cases...Viele ...

Wie behalten wir die Übersicht???

Use Case Diagramm „Notfallzentrale“

uc Notfallzentrale



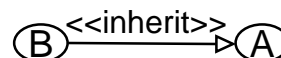
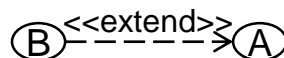
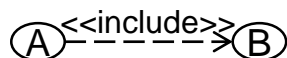
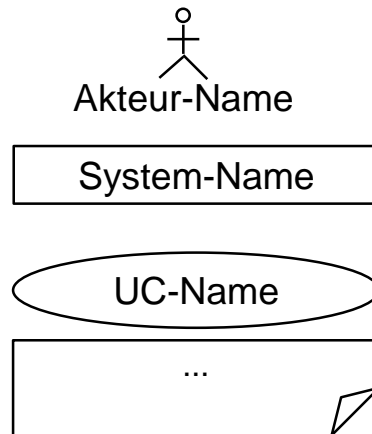
Use-Case Diagramm ▶ Elemente

● Einsatz

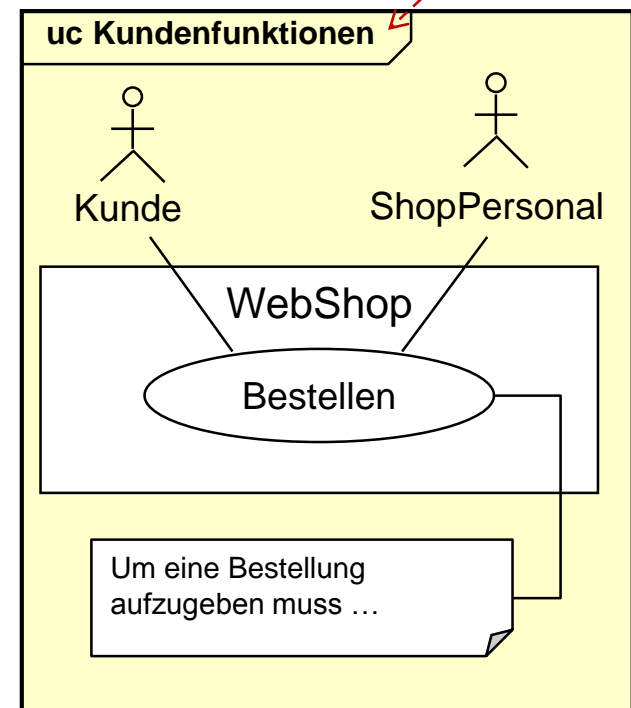
- ◆ Kommunikation mit Kunde / Benutzer während Anforderungserhebung
- ◆ Erfassung von Beziehungen zwischen
 - ⇒ Actors untereinander
 - ⇒ Use Cases untereinander
 - ⇒ Actors und Use Cases

● Elemente

- ◆ Akteur
- ◆ System
- ◆ Use Case
- ◆ Kommentar
- ◆ Beziehungen

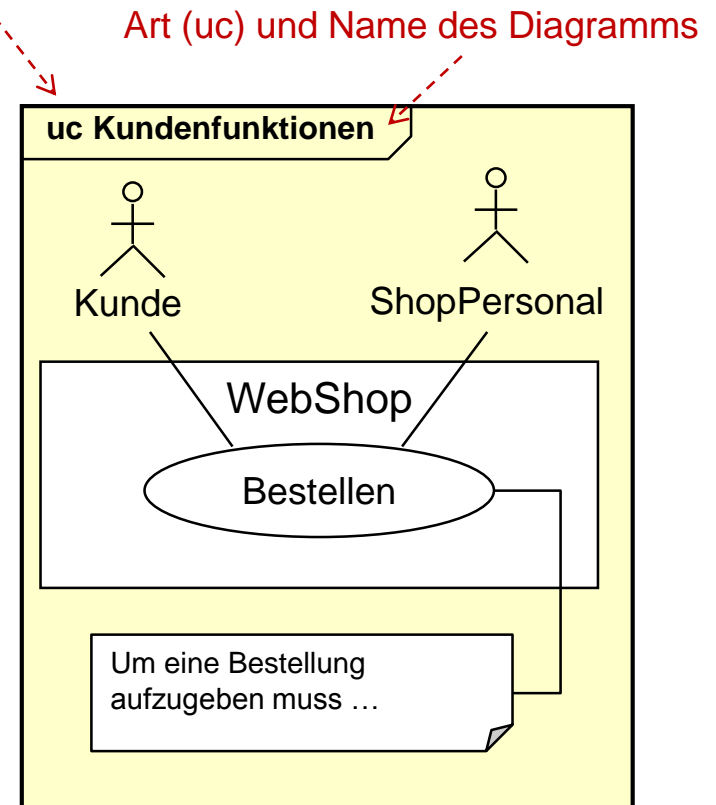


Art (uc) und Name des Diagramms



Use-Case Diagramm ▶ Elemente

- Wie jedes UML-Diagramm ist auch ein Use-Case-Diagramm in einem Rahmen enthalten, in dessen oberen linken Ecke die Diagrammart (hier „uc“ für Use Case) und der Name des Diagramms angegeben ist.
- Der Name des Diagramms ist nicht gleich dem Namen des Systems. Es kann viele Diagramme geben, die Ausschnitte des gleichen Systems darstellen. Z.B. kann das System „WebShop“ dargestellt werden, durch ein Diagramm, das alle UseCases zusammenfasst an denen Kunden und ShopPersonal beteiligt sind, eines für die UseCases an denen nur ShopPersonal beteiligt ist, etc. Es steht uns frei nach welchen Kriterien wir Use Cases in Diagramme gruppieren.
- Im Folgenden werden aus Platzgründen die Rahmen die das Gesamtdiagramm darstellen und manchmal auch die für das System weggelassen. Das gleiche gilt für Akteure, da wo der Fokus des Diagramms auf etwas anderen liegt. Bitte denken Sie trotzdem dran wenn Sie Aufgaben bearbeiten, denn all diese Elemente gehören zu einer korrekten Notation dazu und werden entsprechend bewertet (in den Übungsaufgaben wie in der Klausur)!



Verfeinerung und Strukturierung von Use Cases

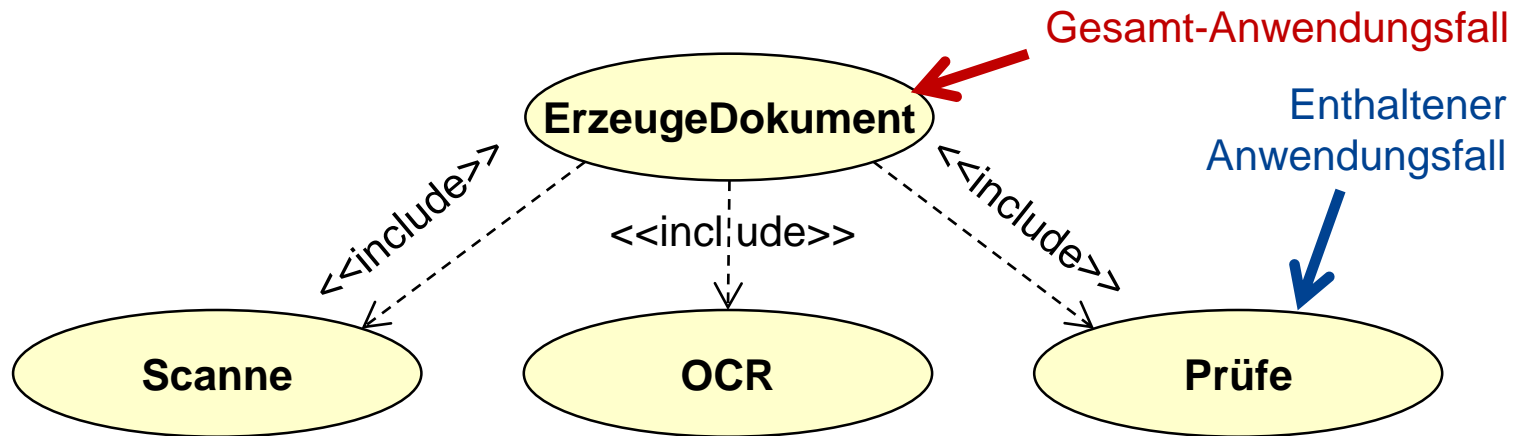
- Ausgangspunkt: Mehrere Use Cases erfasst
- Verfeinere und strukturiere die Use Cases durch Assoziationen
 - ◆ Use Case Assoziation = Beziehung zwischen Use Cases

Beziehungen zwischen Use-Cases

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

<<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

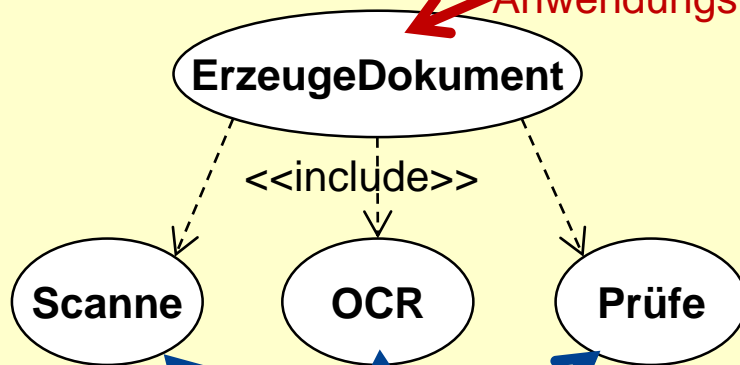
Ein Use Case ist zu komplex und muss zerlegt werden.

