

## Kapitel 6

# Anforderungsanalyse („Requirements Analysis“)

Stand: 20.11.2017

- 6.1 Das Analyse-Modell
- 6.2 Objektmodellierung im Analyseworkflow
- 6.3 Erstellung des Analyse-Objektmodells
- 6.4 Verfeinerung des dynamischen Modells
- 6.5 Modellierung der Dynamik von GUIs
- 6.6 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

## 6.1 Das Analyse-Modell

Abgrenzung Anforderungserhebung zu Anforderungsanalyse

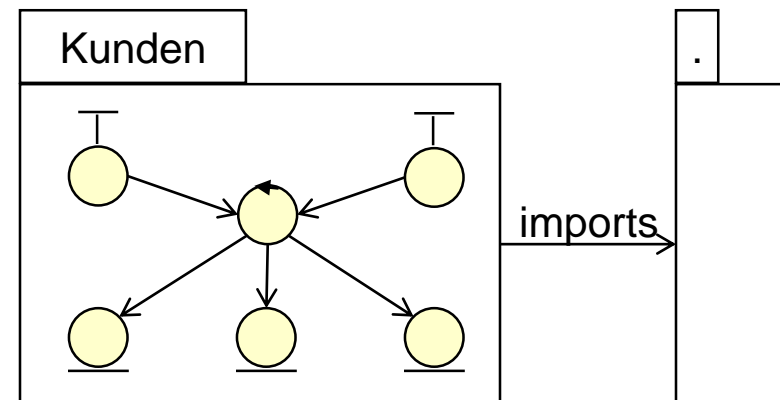
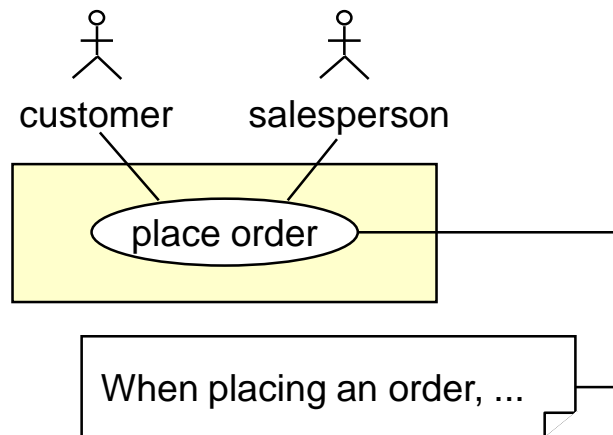
# Use Case vs. Analyse Modell (1)

## Use Case Modell

- in der Terminologie der Anwender verfasst
- externe Sicht des Systems (was tut es?)
- in „use cases“ strukturiert

## Analyse Modell

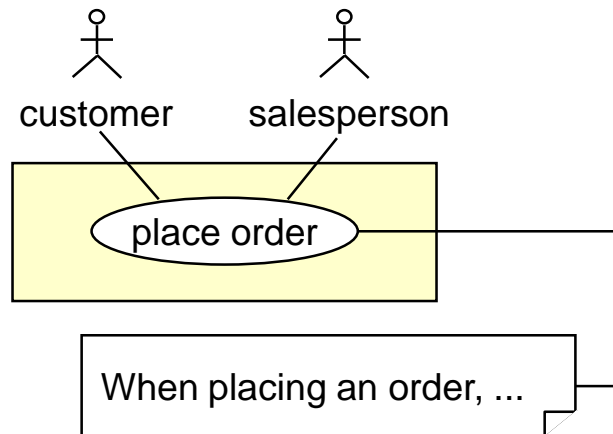
- in der Terminologie der Entwickler verfasst
- interne Sicht des Systems (wie tut es das?)
- in Module und Klassen-Stereotypen strukturiert



# Use Case vs. Analyse Modell (2)

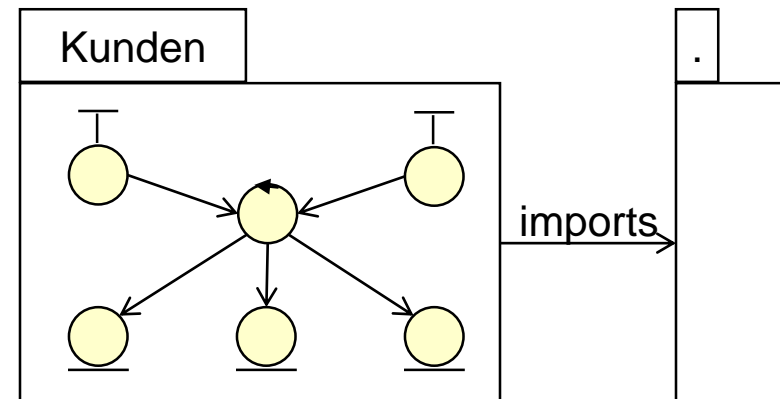
## Use Case Modell

- 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



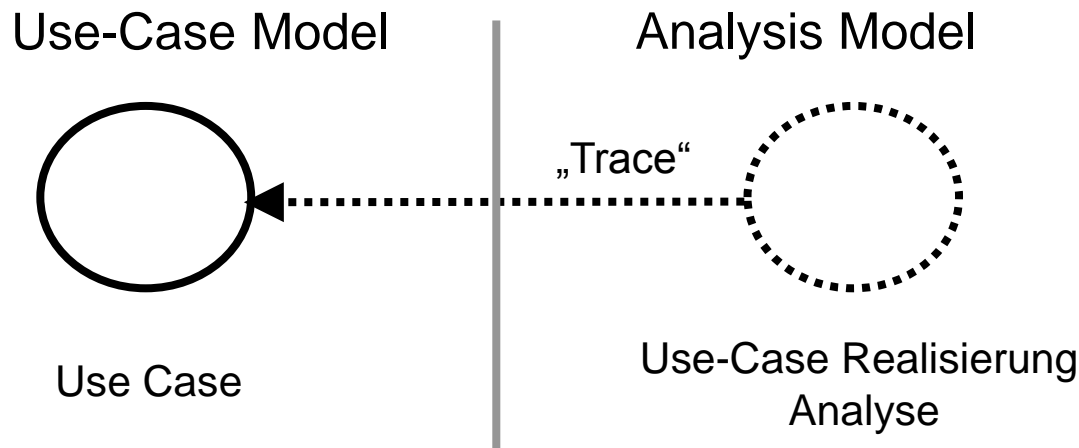
## Analyse Modell

- 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



## 6.2 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:

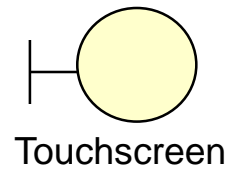
- **Boundary Typ** = Schnittstelle zu Akteur
- **Control Typ** = Ereignisfluss eines Use Cases
- **Entity Typ** = Anwendungskonzept („Business object“)



- 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

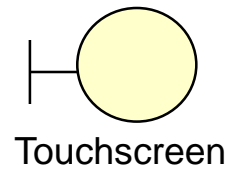
- Entspricht oft den persistenten Informationen der Anwendung
  - ◆ Das ergibt sich aus der Use Case übergreifenden Bedeutung
  - ◆ Entities sind aber keine reinen Daten-Klassen
  - ◆ Sie können komplexes Verhalten besitzen (z.B. Zinsberechnung)



- 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

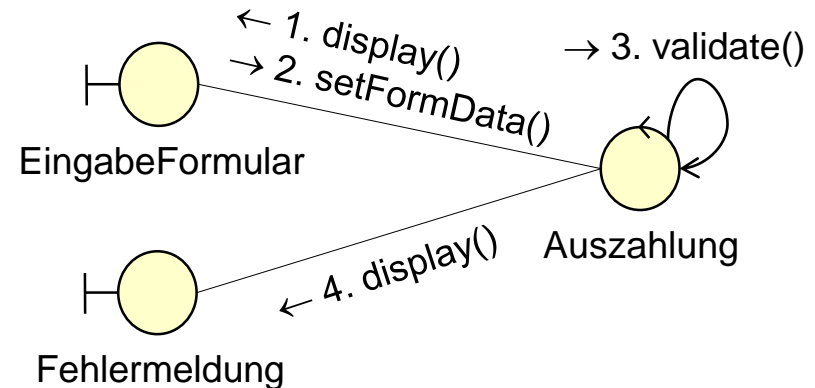
## Benennung

- 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



- 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

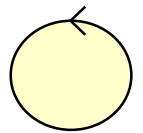


- So grob wie möglich ▶ Möglichst große logische Einheiten zusammenfassen.

- ◆ Nicht jeder Knopf ist ein Boundary!

