Design-Vorgehen (OOD)

- Unified Modeling Language (UML)
- Zwei Sichten auf ein Modell
- Entwicklung von
  - Sequenzdiagramm(en) und
  - Klassendiagramm

  gleichzeitig
- Ergebnisse des Vorgehens

Klarstellung

UML ≠ Klassendiagramm

UML ⊃ Klassendiagramm

Die Unified Modeling Language (UML)

- Die UML ist eine *graphische Notation* für die objektorientierte Entwicklung (Analyse, Entwurf, Implementierung)
- Graphische Notationen sind geeignet
  - um sich einen (abstrakten) Überblick über den Aufbau eines Anwendungsgebiets oder eines Programms zu verschaffen
  - um mit Kunden, Anwendern usw. zu kommunizieren
  - zum "drüber reden" unter Informatikern, für die Diskussion alternativer Implementierungsmöglichkeiten
  - etc.
- Diese Aktivitäten sind z.B. auf (Java-) Codeebene und mit Pseudocode nicht möglich!

Die Unified Modeling Language (UML)

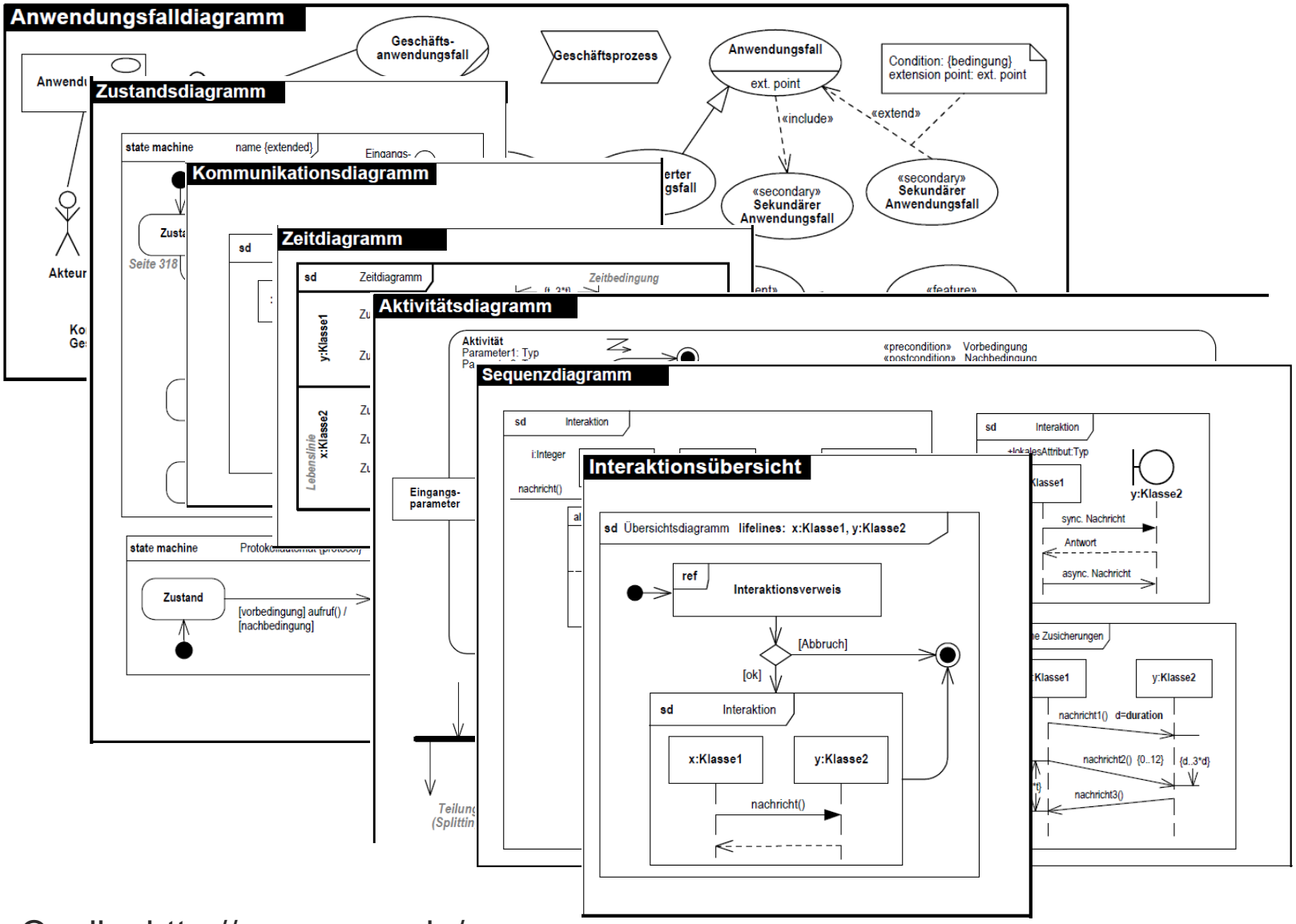
Die UML bietet viele Diagrammarten

- für die objektorientierte Analyse; darstellen lassen sich
  - Abläufe (hier besonders wichtig)
  - Strukturen

  des zukünftigen Systems
- für das objektorientierte Design; darstellen lassen sich
  - Abläufe
  - Strukturen