Gemeinsame Verwendung

Gemeinsames Verhalten verschiedener Use Cases muss ausgedrückt werden

Scanner

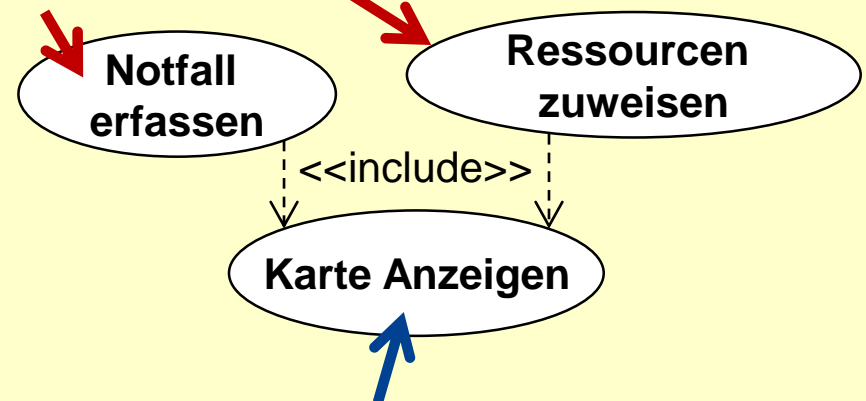
Gesamt-
Anwendungsfall



Enthaltene Anwendungsfälle

Gesamt-
Anwendungsfälle

Notfallzentrale

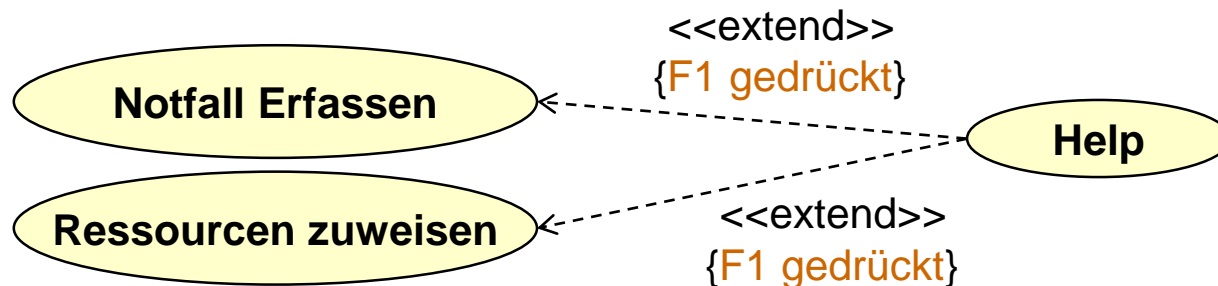


Gemeinsam enthaltener Anwendungsfall



<<extend>>-Beziehung

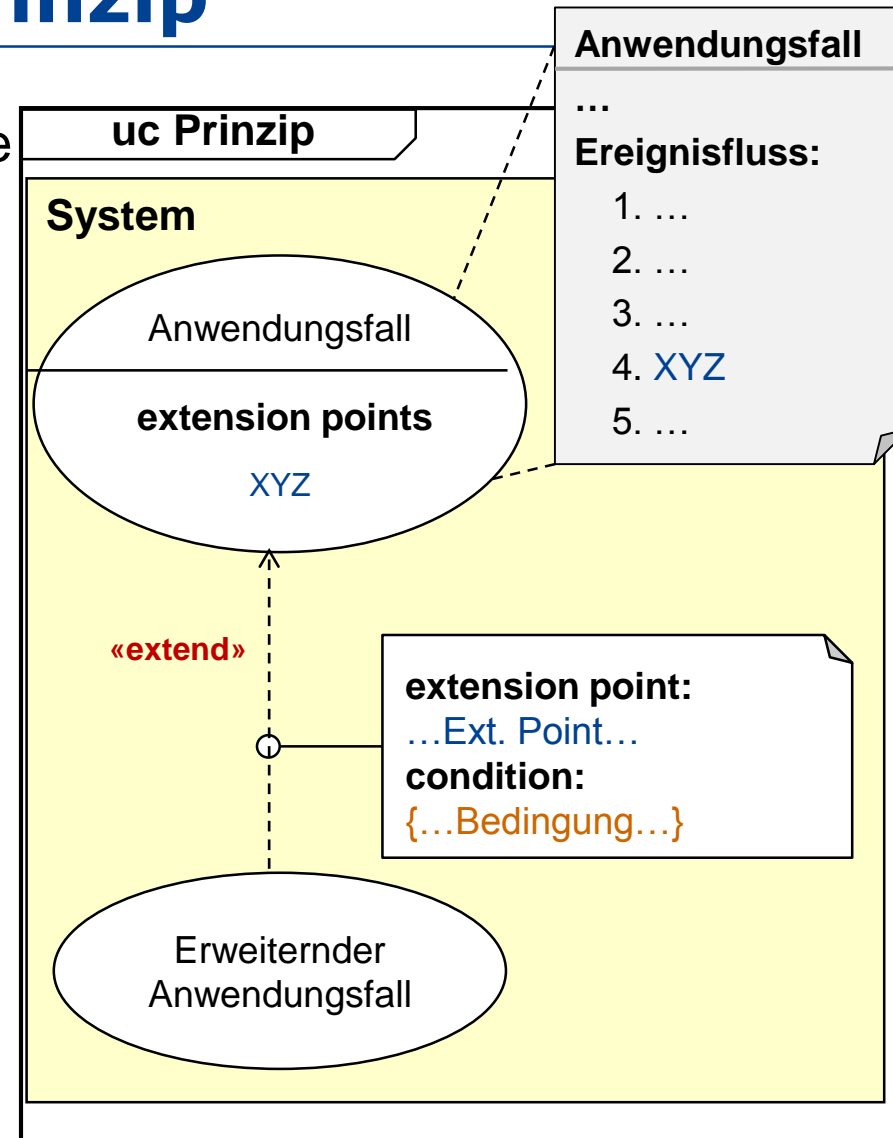
- Problem
 - ◆ Unter bestimmten Bedingungen (Sonderfälle, Fehlerfälle, ...) muss der Ablauf eines Use-Cases erweitert werden
- Lösung / Semantik
 - ◆ Eine extend-Beziehung von Use Case A nach B bedeutet, dass A eine Erweiterung von B ist, die nur unter der **angegebenen Bedingung** aktiv ist.
 - ◆ Der 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: Prinzip

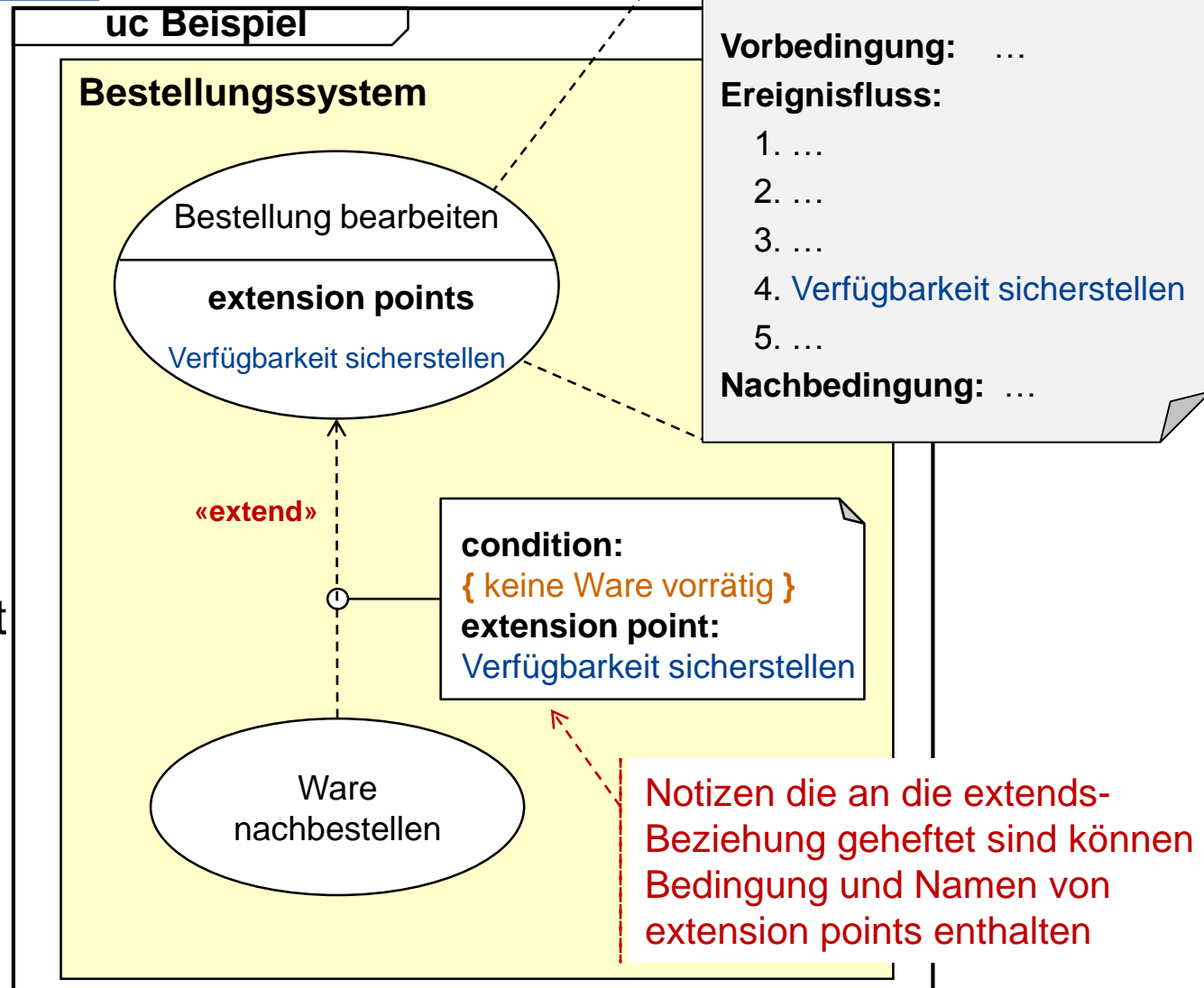
- Deklaration im erweiterten Use Case
 - ◆ Benannte Stellen im **Ereignisfluss** des erweiterten Use Case, wo der Ablauf des erweiternden Use Cases einzufügen ist
- Bezugnahme in <<extends>>-Beziehung
 - ◆ Erweiternder Use Case am **Extension Point** aktiviert
 - ◆ ... falls die spezifizierte **extends-Bedingung**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



<<extend>>-Beziehung

► Extension Points: Beispiel

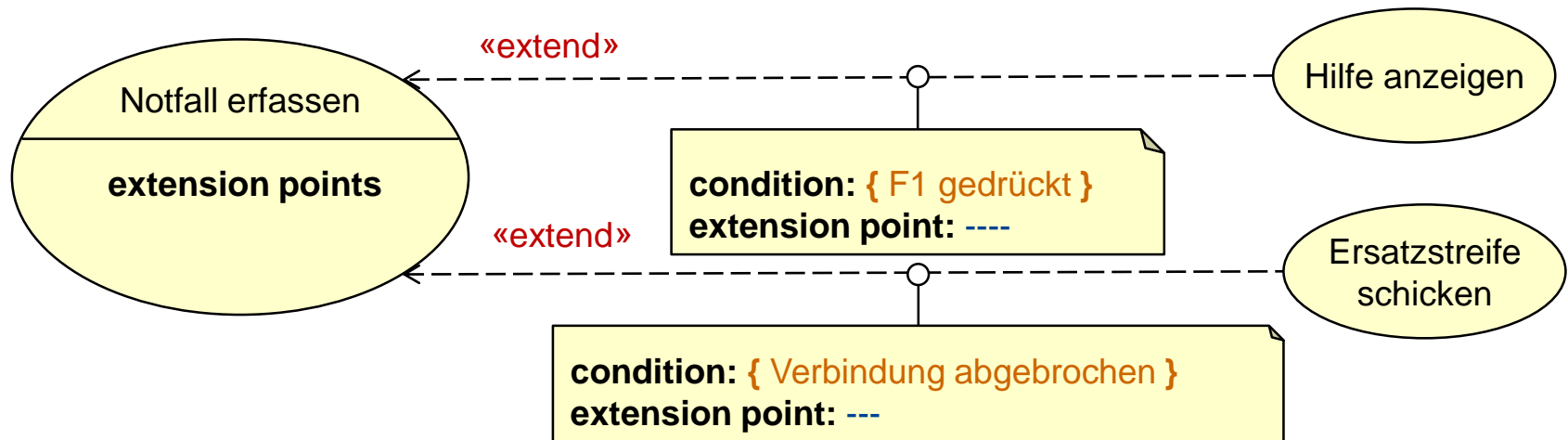
- „Ware nachbestellen“ ist ein Sonderfall
- ... der an der Stelle „Verfügbarkeit sicherstellen“ im Ereignisfluss von „Bestellung bearbeiten“ auftritt
- ... falls dann die Bedingung „keine Ware vorrätig“ gilt