- ◆ Eine zu feine Aufteilung würde

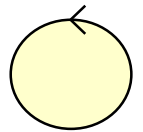
- ⇒ die Modellierung unnötig komplex machen (sich in Details verlieren)
- ⇒ dauernde Änderungen des Analysemodells für jede Änderung der graphischen Oberfläche nach sich ziehen



- 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`



- **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 Systems (boundaries) ändern sich häufig
    - ⇒ Darstellung auf zeilenorientiertem Terminal, im Webbrowser, ...
  - ◆ Die Geschäftsabläufe (controller) ändern sich häufig
    - ⇒ Ablauf der Kreditvergabe, Kreditwürdigkeitskriterien, ...
  - ◆ Jede dieser Änderungen soll den Rest des Systems nicht beeinflussen
    - ⇒ Insbesondere sollte die Anwendungsdomäne (entities) möglichst stabil bleiben
    - ⇒ Konten, Kredite, Zinsen, Tilgung, etc.
- **Flexibilität** ▶ Modelle die **leichter 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

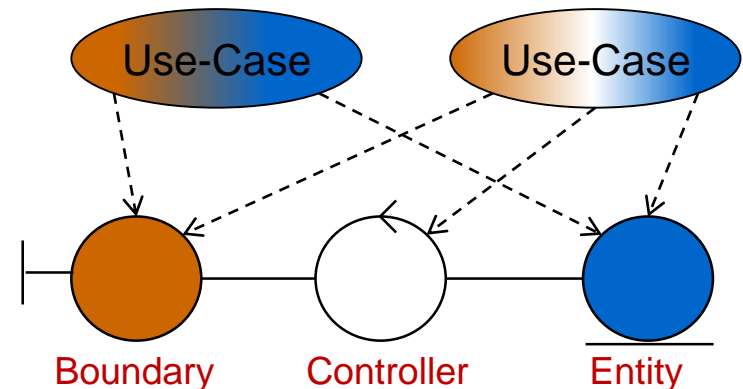
- „Entity, Boundary, Controller“ = „Model, View, Controller“ (MVC)

# 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

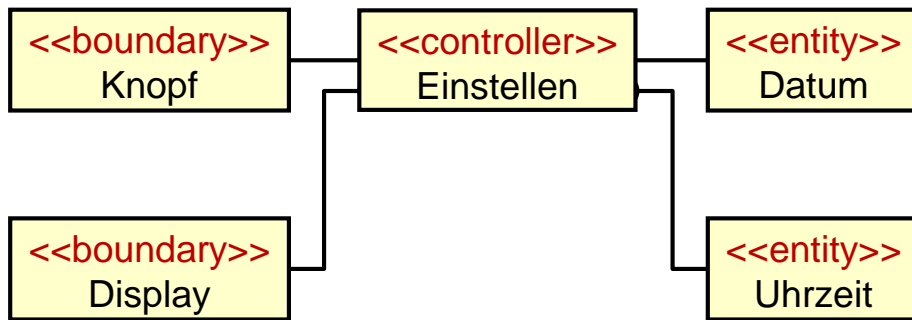


## Layout von Analyse-Klassendiagrammen

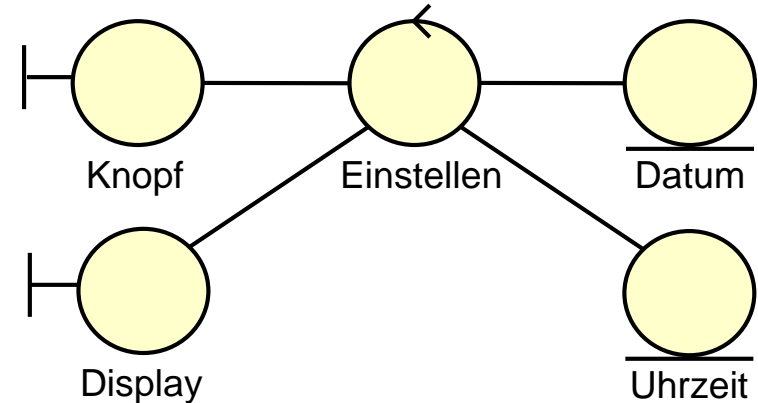
- Boundaries links, Controller in der Mitte, Entities rechts

# Analyse-Objektmodell „Digitaluhr“

## Textuelle Notation\*



## Graphische Notation\*



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

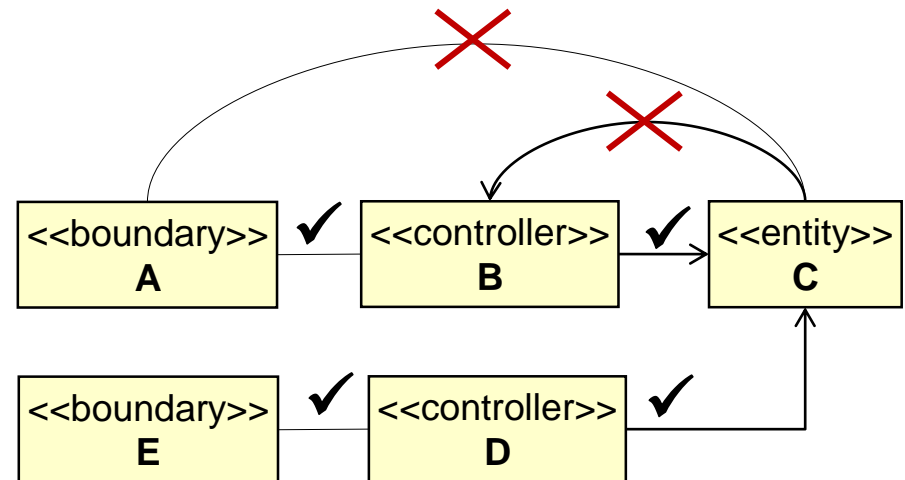
- Man kann die Stereotypen für Analyse-Klassen graphisch oder textuell notieren
- Klassendiagramme des Analysemodells die die drei Stereotypen nutzen heißen auch „**Robustness Diagrams**“ → Warum wohl?
- Vergleichen Sie obiges Diagramm mit dem Klassendiagramm der Digitaluhr im UML-Foliensatz und diskutieren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

## **6.3 Workflow „Erstellung des Analyse-Objektmodells“**

Von Use Cases zu Boundaries, Controllern und Entities in einer kleinen Bankanwendung

# Analyse-Objektmodell-Erstellung

---

## Ausgangspunkt

(0) Vorhandenes Use-Case-Modell und Domain Object Model

## Verfahren (pro Use Case)

(1) Identifikation des Controllers und der Boundaries, (und Entities)

- ◆ Erzeugung entsprechender Klassen
- ◆ Erzeugung ihrer Assoziationen

(2) Eventuell Ergänzung von Entities

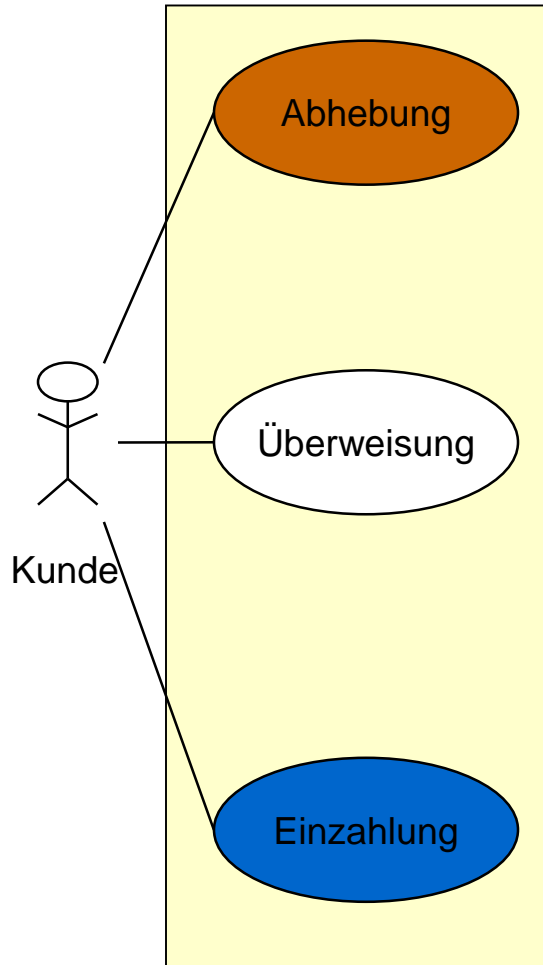
- ◆ Möglicherweise vorher übersehene Domänen-Konzepte

(3) Konsolidierung des erweiterten statischen Modells

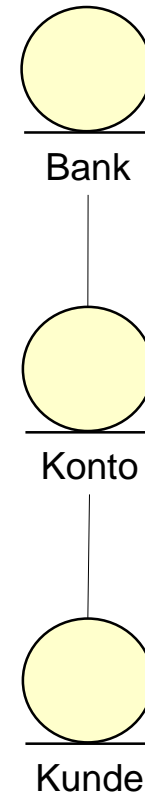
- ◆ Zusammenfassung von Klassen mit ähnlichen oder identischen Funktionen