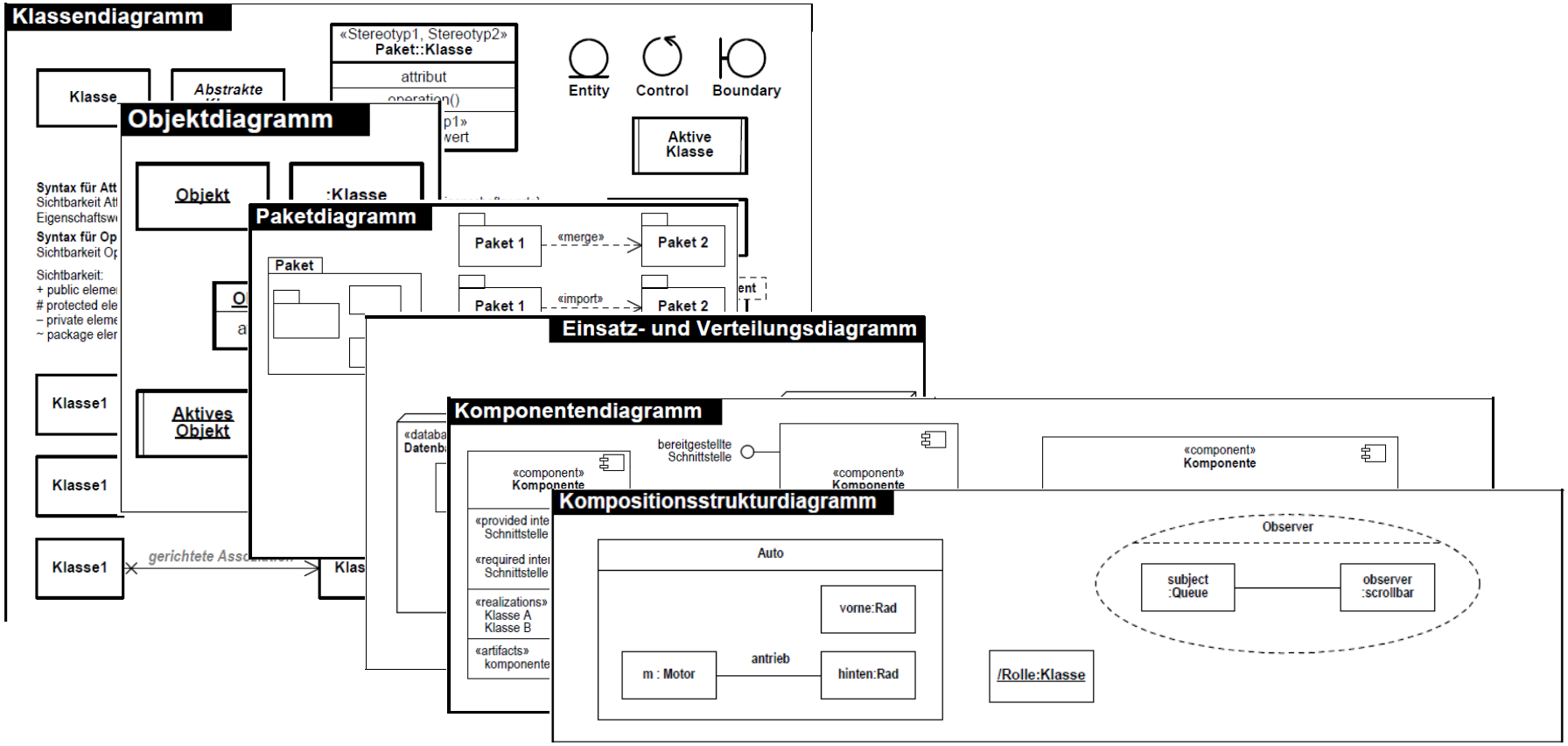
  des zukünftigen Systems

UML-Diagrammtypen: Abläufe



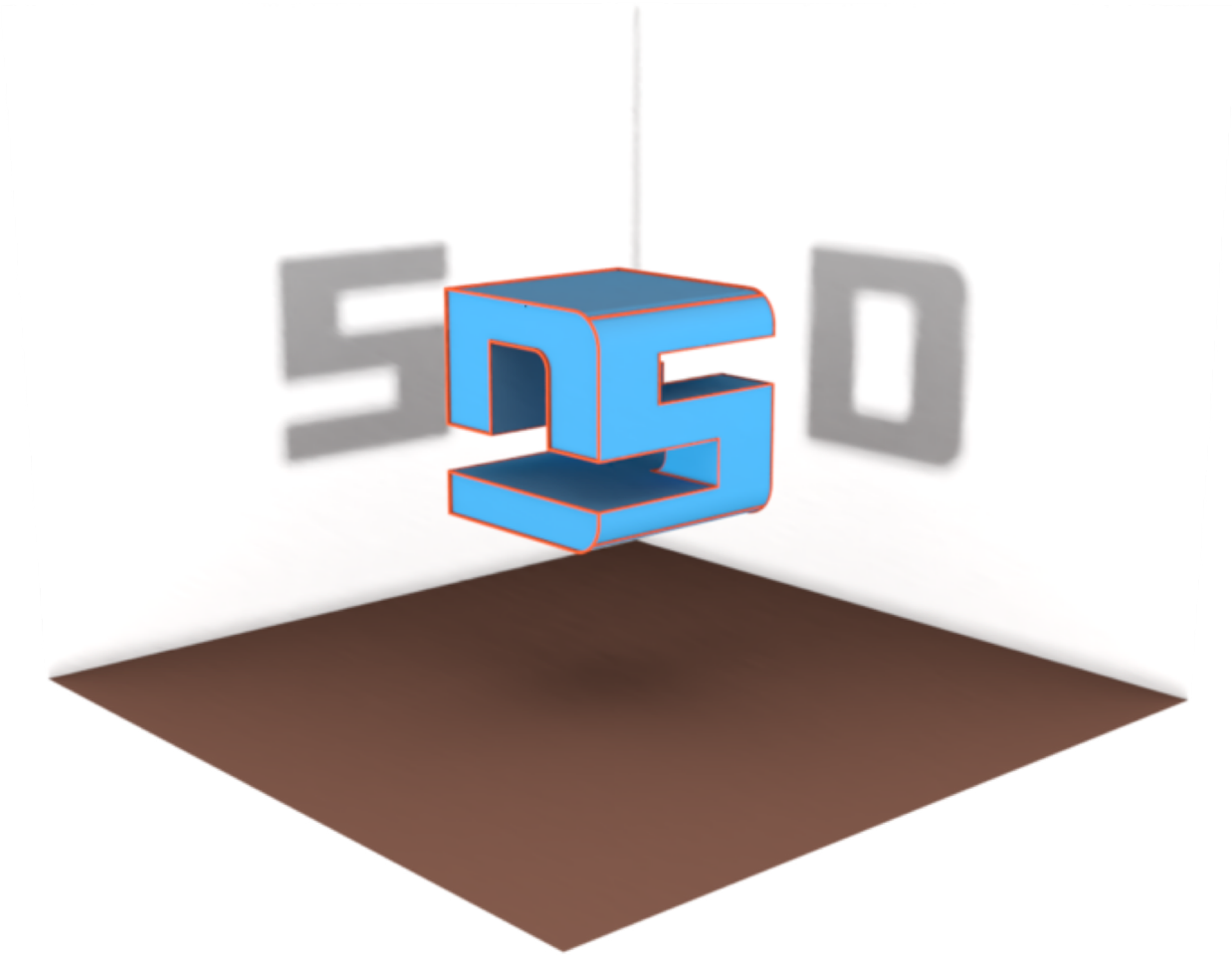
Quelle: http://www.oose.de/