<<extend>>-Beziehung

▶ Keine Extension Points für „Ereignisse“

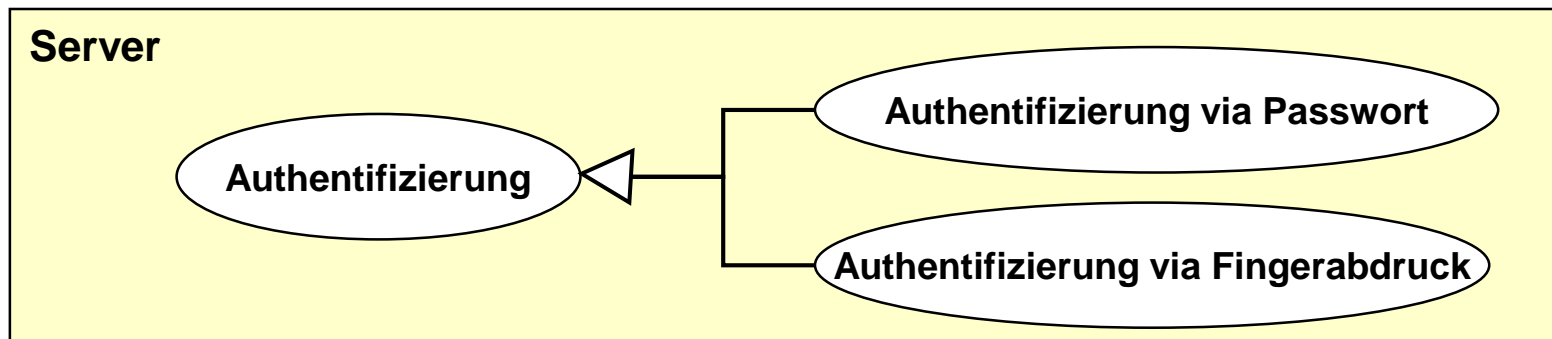
Wenn der Auslöser ein „Ereignis“ ist, werden Extension Points weggelassen, denn Ereignisse können jederzeit auftreten:



Merke: Die Bedingung kann nicht weggelassen werden!

Generalisierung von Use Cases

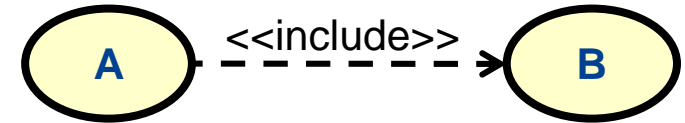
- Ziel
 - ◆ Modellieren, dass verschiedene Use Cases Varianten eines allgemeineren Ablaufs sind
- Prinzip
 - ◆ Die „Kinder“ erben die Kommunikationsbeziehungen, die Bedeutung und das Verhalten der „Eltern“, modifizieren es aber im Detail.
- Beispiel
 - ◆ „Authentifizierung“ prüft die Benutzeridentität. Der Kunde könnte zwei Umsetzungen fordern: „Via Passwort“ und „Via Fingerabdruck“



Beziehungen zwischen Use-Cases

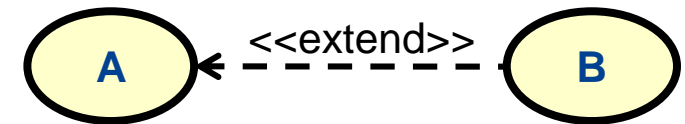
● Include-Beziehung

- ◆ Immer wenn **A** ausgeführt wird, **muss** auch **B** ausgeführt werden
- ◆ **B** ist unbedingt nötig, um **A** auszuführen



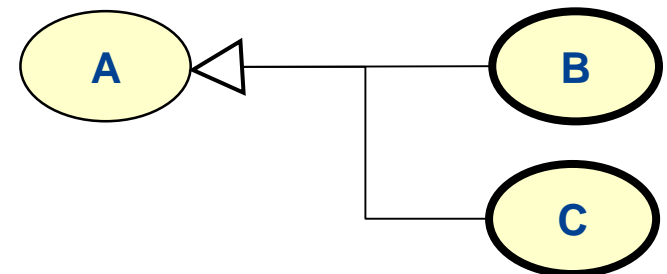
● Extend-Beziehung

- ◆ Wenn **A** ausgeführt wird, **kann** auch **B** ausgeführt werden
- ◆ **B** ist nicht zwingend nötig, um **A** auszuführen

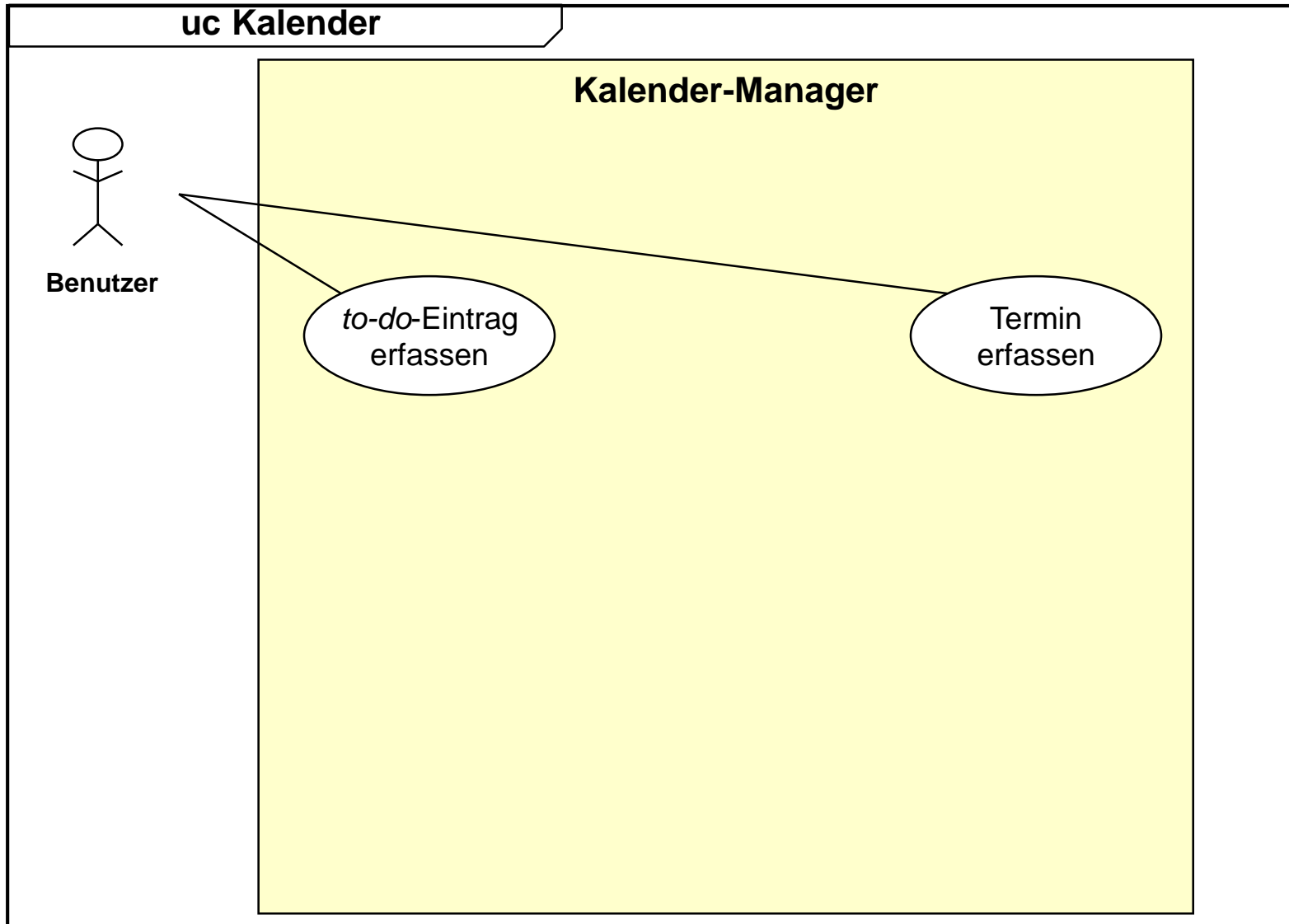


● Generalisierungs-Beziehung

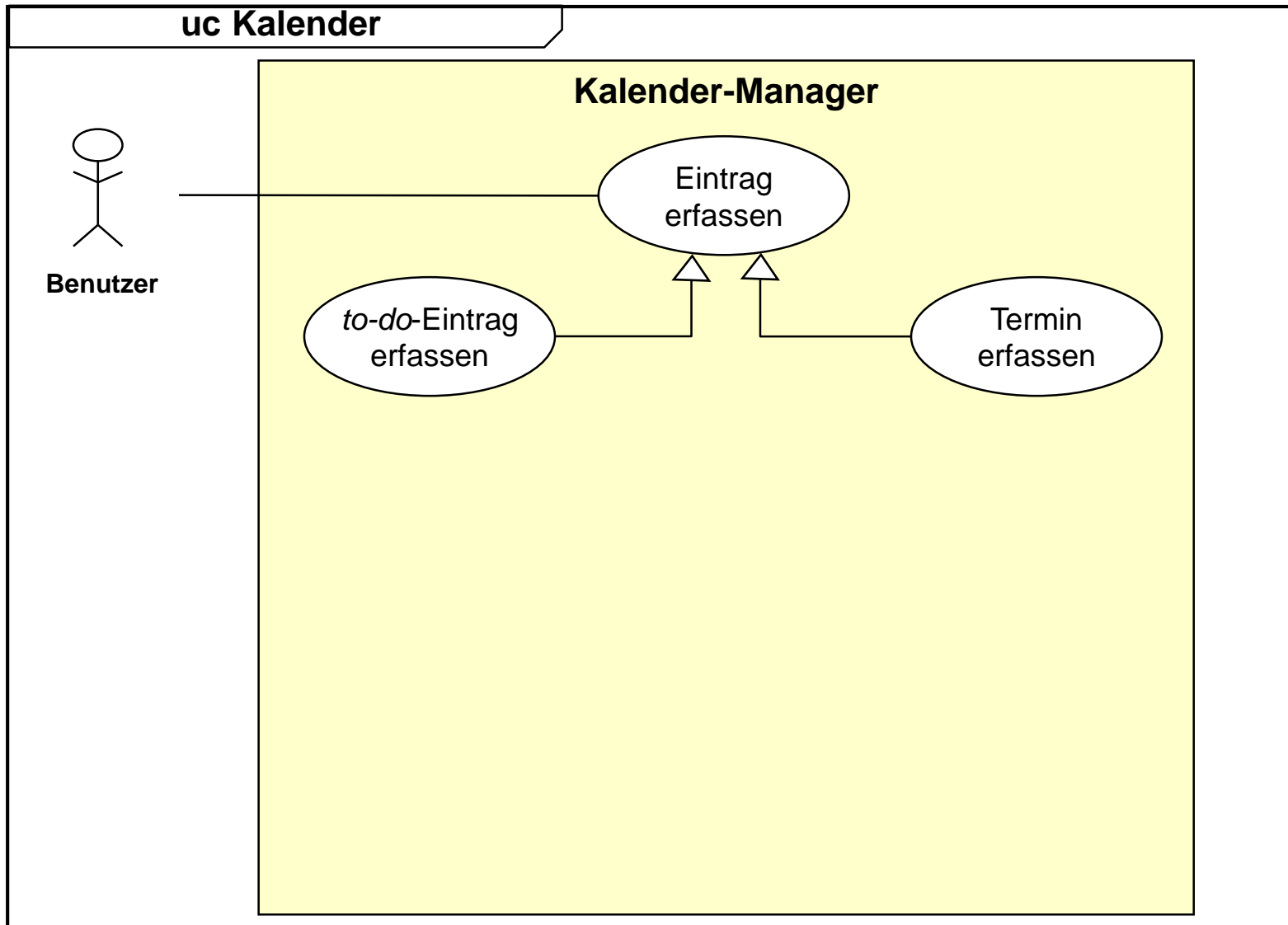
- ◆ **B** und **C** sind jeweils spezielle Ausprägungen des **gesamten** Ablaufs von **A**
- ◆ **Nur eine** der verschiedenen Spezialisierungen wird durchgeführt, die aber **komplett**