# Beispiel: Ausgangspunkt

## Use-Case Modell

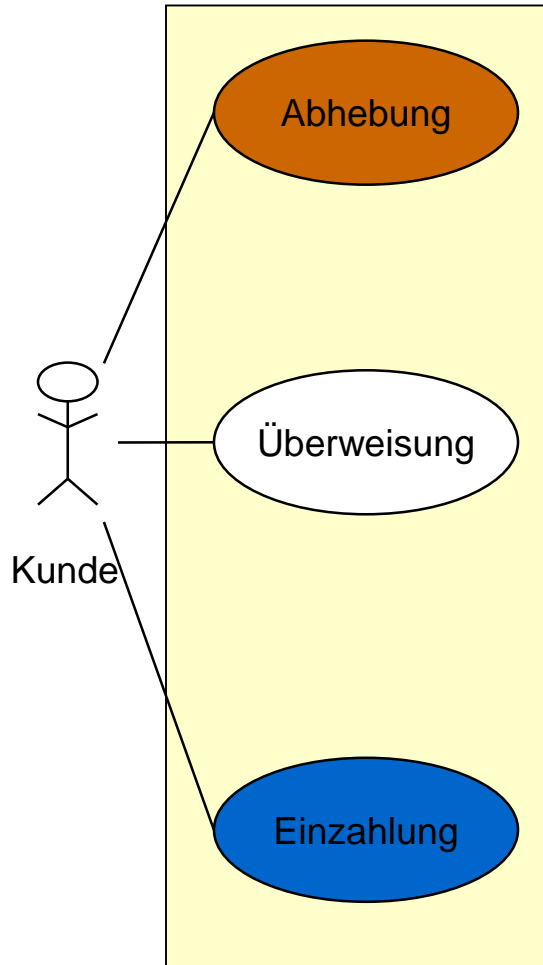


## Domain-Objektmodell

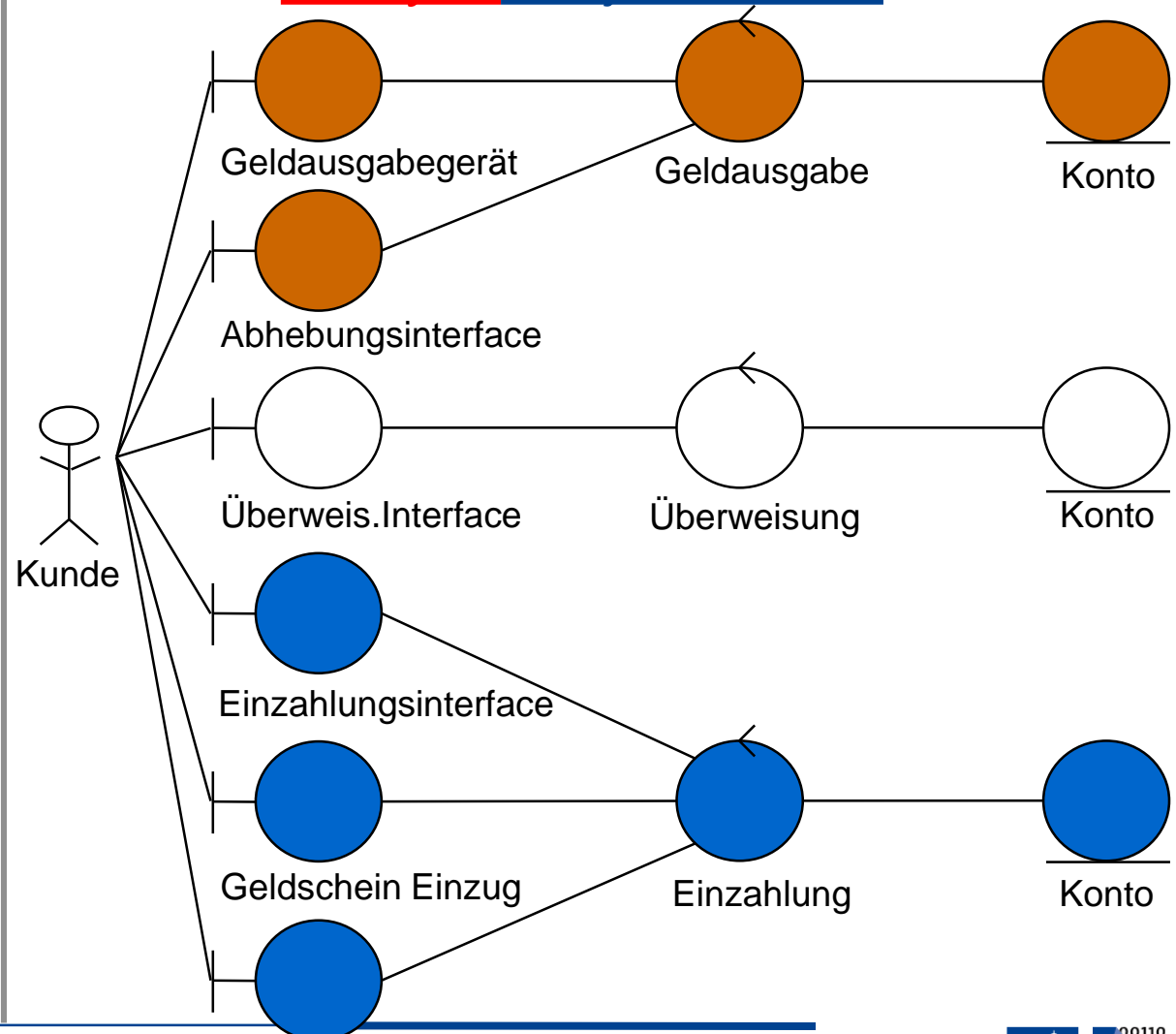


# Use-Case Modell ▶ Analyse-Objektmodell

## Use-Case Modell

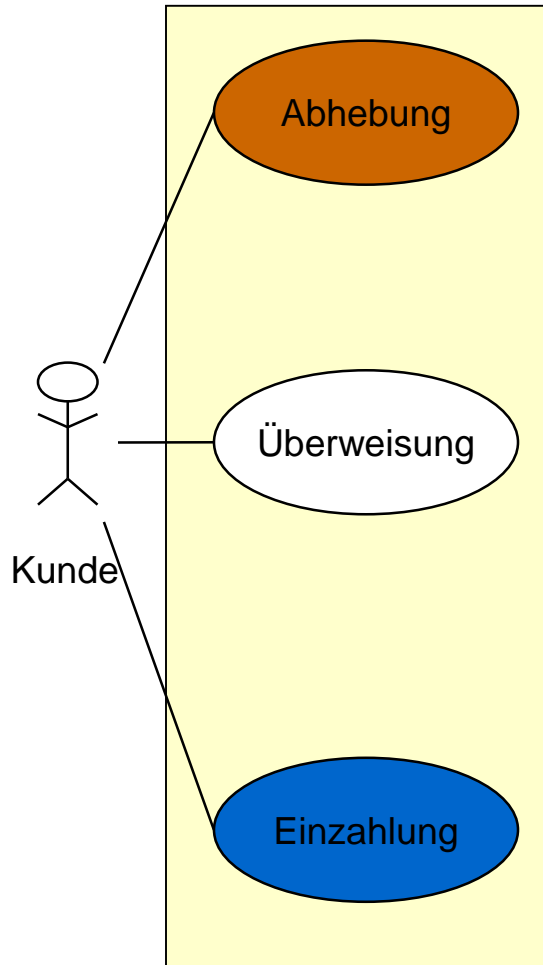


## Analyse-Objektmodell

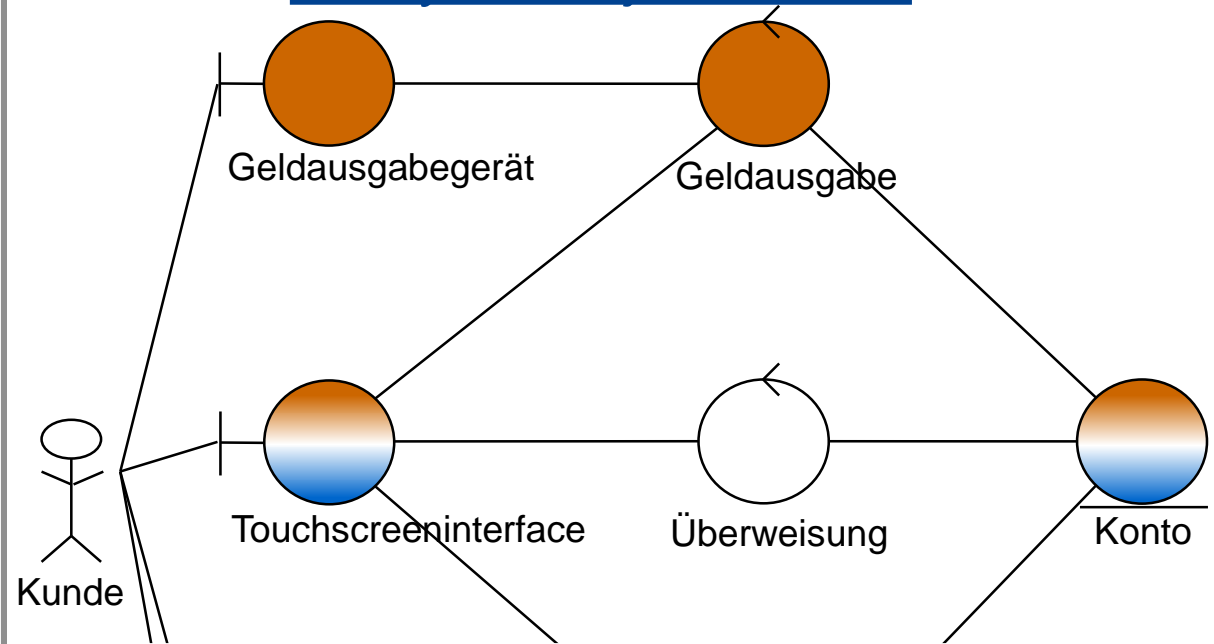


# Use-Case Modell ▶ Analyse-Objektmodell

## 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

## 6.4 Workflow: „Verfeinerung des dynamischen Modells“

Spezifikation aller Interaktionen der Analysetypen