UML-Diagrammtypen: Strukturen



Quelle: http://www.oose.de/

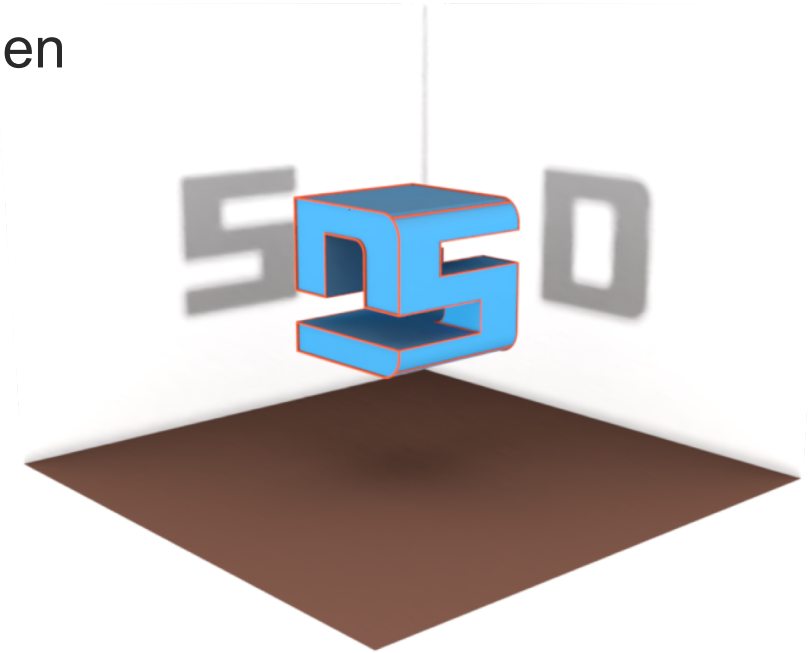
Statische und dynamische Sicht: die selbe Architektur aus mehreren Perspektiven