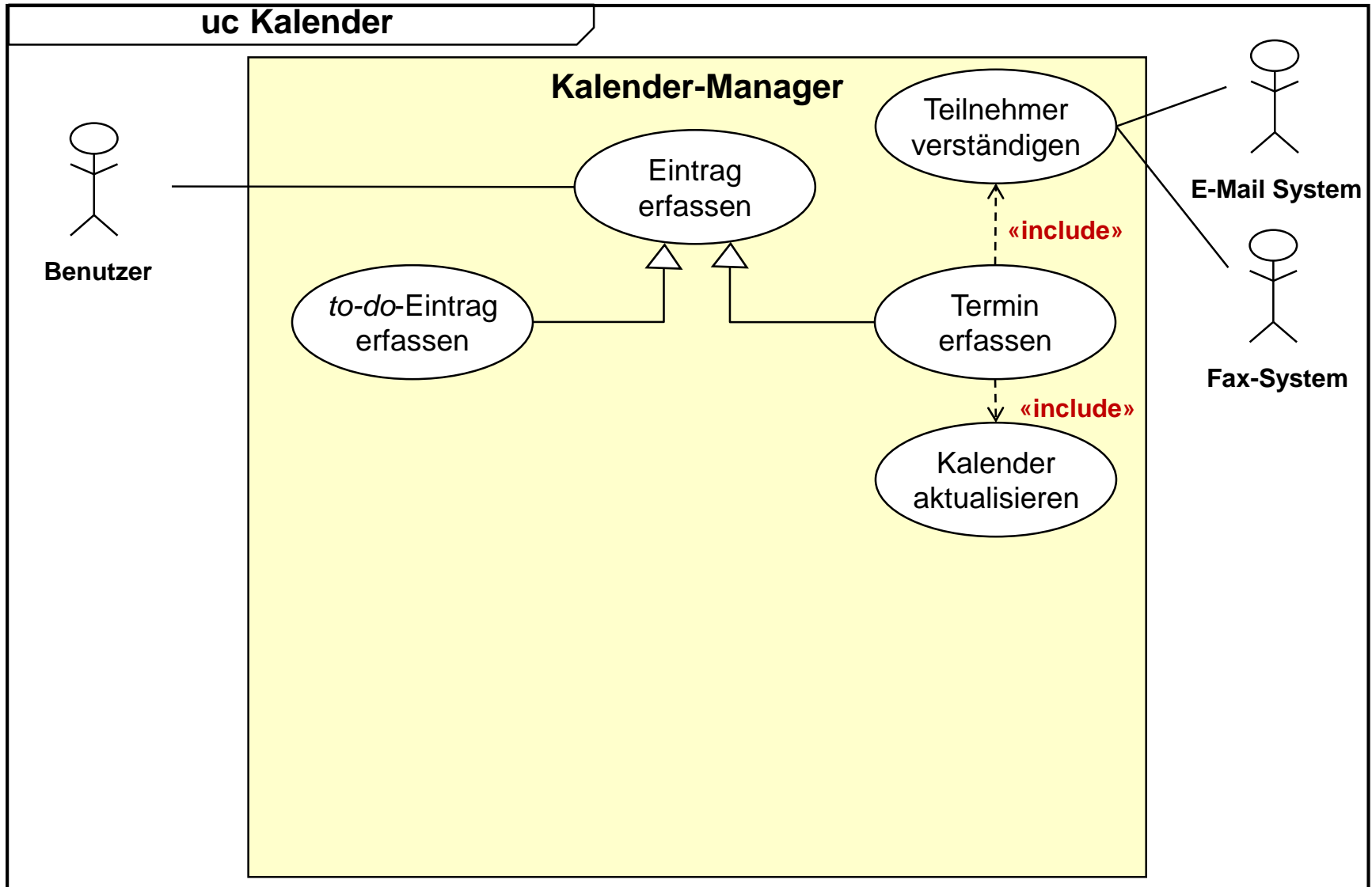
Beispiel „Kalender-Manager“



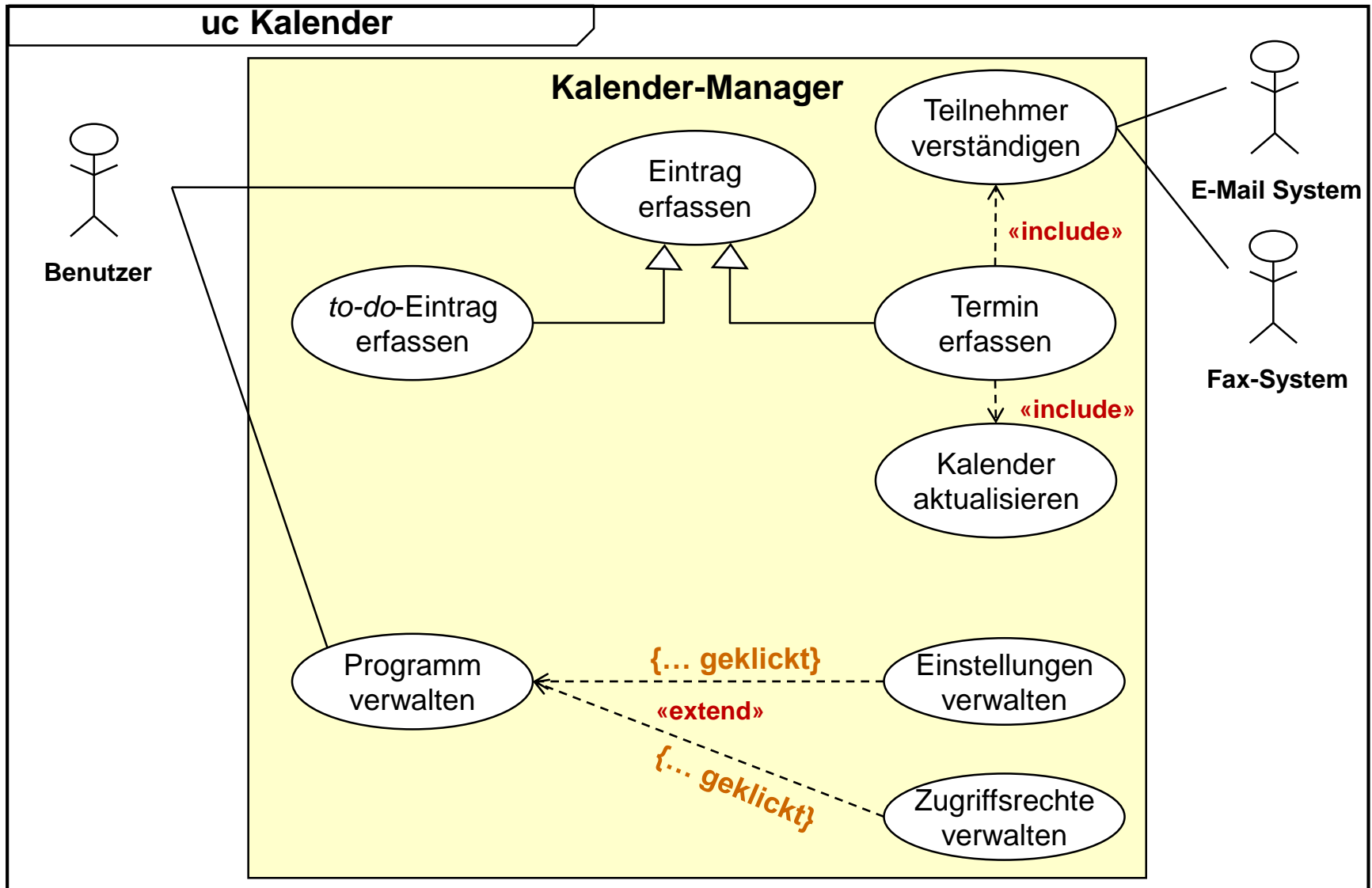
Beispiel „Kalender-Manager“ ▶ Varianten der Eintragungserfassung als Generalisierung