# Verhaltensmodellierung eines Use Case

---

## Ausgangspunkt

- (0) Vorhandenes Analyse-Objektmodell

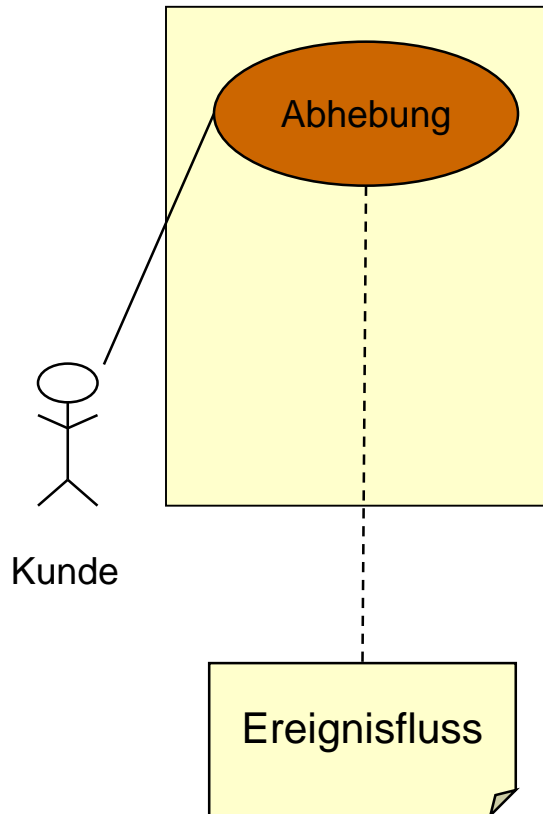
## Verfahren (pro Use Case)

- (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)

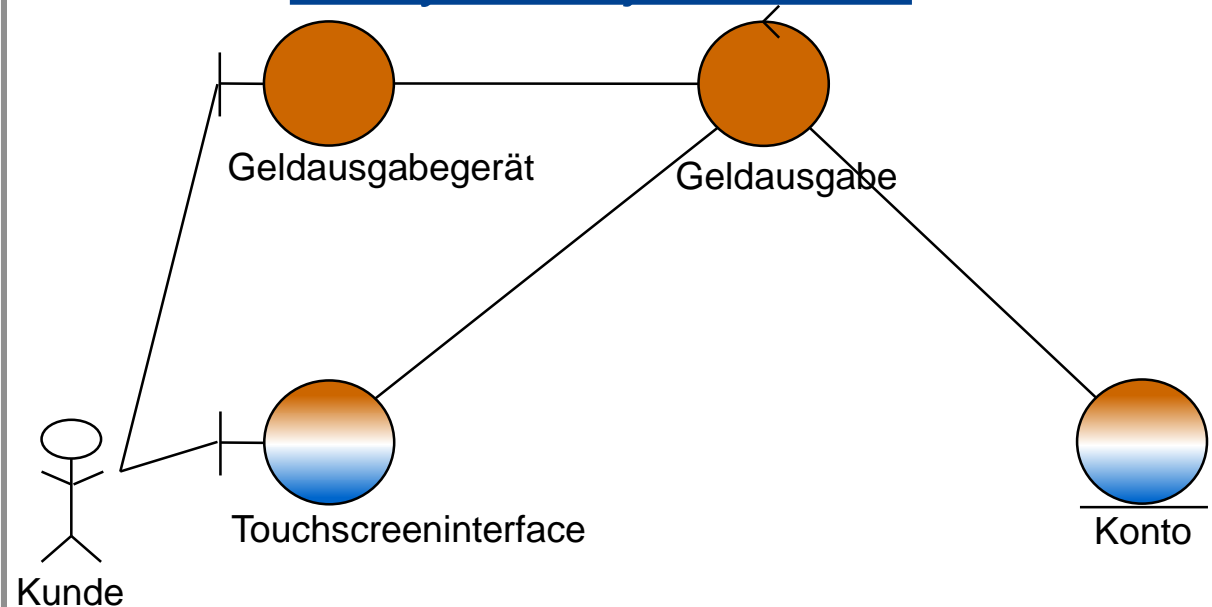


# Use-Case Modell ▶ Verhaltensmodell der Analyse

## Use-Case Modell



## Analyse-Objektmodell



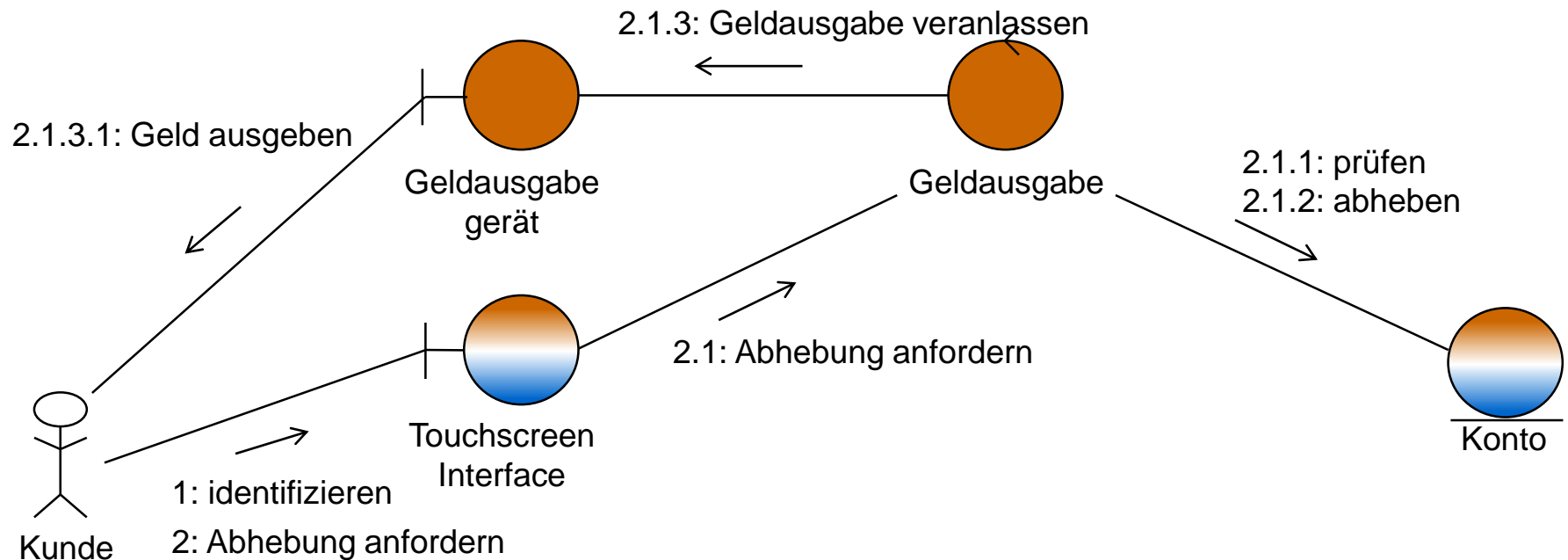
Nächster Schritt: Ereignisfluss des Use Case „Abhebung“ auf Interaktionen der betroffenen Analyse-Objekte abbilden!