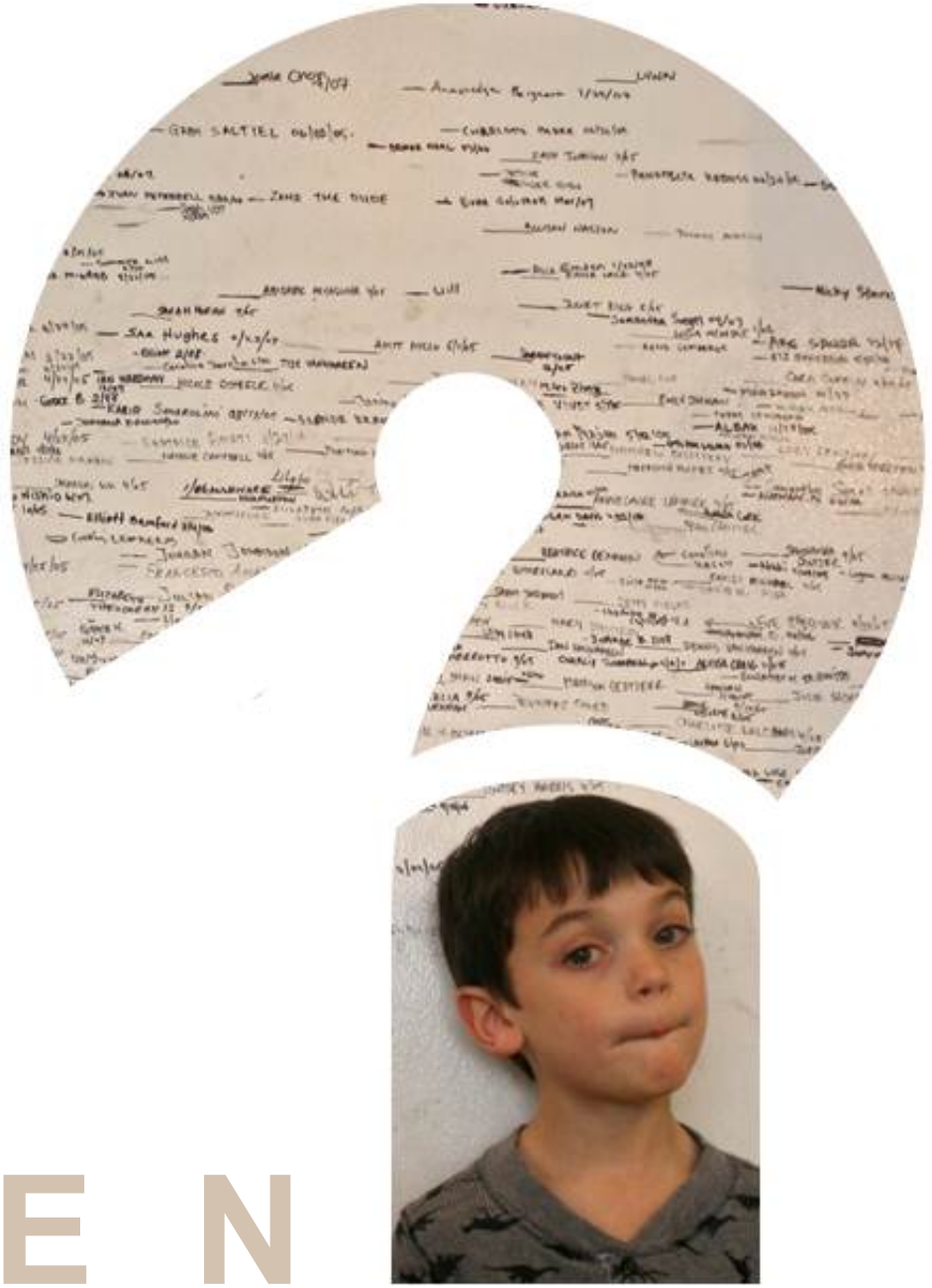


Vorgehen beim Design (OOD)

1. Wir überlegen uns die wichtigsten / offensichtlichen Klassen, die wir benötigen
   -> als (ein) Klassendiagramm zeichnen

2. Wir überlegen uns, wie die wichtigsten / schwierigsten Aufgaben zwischen diesen Klassen abgewickelt werden; der Fokus liegt auf der *Interaktion*, nicht den Algorithmen
   → als (mehrere) Sequenzdiagramme zeichnen
   → die dafür benötigten Methoden in das Klassendiagramm übernehmen