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

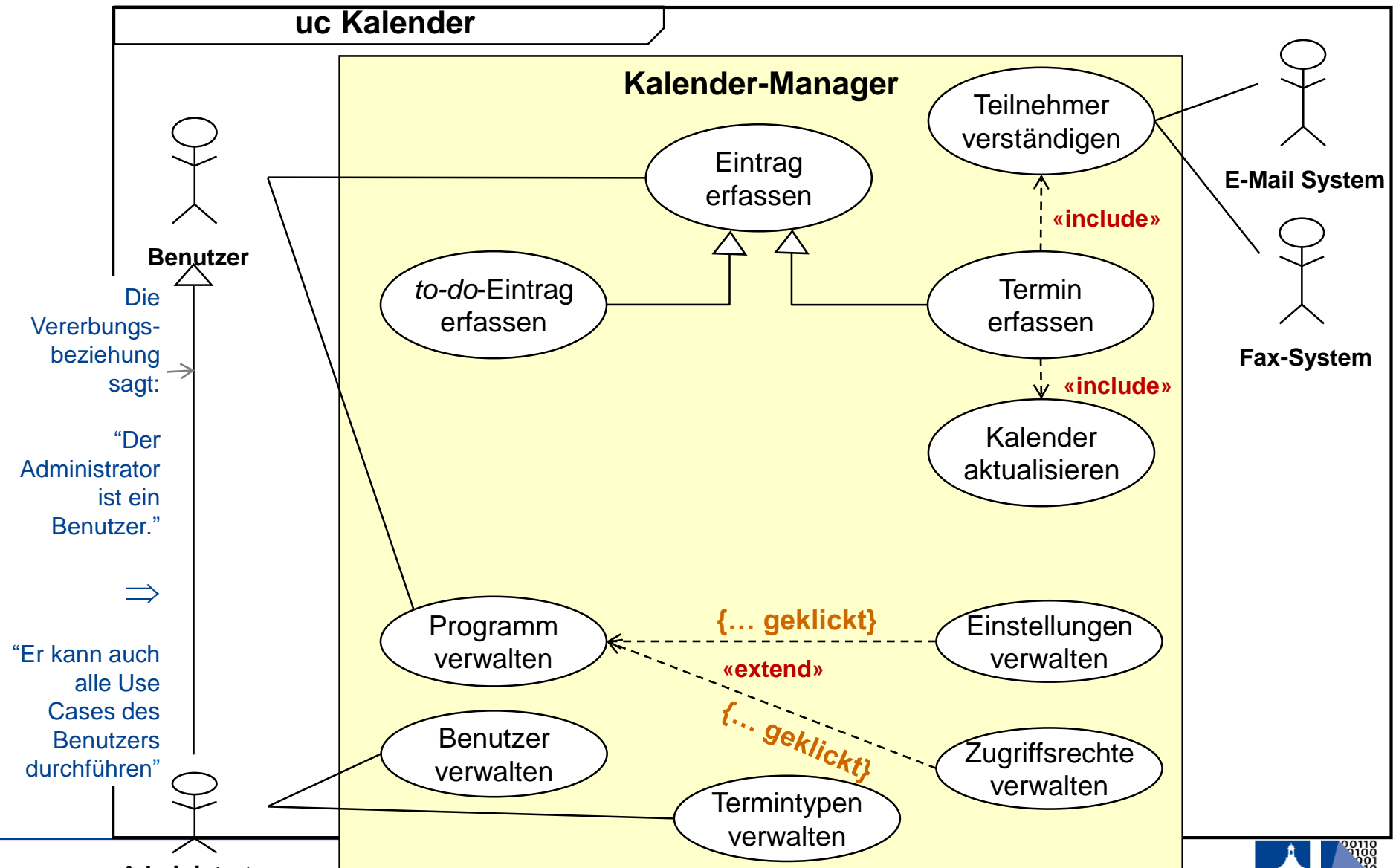


Beispiel „Kalender-Manager“ ▶ <<extend>> für Sonderfälle der Programmverwaltung



Beispiel „Kalender-Manager“ ▶

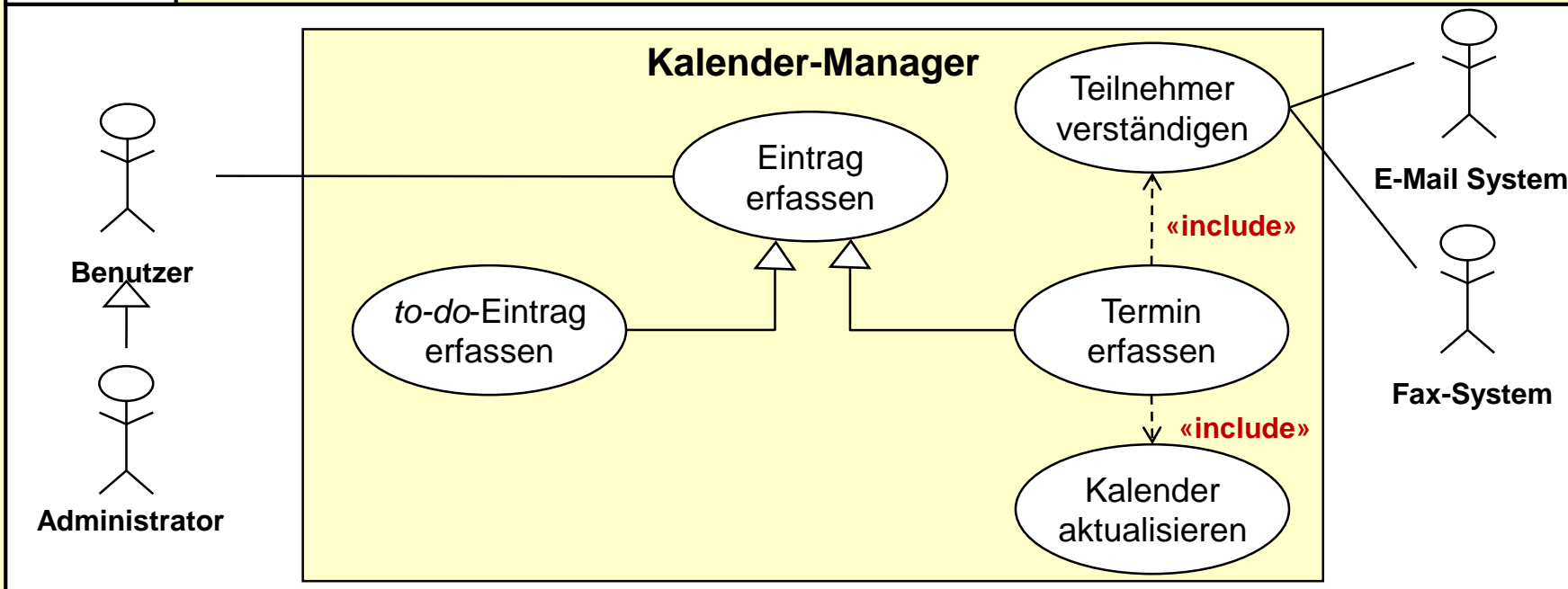
Generalisierung zwischen Akteuren



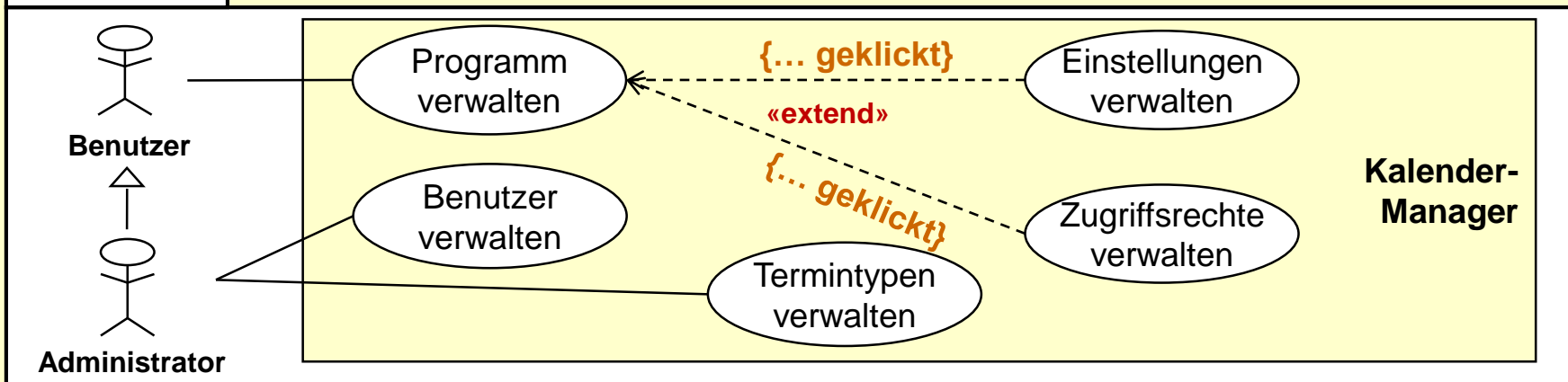
Beispiel ▶ Modularisierung durch Packages

uc Kalender

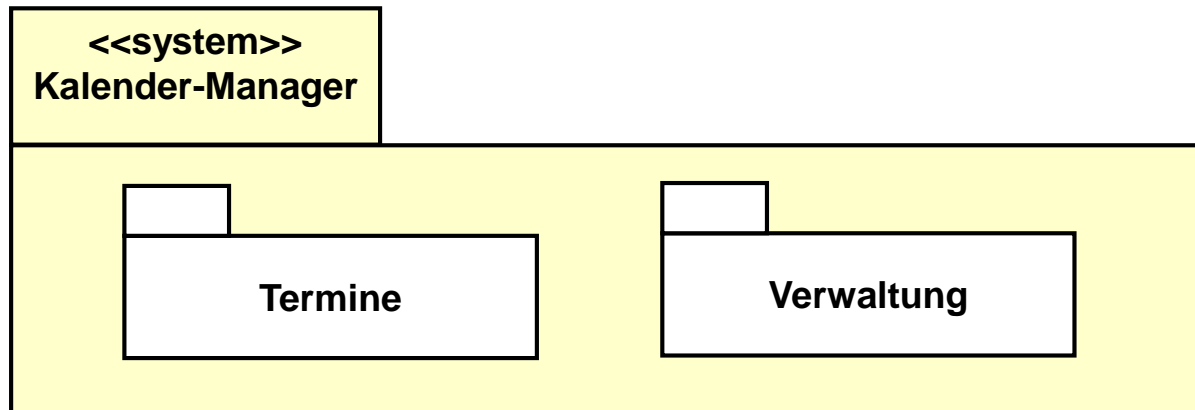
Termine



Verwaltung



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

4.4 Domain Object Model

Abbott's Technik zur Objektidentifikation
Elementare Objektmodellierung
Beispiele

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
- } Input für
- 3. Entwickle das Objektmodell
 - ◆ Entwickle Klassendiagramme, die die Anwendungsdomäne 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

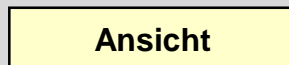
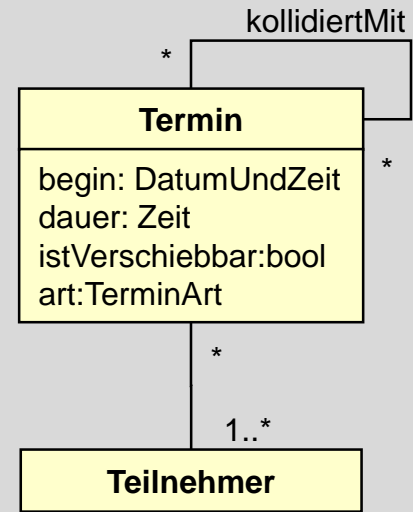
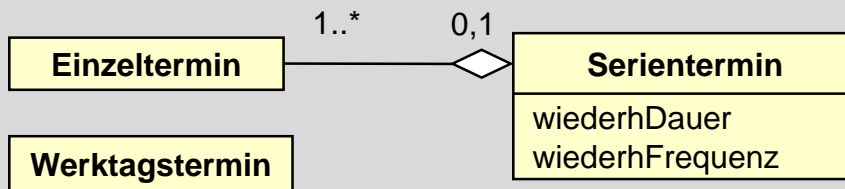
Domain Object Model (DOM)

- 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)
 - ◆ Anschließend Textanalyse nach Abbott → Initiales Klassendiagramm
 - ◆ Anschließend Verfeinerung des Modells anhand allgemeiner Modellierungsprinzipien (s. nächstes Kapitel)

Domain Object Model ▶ Beispiel „Terminverwaltung“

class Terminverwaltung



Objekt-Identifikation

- 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 Objektidentifikation
 - ◆ 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

Finde für jeden Use Case Begriffe die Nutzer oder Entwickler klären müssen um den Ereignisfluss zu verstehen:

1. **Entitäten**, die das System im Auge behalten muss.
Beispiele: `FieldOfficer`, `Disponent`, `Resource`
2. **Vorgänge**, die das System im Auge behalten muss.
Beispiel: `Notfallplan`
3. **Datenquellen** oder **-senken**. Beispiel: `Drucker`
4. **Schnittstellen**. Beispiel: `PoliceStation`
5. Führe eine Textanalyse zum Finden zusätzlicher Objekte durch (Abbott's Technik)
6. 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 betritt 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“ ≈ ...