→ **Kommunikationsdiagramm**



# Objektmodell ▶ Verhaltensmodell

## Ereignisfluss "Abhebung" auf Kommunikationsdiagramm abbilden

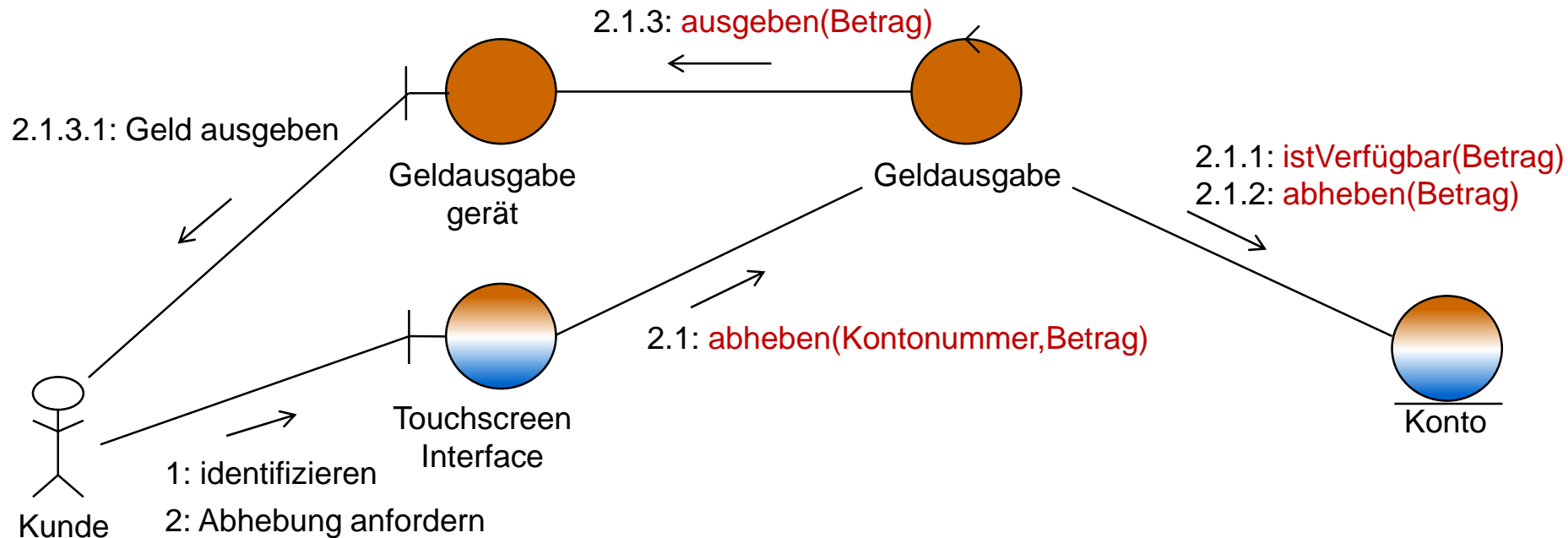


- Fortsetzung (1)

- ◆ informelle Aktionen durch konkrete Nachrichten ersetzen

# Objektmodell ▶ Verhaltensmodell

## Ereignisfluss "Abhebung" auf Kommunikationsdiagramm abbilden

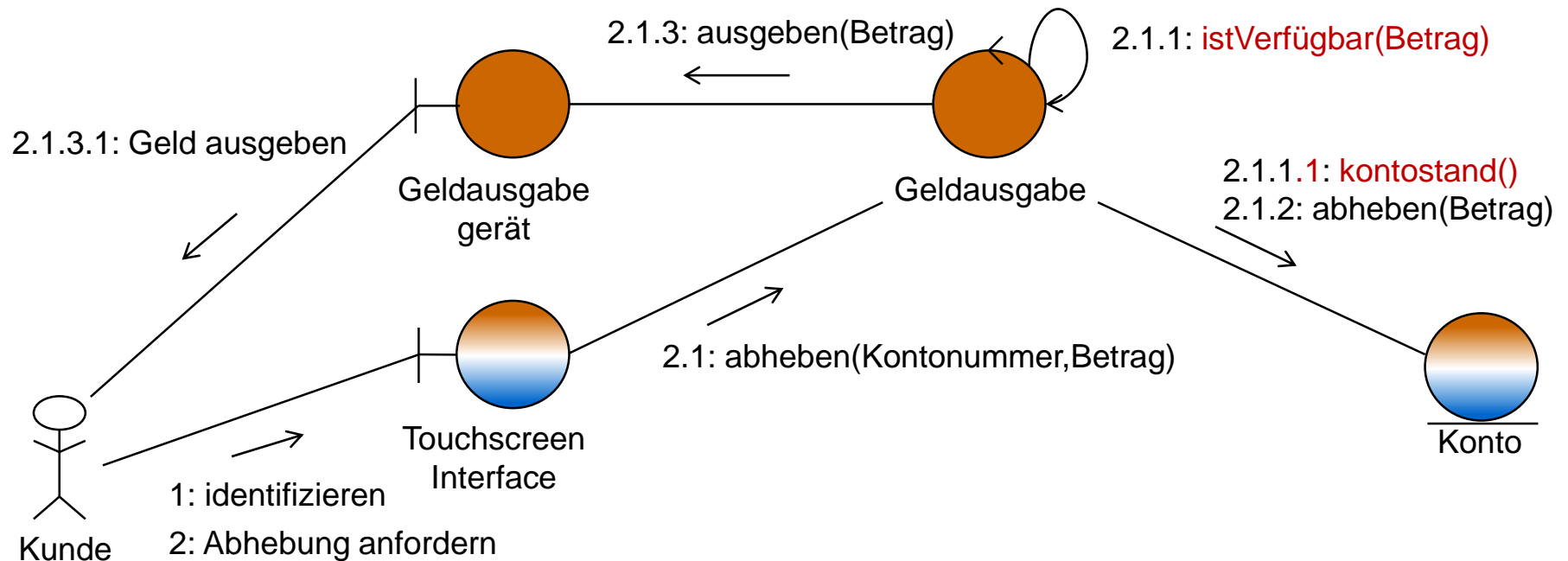


- Fortsetzung (1)