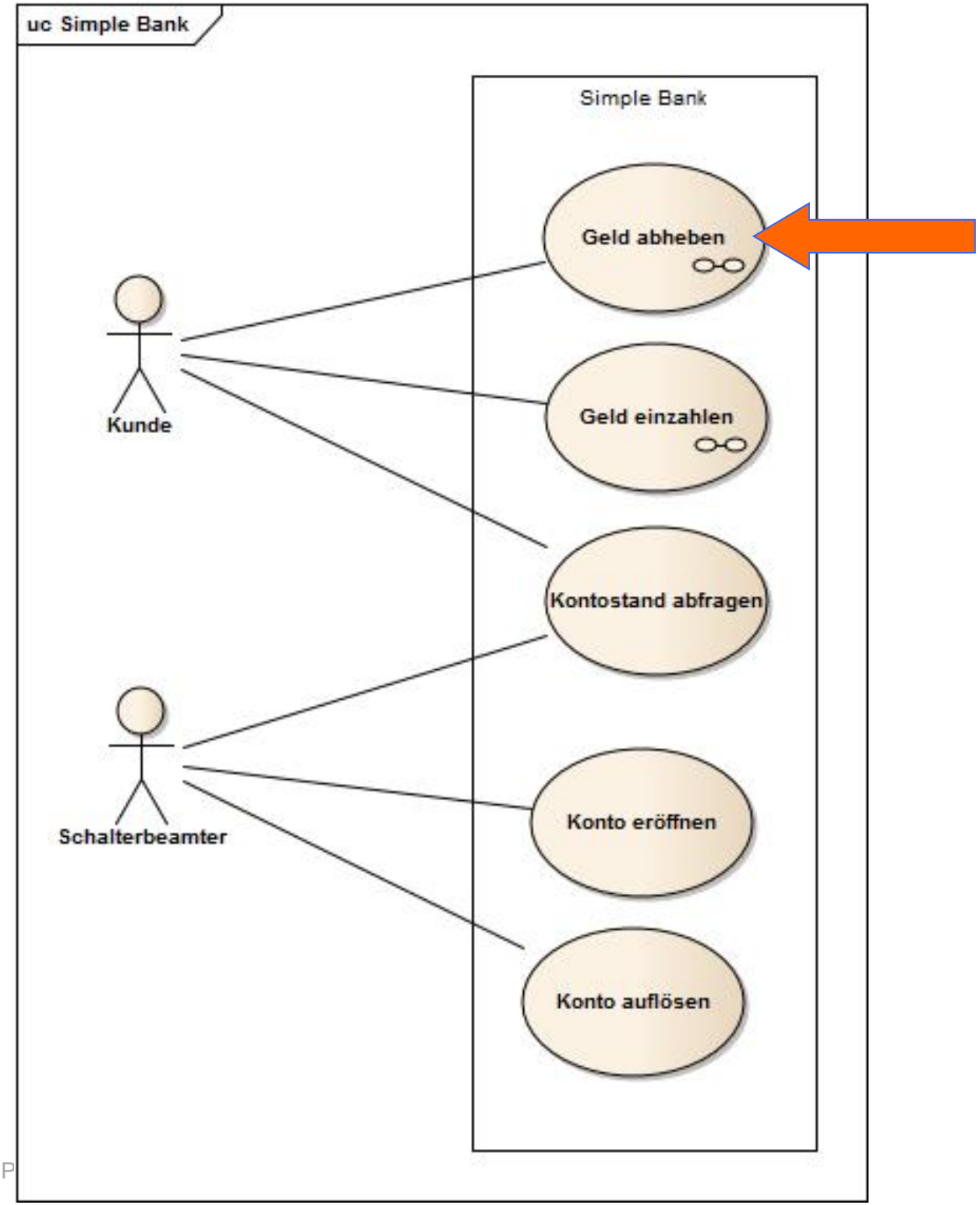




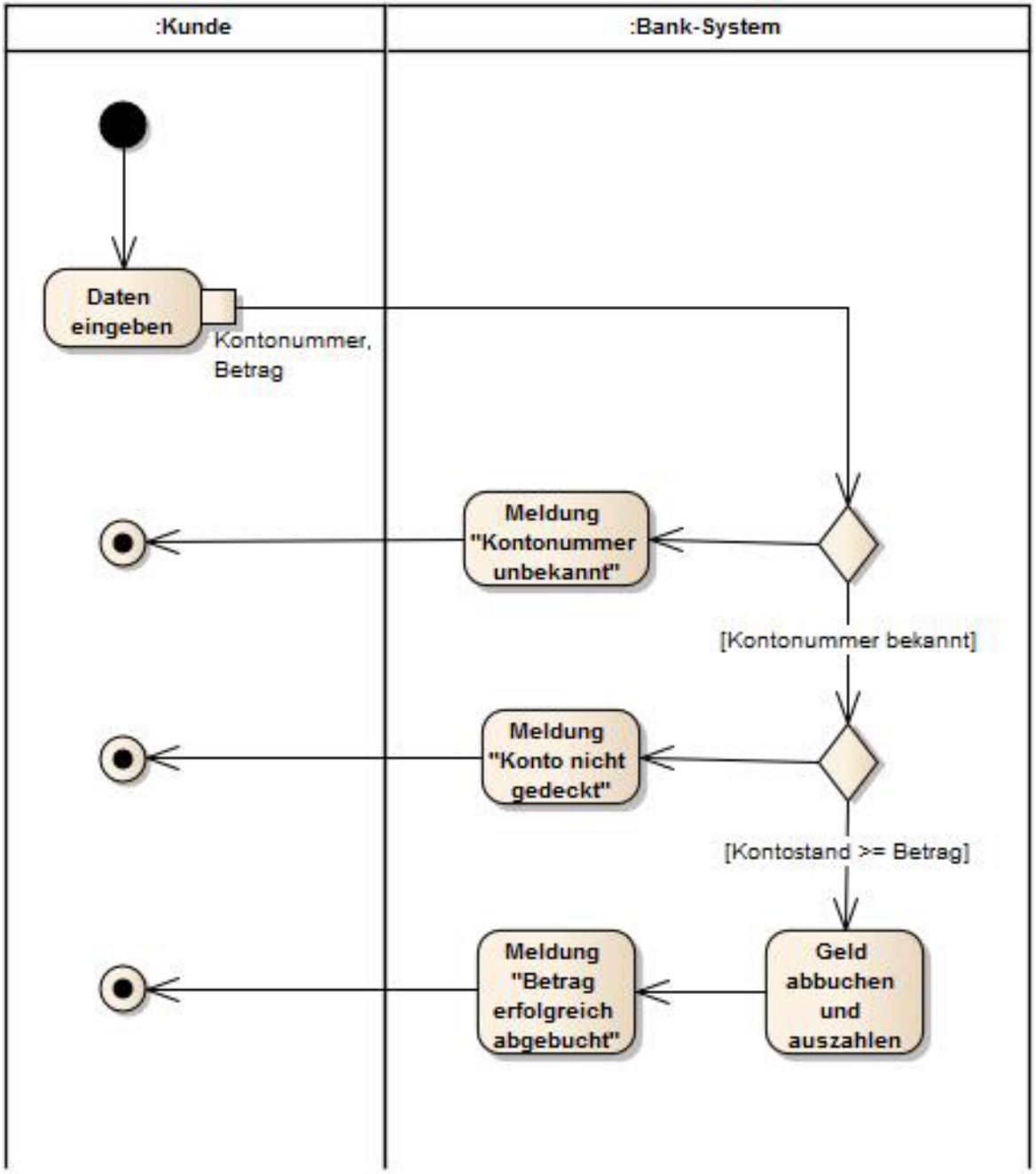
F R A G E N

Anforderungsbeschreibung: Simple Bank

- Modelliert wird ein einfaches Bank-System.
- Ein Kunde geht zur Bank und weist sich mit seinem Namen aus, um ein Konto zu eröffnen. Er bekommt dann eine (neue) Kontonummer zugewiesen. Für die Kontoeröffnung ist eine Mindesteinzahlung von € 50 verpflichtend.
- *Mit der Kontonummer kann ein Kunde* auf sein Konto zugreifen: Er kann *Beträge* einzahlen und *abheben* und er kann den Kontostand abfragen.
  Er kann auch sein Konto wieder auflösen.
- Die Bank verwaltet (unter Zuhilfenahme der Kontonummer) Konten, die jeweils den Namen des Besitzers und den aktuell verfügbaren Betrag in Euro speichern.


uc Simple Bank
Simple Bank
Geld abheben
Geld einzahlen
Kontostand abfragen
Konto eröffnen
Konto auflösen
Kunde