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

Aus Platzgründen wird der Rahmen mit Bezeichner hier weggelassen

Pfandrückgabemaschine

Getränkebehälter

Behältertyp

Pfandbetrag

Dose

Flasche

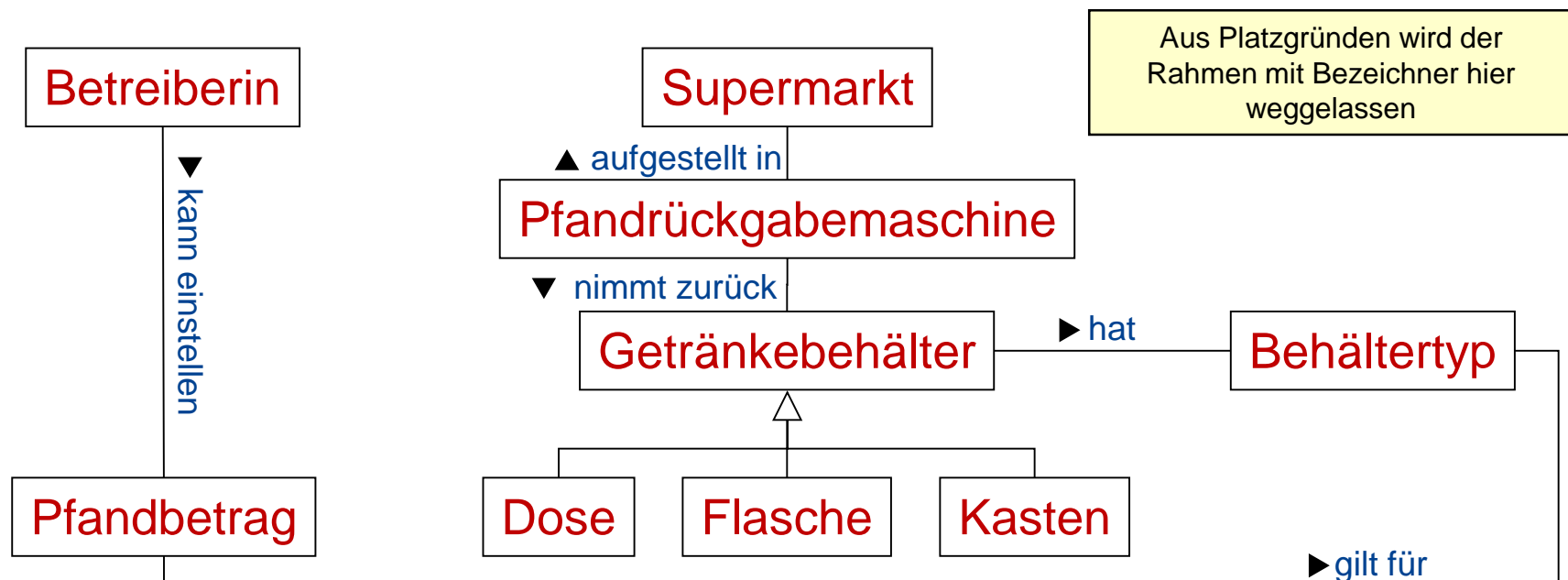
Kasten

Beispiel ▶ Textanalyse nach Abbott

▶ Assoziationen extrahieren

- Substantive + Verben + Attribute → Klassen + Assoziationen

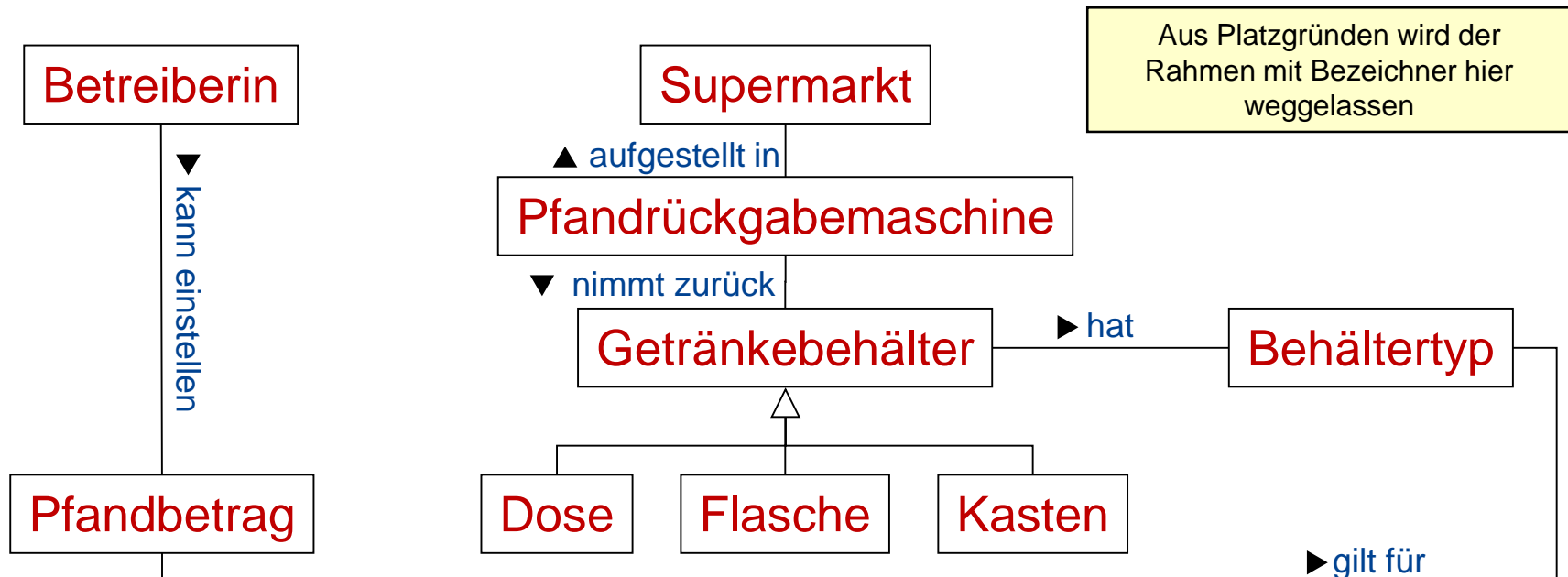
„Die Pfandrückgabemaschine (PRM) wird in Supermärkten aufgestellt um wiederverwendbare Getränkebehälter zurückzunehmen (leere Dosen, Flaschen und Kästen, welche Endbenutzern geliefert werden). Für jeden Behältertyp kann unsere Betreiberin einen individuellen Pfandbetrag einstellen.“



Beispiel ▶ Schematisch erstelltes Domain Object Model überdenken (1)

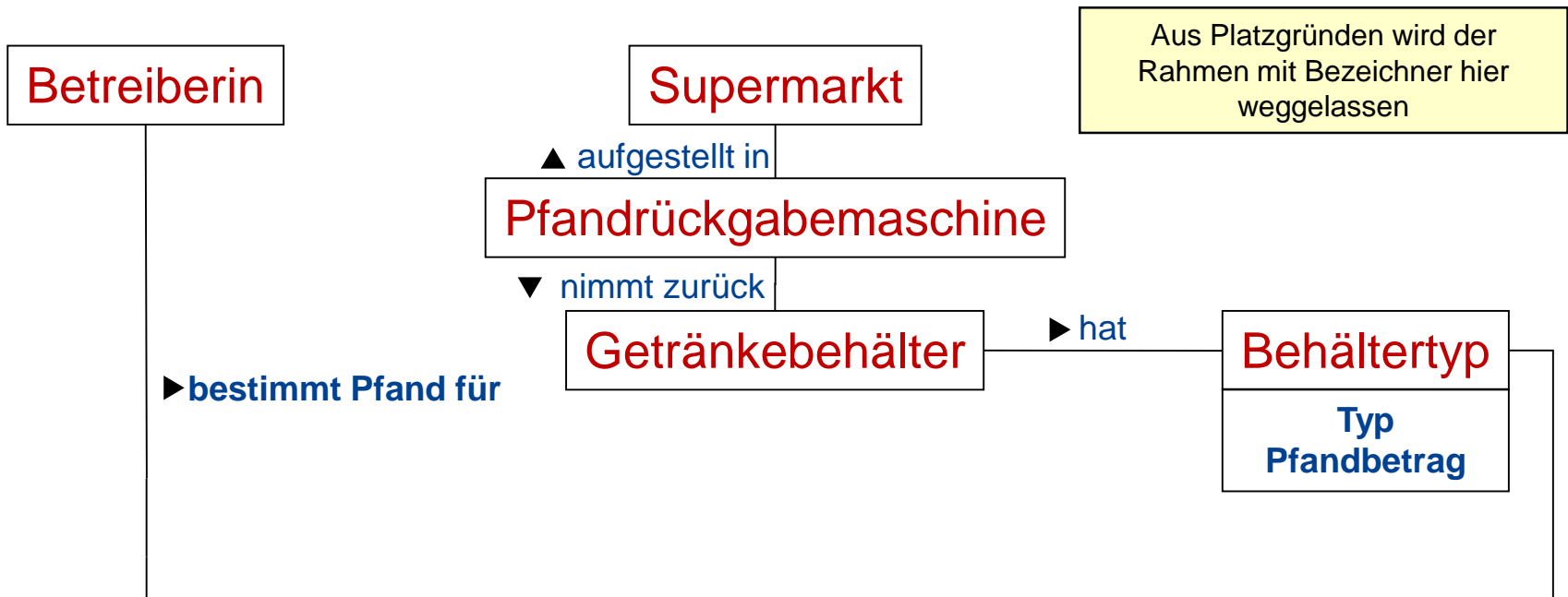
- Vererbungshierarchie überdenken
 - ◆ Dosen, Flaschen und Kästen sind doch eigentlich Behältertypen!
 - ◆ Die Klasse „Behältertyp“ oder die Unterklassen von „Getränkebehälter“ sind redundant.
- Entscheidungshilfen
 - ◆ Welche Klassen besitzen **kein eigenes Verhalten**?
 - ◆ Welche Klassen liefern als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



Beispiel ▶ Schematisch erstelltes Domain Object Model überdenken (3)