- ◆ informelle Aktionen durch konkrete Nachrichten ersetzen

# Objektmodell ▶ Verhaltensmodell

## Ereignisfluss "Abhebung" auf Kommunikationsdiagramm abbilden

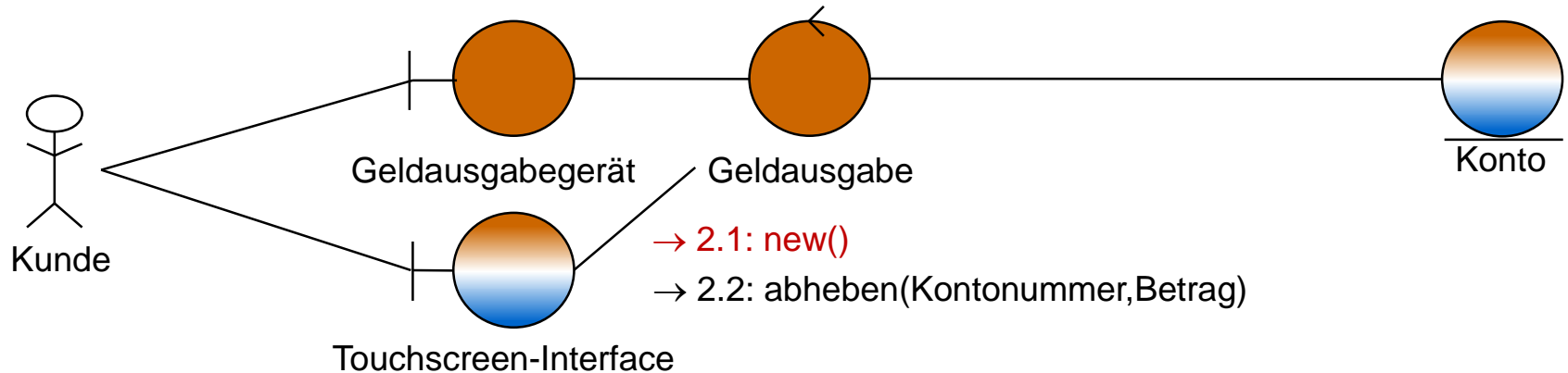


### ● Fortsetzung (2)

- ◆ 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

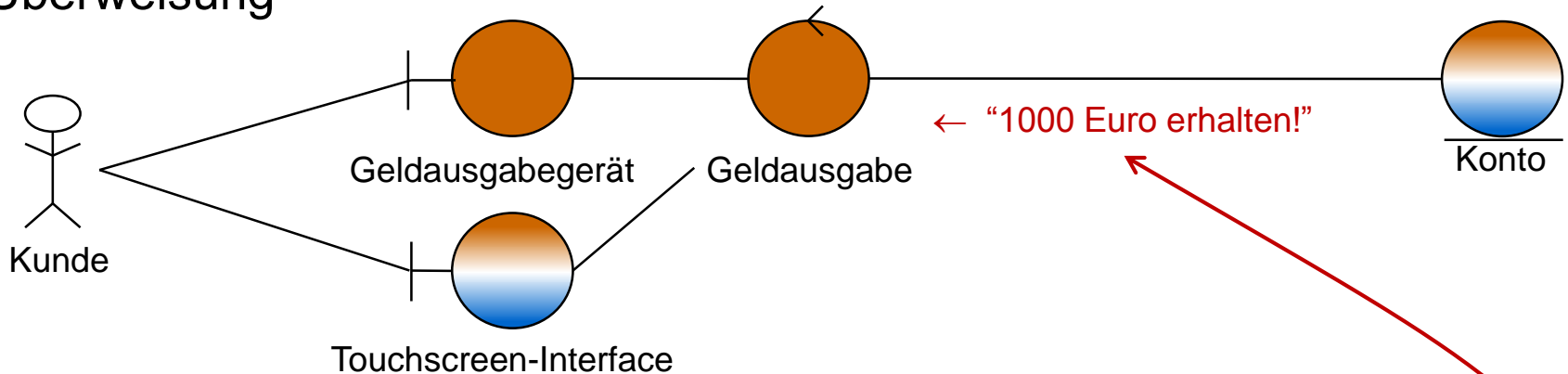


- **Controller-Objekte** werden bei der Initiierung des Use Case erschaffen
  - ◆ **A)** In einem Boundary das der Initiierung des Use Case dient
  - ◆ **B)** In Controller eines Use Case der die Quelle einer <<include>> oder das Ziel einer <<extend>>-Beziehung zum eigenen Use Case ist
- **Boundary-Objekte** 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 **Typebene** (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-Verfeinerung

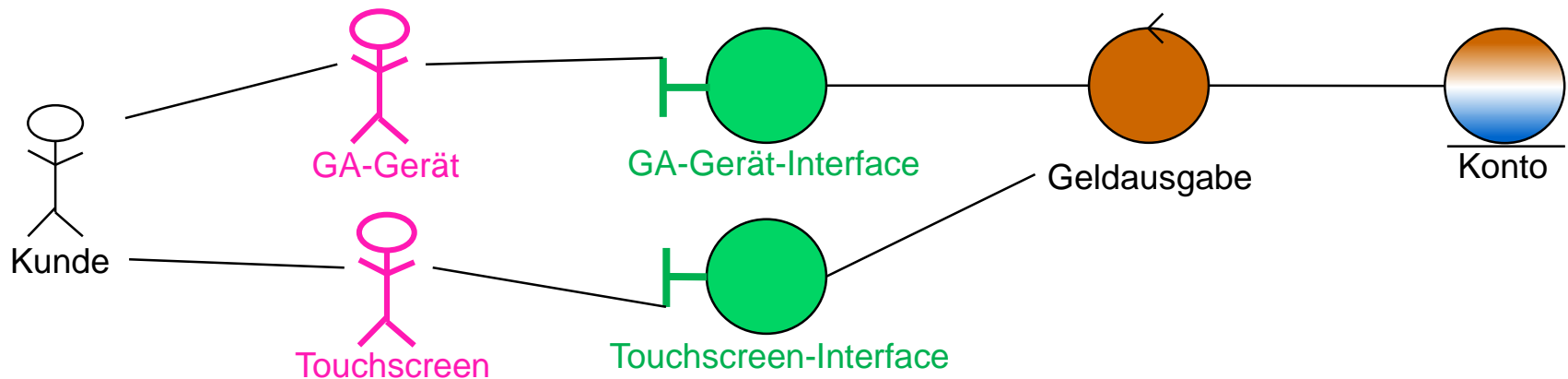
## ▶ Akteure versus Boundaries (1)

**Akteure:** Alles Äußere, womit das System das wir modellieren direkt oder indirekt interagiert.

- ▶ Beispiele: Die physischen Geldausgabe-Geräte und Touchscreens (die wir als Softwareentwickler in einer Bank nicht selber bauen!)

**Boundaries:** Die Teile unseres Systems, die direkt mit der Außenwelt (= den Akteuren) kommunizieren.

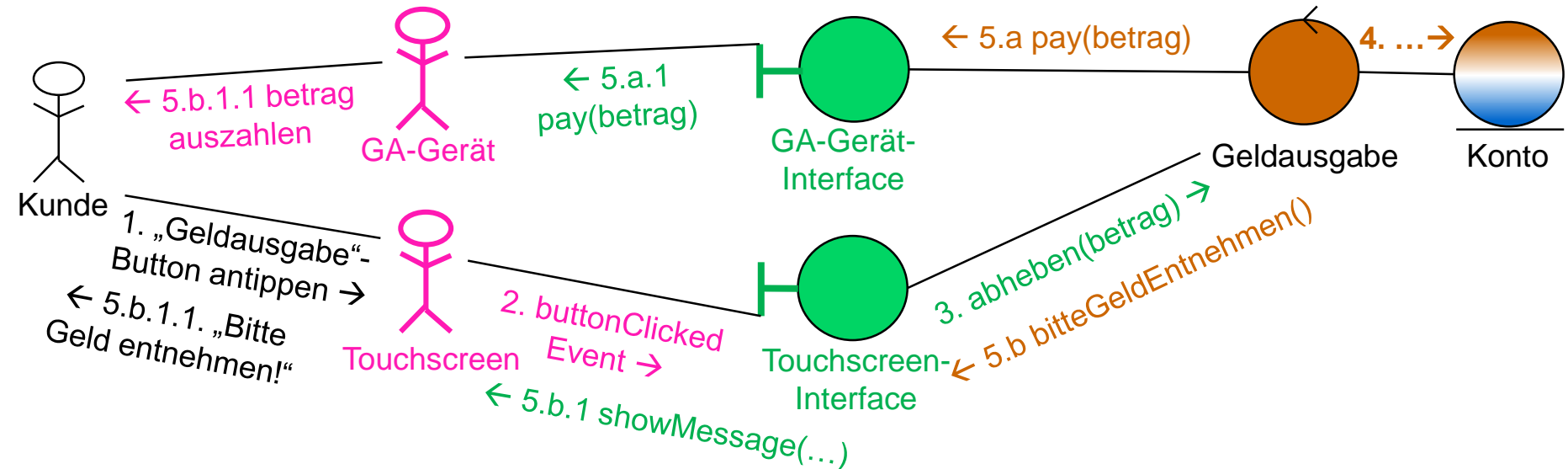
- ▶ Beispiele: Unsere Software-Schnittstelle zu Geldausgabe-Geräten und die GUI die wir auf den Touchscreens anzeigen lassen (beides müssen wir als Entwickler der Bankanwendung selber bauen!)



# Analyse-Verhaltensmodell

## ► Akteure versus Boundaries (2)

Die Trennung der beiden Konzepte macht Ihr Modell klarer. Sie ermöglicht zu unterscheiden zwischen Aktionen des Kunden, anderer Akteure und des Systems:

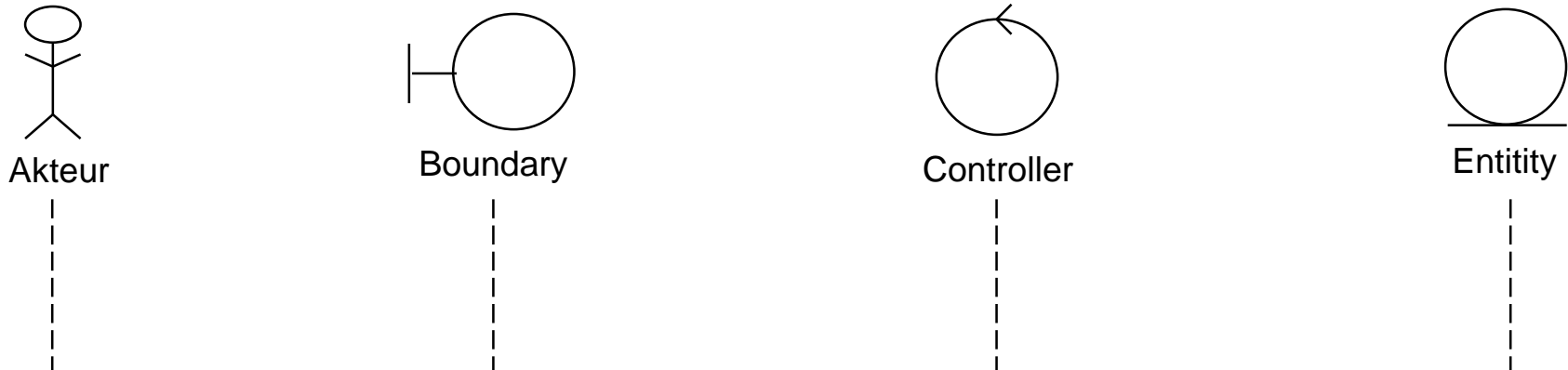


Diese Unterscheidung ermöglicht es Abläufe präziser / feiner granular zu spezifizieren (z.B. **buttonClickedEvent**, **pay(betrag)**).