Schalterbeamter

Geld abheben



Überlegungen für das Design

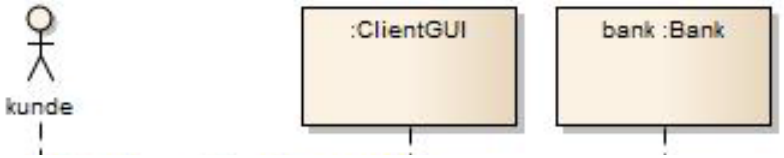
Analyse

- Ein *Use Case*-Diagramm sagt, *wer was* mit dem (Bank-) System tun kann
- Ein Aktivitätsdiagramm pro Aktivität im *Use Case*-Diagramm beschreibt, *was* normalerweise bzw. in Ausnahmesituationen *passiert*

Design

- Wir überlegen uns, *welche Klassen* (zunächst *ohne Methoden und Felder*!) wir vermutlich benötigen werden:
  - eine Klasse für die GUI
  - eine Klasse für die Bank
  - eventuell weitere Hilfsklassen (die wir aber noch nicht kennen)
- Wir überlegen uns, *wie* diese Klassen *zusammenarbeiten* können / sollen (genauer: welche Methoden und Felder sie dafür brauchen) und zeichnen das in ein Sequenzdiagramm ein (sowie gleichzeitig in ein Klassendiagramm)
- Das fertige Klassendiagramm ist die Basis für die *Arbeitsteilung im Team*!