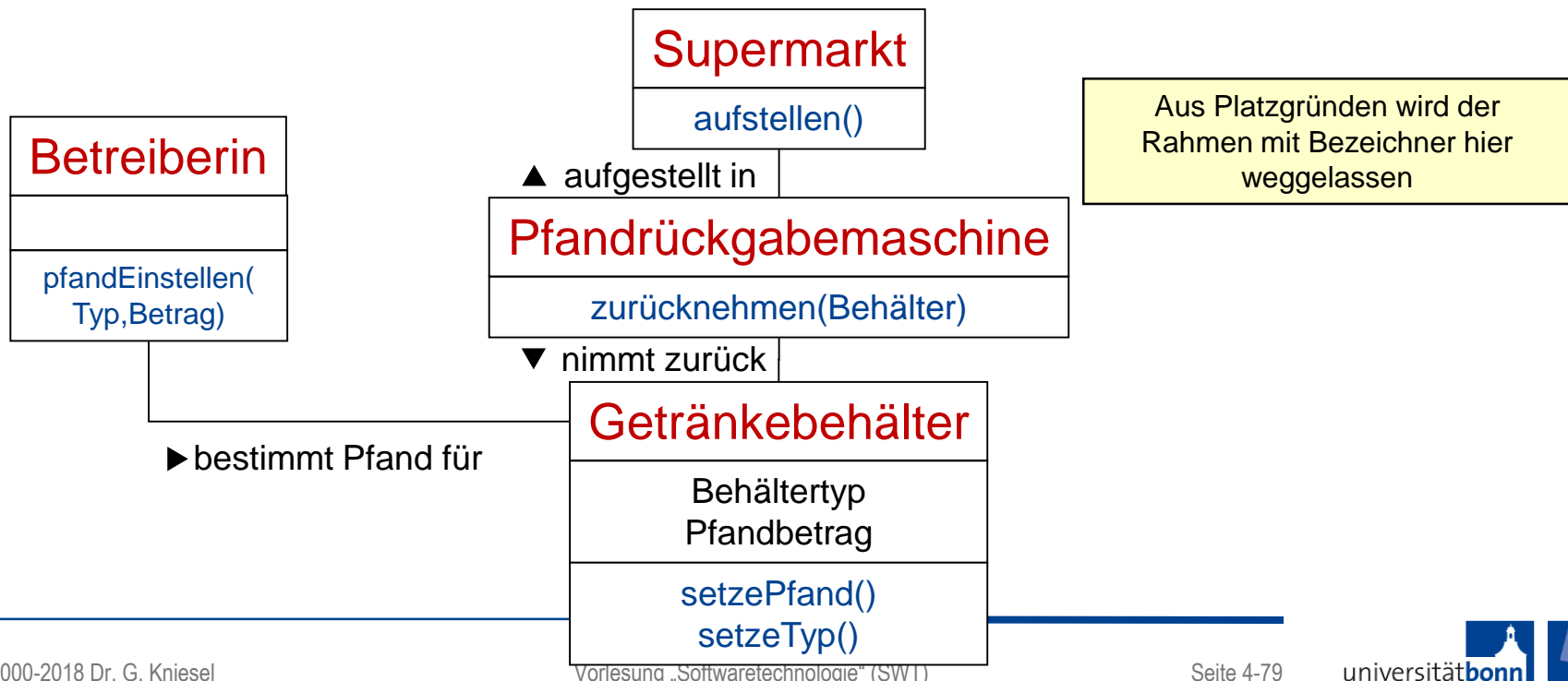
- Korrektur-Maßnahmen: „Inline Class“
 - ◆ Überflüssige Klasse (→ „Behältertyp“) identifizieren und ihre Elemente in eine per Assoziation verbundene Nachbarklasse einbetten
- Anwendung des Prinzips in unserem Beispiel
 - ◆ Die Attribute „Name“ und „Pfandbetrag“ wandern in „Getränkebehälter“
 - ◆ „Name“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

Aussage	Wort	Wortart	Modellelement
PRM nimmt Getränkebehälter zurück	zurücknehmen	Verb	Methode " <u>accept</u> "
PRM akzeptiert Dosen, Flaschen, Kästen	PRM	Nomen (Subjekt)	Klasse "PRM" Empfänger der " <u>accept</u> " Nachricht
PRM akzeptiert Dosen, Flaschen, Kästen	Dosen, Flaschen, Kästen	Nomen (Objekte)	Klassen Parameter der Methode " <u>accept</u> "
PRM druckt Quittung	druckt	Verb	Methode " <u>printReceipt</u> "
PRM erzeugt Bericht	erzeugt	Verb	Methode " <u>generateReport</u> " Report <u>generateReport()</u>
Behältertyp	Behältertyp	Nomen	Klasse "Container"
Getränkebehälter (Dose, ...)	???	???	???
Dose ist Behältertyp	ist	Verb	Behälter ist Generalisierung von Dose
Flasche ist Behältertyp	ist	Verb	Behälter ist Generalisierung von Flasche
Kasten ist Behältertyp	ist	Verb	Behälter ist Generalisierung von

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 weitergedacht

class Supermarkt-Pfandrücknahme

Betreiberin

pfandEinstellen(
Typ, Betrag)

Supermarkt

aufstellen(PRM)
getPRM():PRM

aufgestellt in

Die Betreiberin hat nun keine Assoziation mehr zum Getränkebehälter. Sie braucht daher die Methode getRPM(), um an die Pfandrückgabemaschine(n) zu gelangen, denen sie neue Behälterpreise setzen soll.

Behältertyp

Name
Pfandbetrag

Pfandrückgabemaschine

zurücknehmen(Behälter)
reportErzeugen : Report
quittungDrucken
pfandEinstellen(Behältertyp, Betrag)

Bericht

/summe : Double
/Erstattungen
summe():Double

◀ ist vom typ

Alle Getränkebehälter-Instanzen, die Dosen sind, können nun mit der selben Instanz von Behältertyp assoziiert werden, so dass es keine Redundanzen mehr gibt -> Der Pfandbetrag für Dosen wird in genau einem Behälter-Objekt verwaltet.

Bestand

Getränkebehälter

getPfand(Double)
setzeTyp(Behältertyp)

Erstattungen

Quittung

/summe : Double
summe():Double

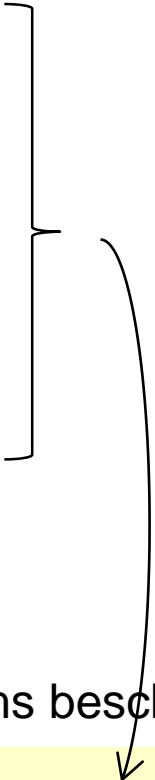
BarcodeReader

scannQuittung:Quittung

4.5 Dynamische Modellierung von Anwendungsfällen

Wir haben die Use Cases und nun auch ein statisches Modell
→ Aber was ist mit dem Verhalten?

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
 - ◆ Interaktionsdiagramme für das Zusammenspiel von Objekten
 - ◆ Zustandsdiagramme für die internen Abläufe von Objekte mit interessantem Verhalten
 - ◆ Aktivitätsdiagramme für Geschäftsprozesse
- Input für
- 