# 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
- ◆ **Danach** ▶ 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?**

## 6.5 Dynamische Modellierung von Benutzerinterfaces

Konzeptionelle Sicht der Abläufe auf dem Benutzerinterface als Navigationsgraphen bzw. Zustandsautomaten



Termine

21.11.2014

23.11.2014

2 Gäste

Art der Unterkunft



Ganze  
Unterkunft



Privatzimmer



Gemeinschaftszimmer

Preisspanne



8€



1000+€

Weitere Filter

1000+ Unterkünfte · Paris

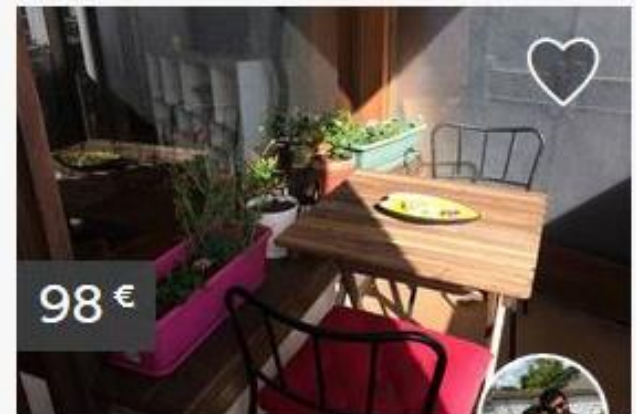


80 €



A 2 PAS DES CHAMPS...

Ganze Unterkunft · 34 Bewertungen · ...



98 €



A deux pas de la Tour E...

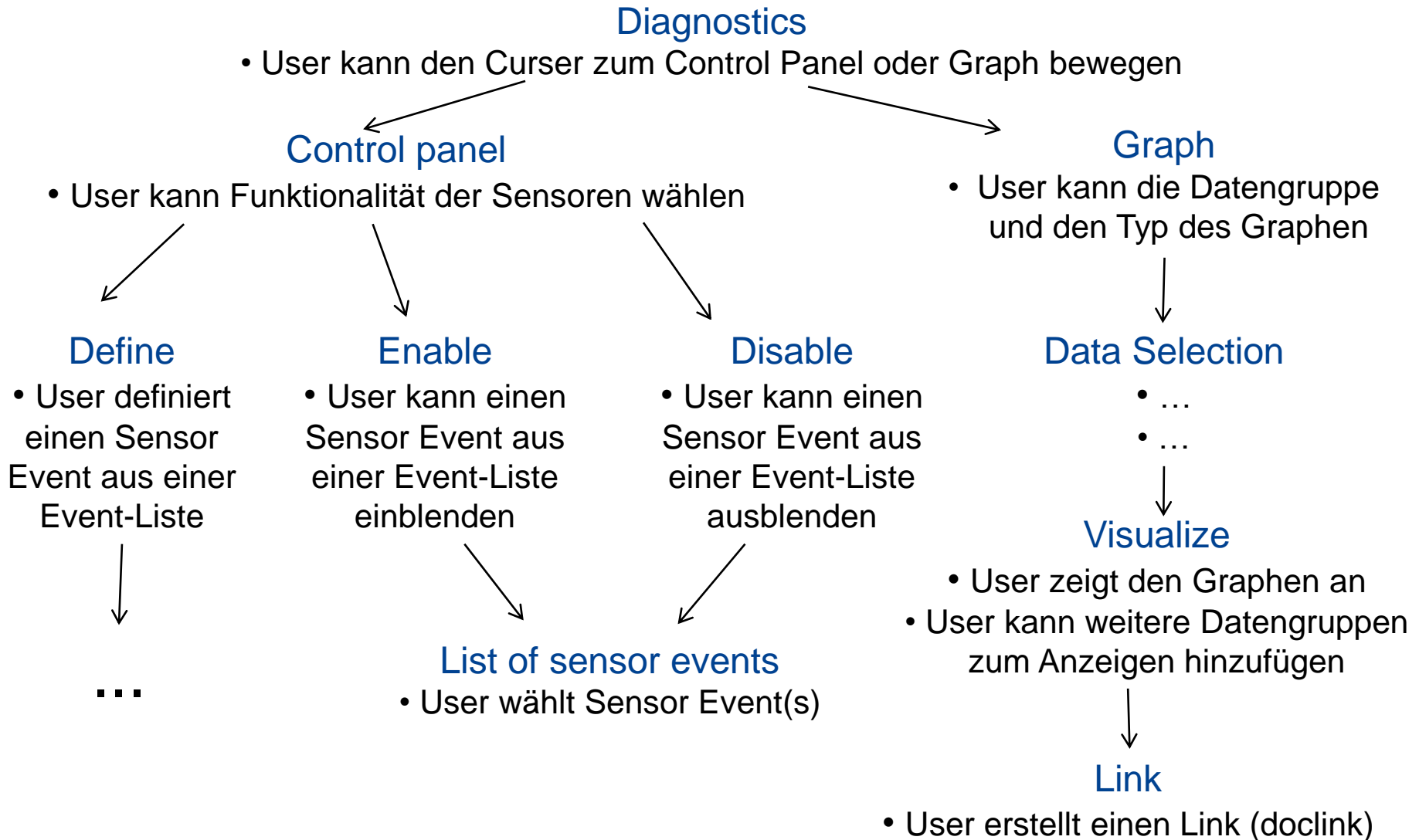
Ganze Unterkunft · 23 Bewertungen · T...

# Dynamische Modellierung von Benutzerschnittstellen

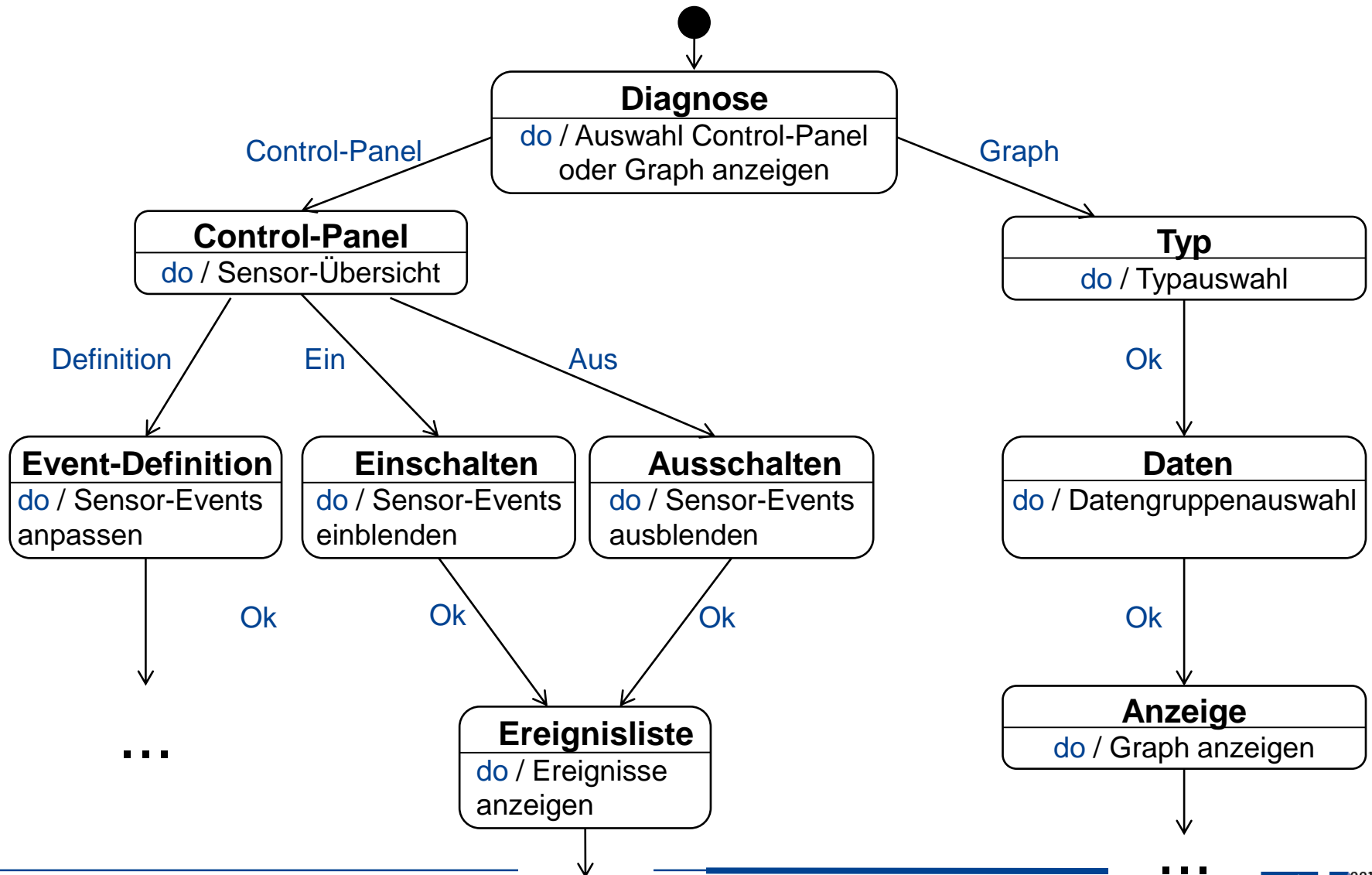
---

- 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





Termine

21.11.2014

23.11.2014

2 Gäste

Art der Unterkunft



Ganze  
Unterkunft



Privatzimmer



Gemeinschaftszimmer

Preisspanne



8€

1000+€

Übung: Malen Sie einige  
Navigationspfade für  
dieses Beispiel (Airbnb)