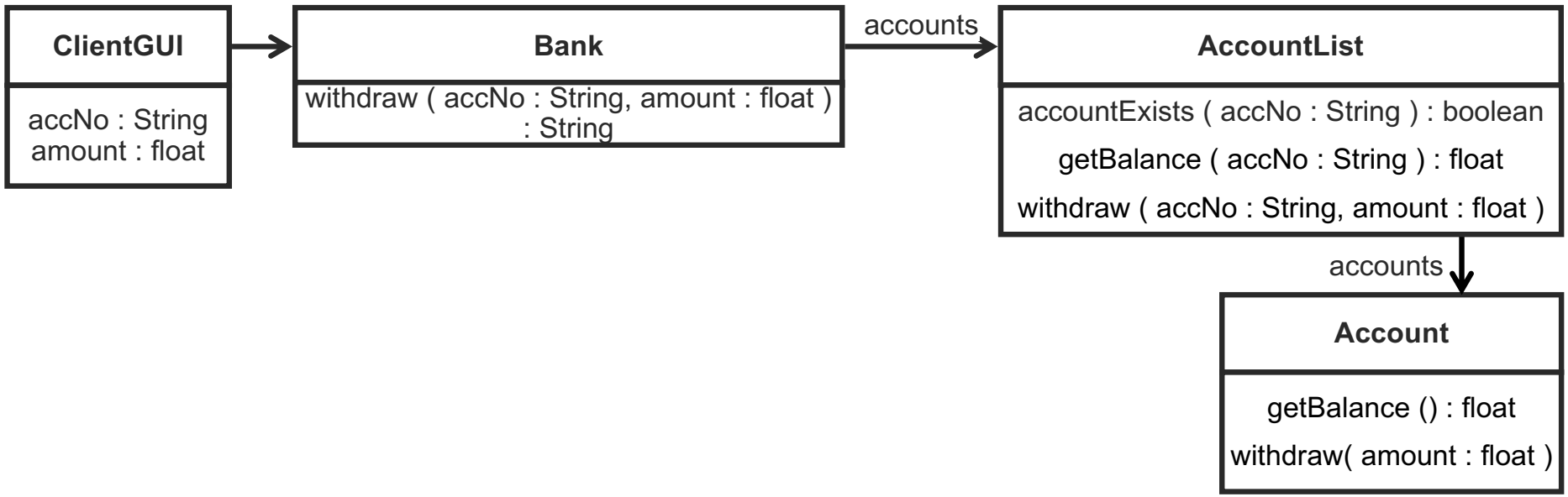
Sequenzdiagramm für die Simple Bank - *withdraw*
→ Auswirkung auf das Klassendiagramm



Oben:
Wir *entwerfen* ein Sequenzdiagramm

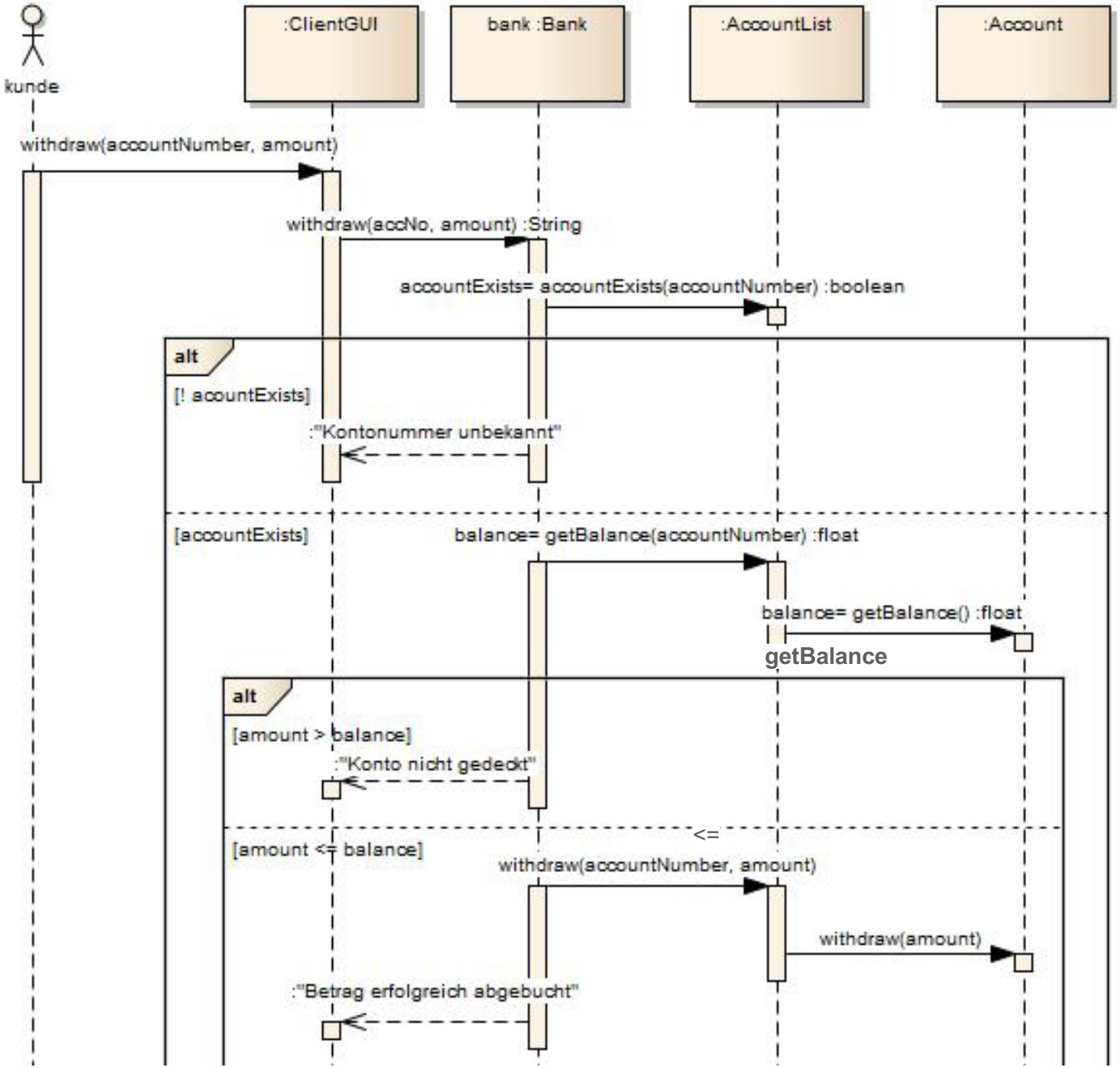
Unten:
Es *entsteht* ein Klassendiagramm