Spielzeugauto ▶ Problembeschreibung

- Strom wird angeschaltet
→ Auto fährt **vorwärts**

-  Scheinwerfer leuchtet

- Strom wird ausgeschaltet

-  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

-  Auto hält an

-  Scheinwerfer geht aus

- Strom wird angeschaltet

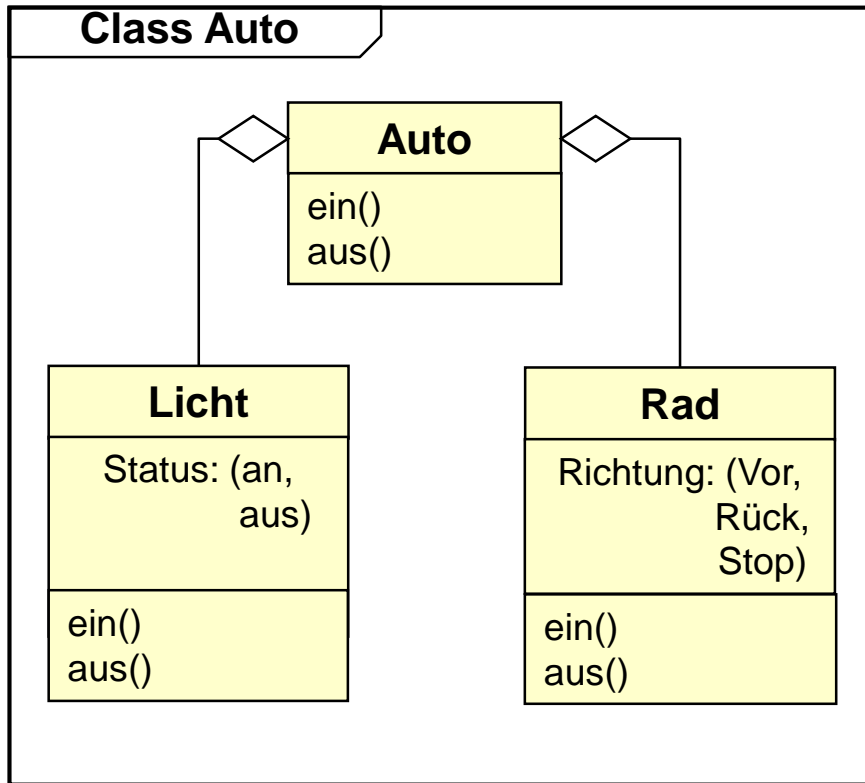
-  Scheinwerfer leuchtet

- Strom wird ausgeschaltet

-  Scheinwerfer geht aus

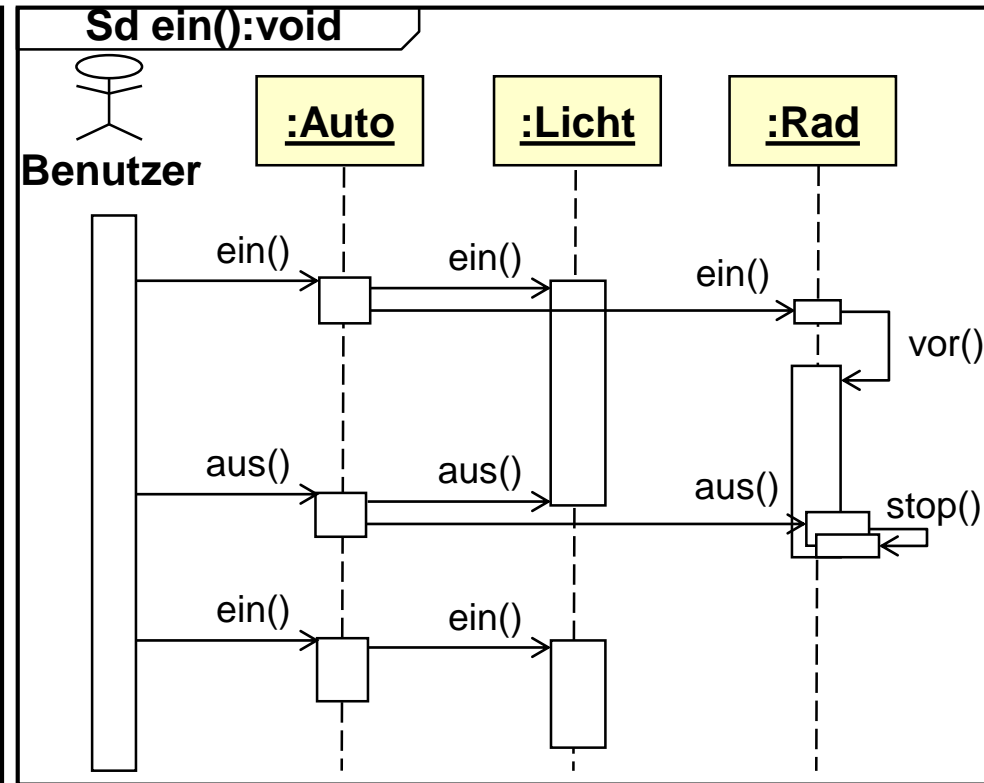
Spielzeugauto ▶ Verschiedene Modelle

Klassendiagramm



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

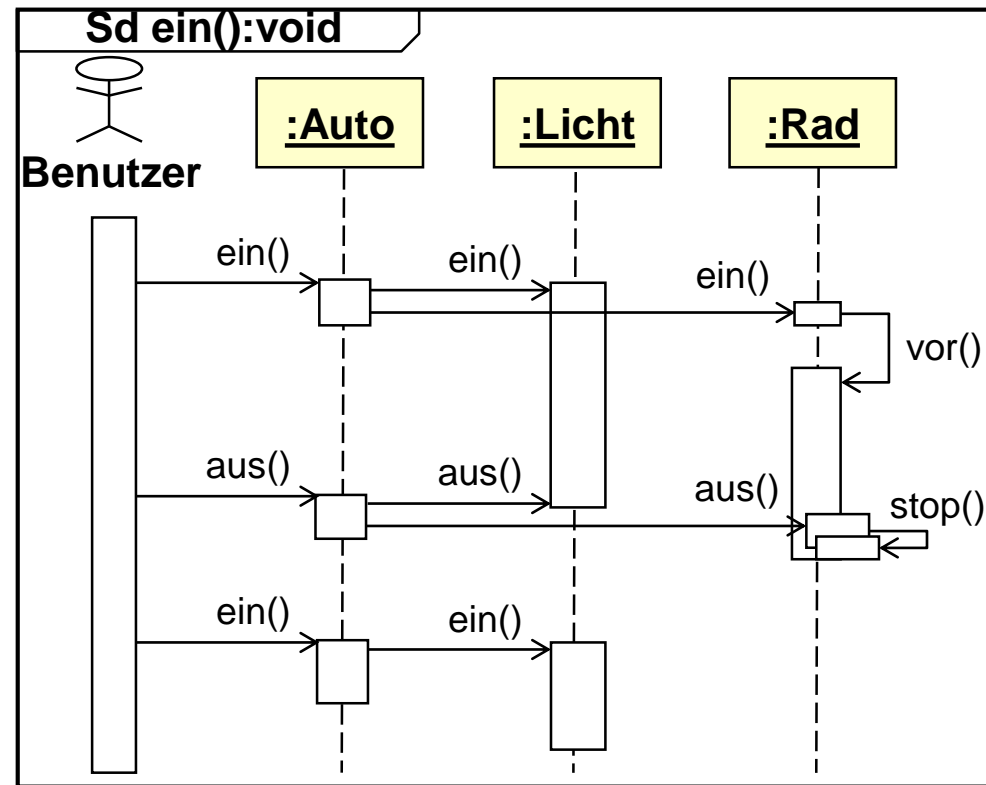
Sequenzdiagramm



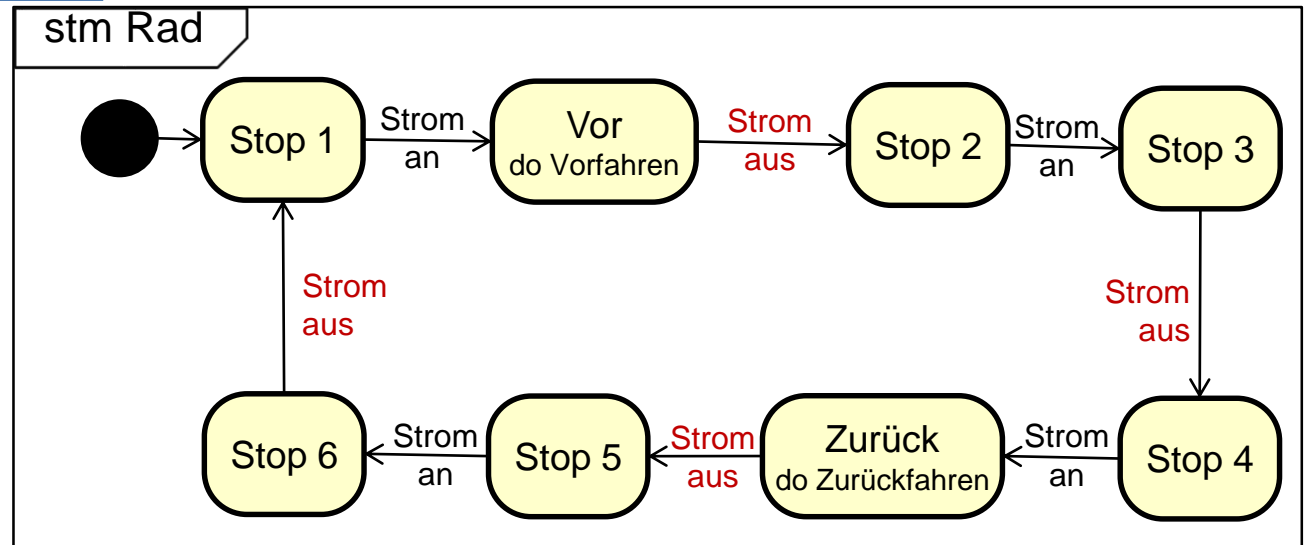
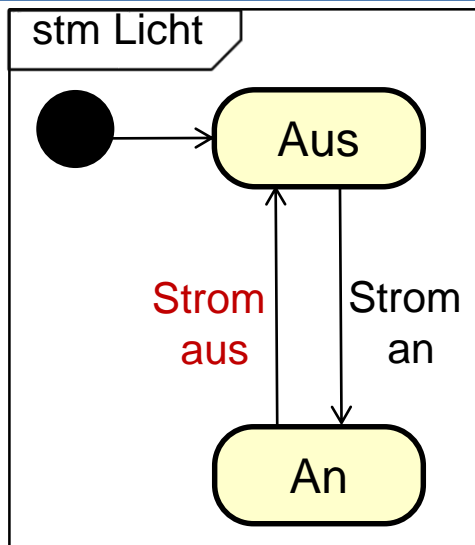
- Modelliert asynchrones Verhalten
- ... allerdings nur einen Ausschnitt

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 richtigen Notation schon der wichtigste Teil der Lösung geleistet.
 - ◆ Um so wichtiger, genau zu verstehen, was welcher Diagrammtyp ausdrücken kann.

4.1.6 Zusammenfassung

Was Sie sich zur Anforderungserhebung mindestens merken sollten:

- Konzepte
- Aktivitäten
- Produkte

Anforderungserhebung ▶ Wichtigste Konzepte

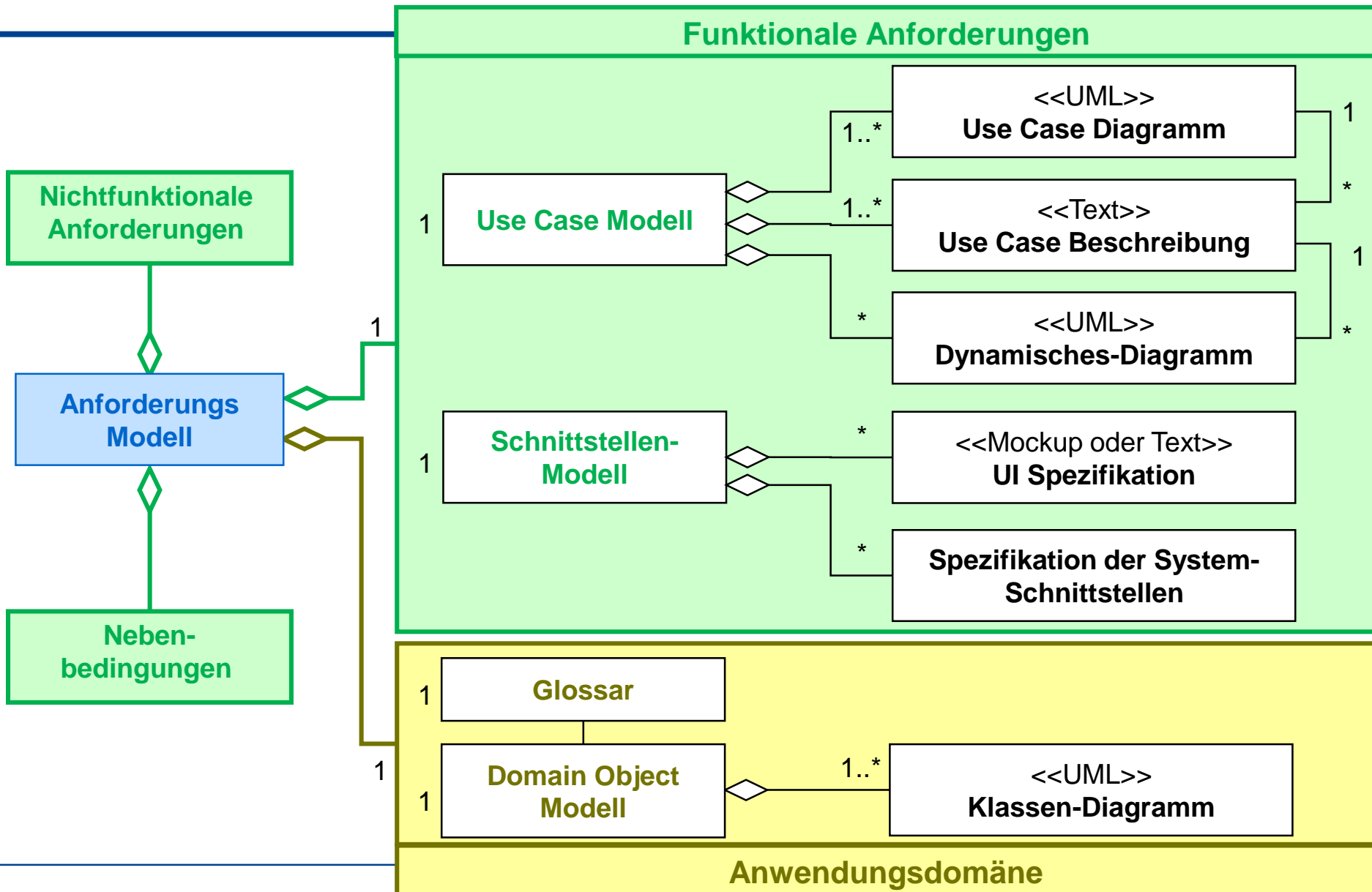
- Szenarien
 - ◆ Großartiger Weg zur Kommunikation mit dem Kunden
 - ◆ As-Is Szenarien, Visionäre Szenarien, Evaluationsszenarien, Training Szenarien
- Use Cases
 - ◆ Abstraktion von Szenarien
 - ◆ Use Case Modell erfasst vor allem funktionale Anforderungen
- Domain Object Modell
 - ◆ Klassendiagramme 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 Modell)
 - ◆ Spezifikation von elementarem Verhalten

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

Anforderungserhebung ▶ Produkte



- Use Case Modell

- ◆ Beschreibung jedes Anwendungsfalls durch strukturierten Text
- ◆ Beschreibung der Beziehungen der Anwendungsfälle durch ein oder mehrere Diagramme
- ◆ Evtl. Beschreibung der Abläufe komplexer Anwendungsfälle durch dynamische Diagramme
- ◆ Evtl. Beschreibung der Geschäftsprozesse in die die Anwendungsfälle eingebettet sind durch dynamische Diagramme.

- Schnittstellen-Modell

- ◆ Mockups (= mit Papier, Farben, etc. simulierte Benutzeroberflächen)
- ◆ System-Schnittstellen-Beschreibung (textuell oder Klassendiagramm)

- Glossar

- ◆ Wörterbuch der Anwendungsdomäne
- ◆ Begriffe sind durch freien Text definiert

- Domain Object Modell

- ◆ Erfassung der wichtigsten Konzepte des Glossars und ihrer Beziehungen durch ein Klassendiagramm / Klassendiagramme