Weitere Filter

1000+ Unterkünfte · Paris



A 2 PAS DES CHAMPS...

Ganze Unterkunft · 34 Bewertungen · ...



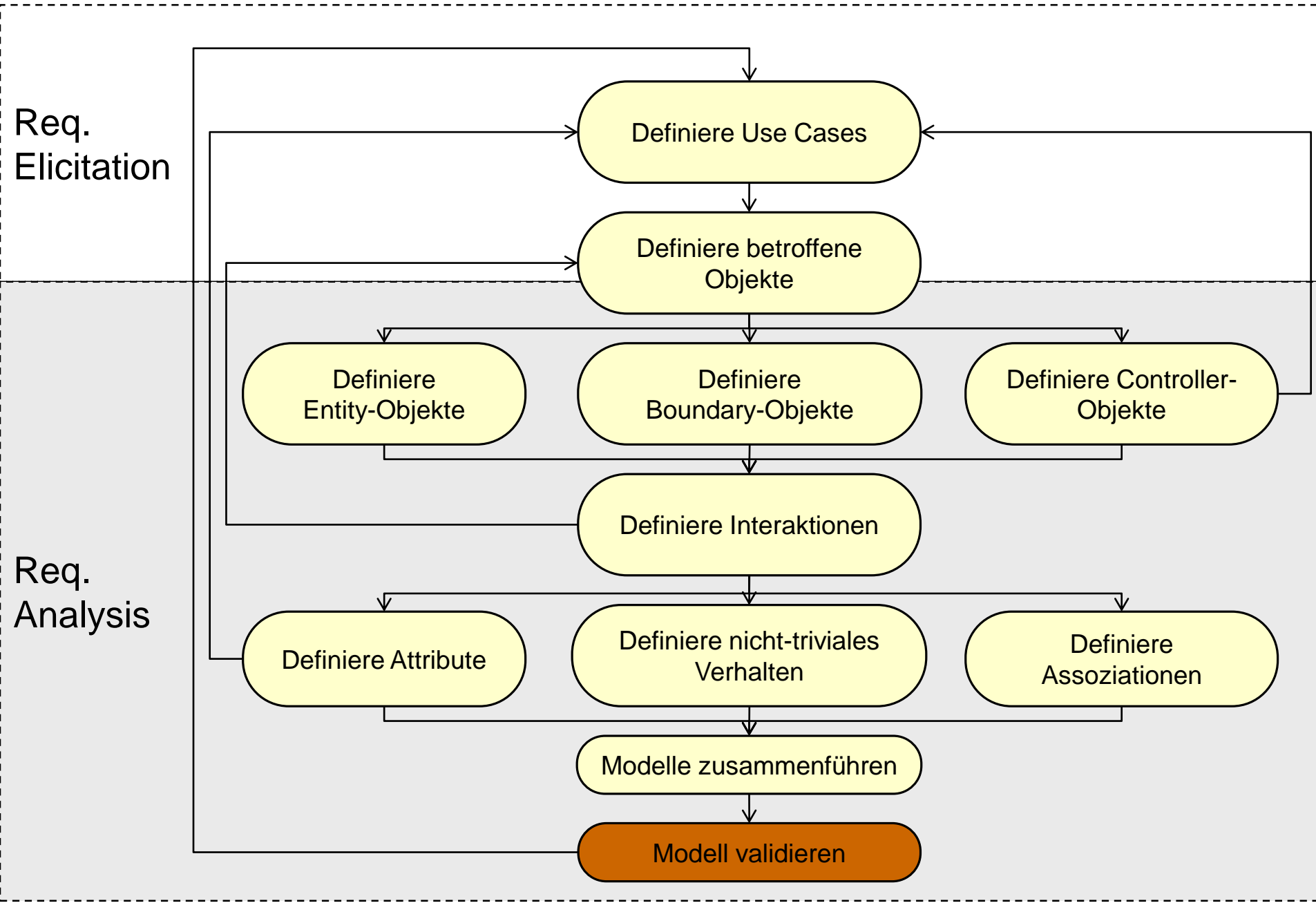
A deux pas de la Tour E...

Ganze Unterkunft · 23 Bewertungen · T...

## **6.6 Konsolidierung der Analyse**



# 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
  - ◆ Requisit Pro (Rational / IBM)
    - ⇒ <http://www.ibm.com/developerworks/rational/products/requisitepro/>
  - ◆ Jira
    - ⇒ <http://www.jira.com>

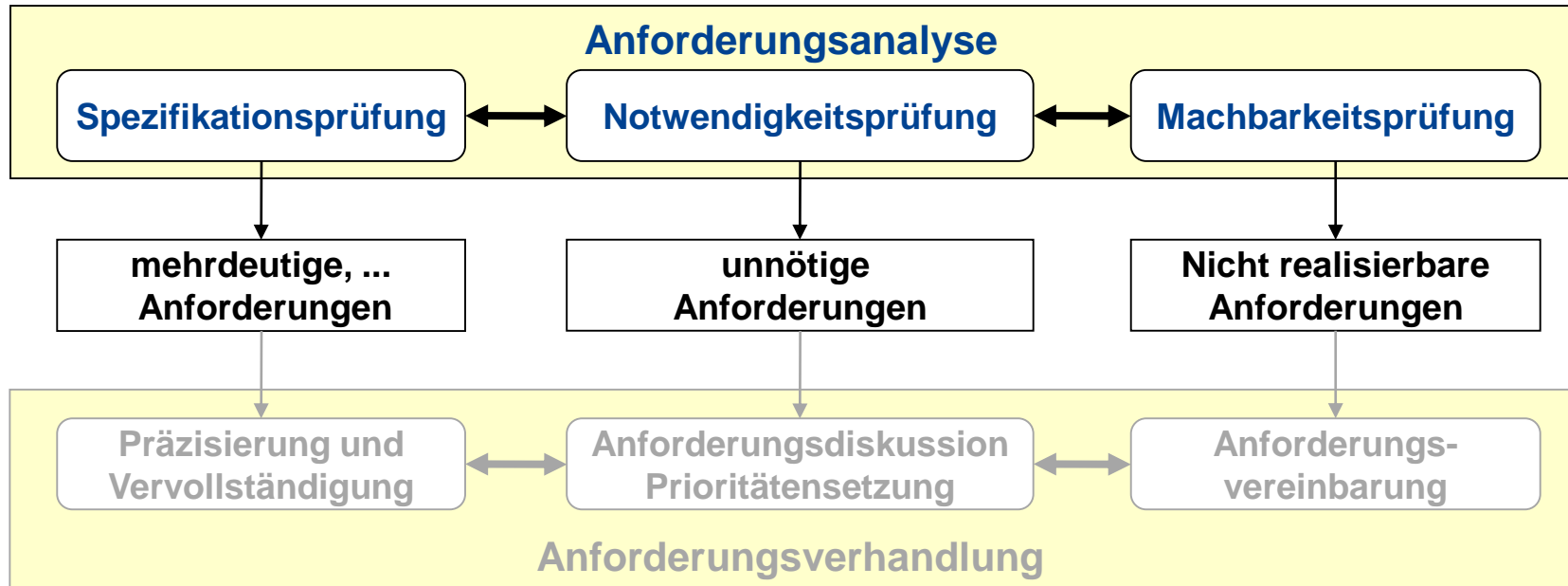
# Formatvorlage Anforderungsanalyse-Dokument

---

- 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“**

# 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?

# 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

---

In diesem Unterkapitel haben wir uns mit folgenden Themen befasst:

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