Ergebnis:
(partielles) Klassendiagramm für die Simple Bank

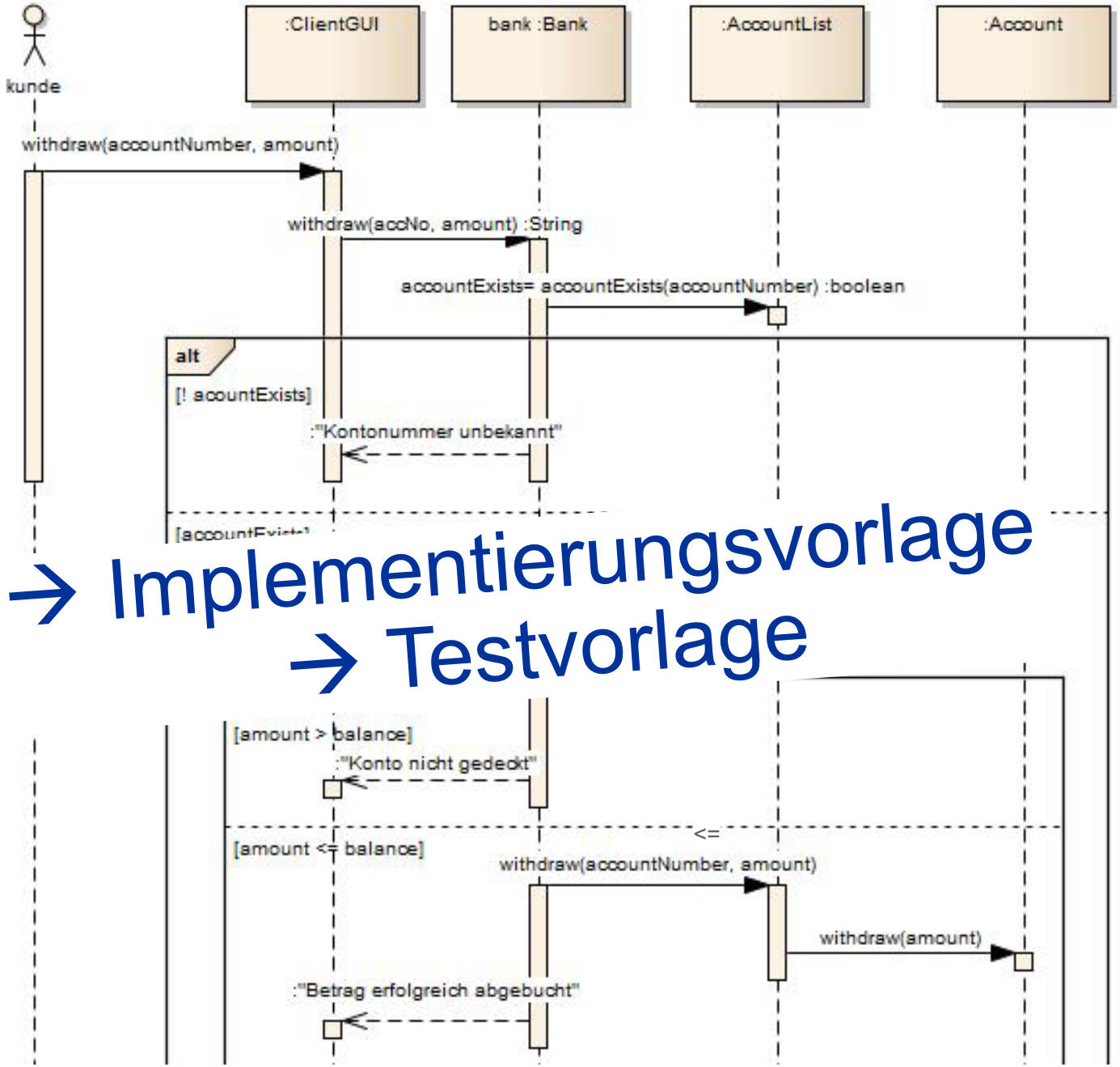


→ ***funktionierende*** Arbeitsaufteilung zwischen den Klassen!

Treiber für die Design-Entwicklung: Sequenzdiagramm für die Simple Bank - *withdraw*



Treiber für die Design-Entwicklung: Sequenzdiagramm für die Simple Bank - *withdraw*



→ Implementierungsvorlage

→ Testvorlage

F R A G E N