universitätbonn ROOTS

Ferientutorien Softwaretechnologie 2010

von Eva Stöwe und Jan Nonnen

Disclaimer

- Die Folien
  - stellen lediglich die Basis der jeweiligen Themen dar
  - erheben keinen Anspruch auf Vollständigkeit
- An mehreren Stellen wurden Sachen vereinfacht oder weggelassen.
- Wir raten dringend dazu, auch andere Quellen zur Klausurvorbereitung zu nutzen.

Klausurrelevant sind die Vorlesungsfolien.

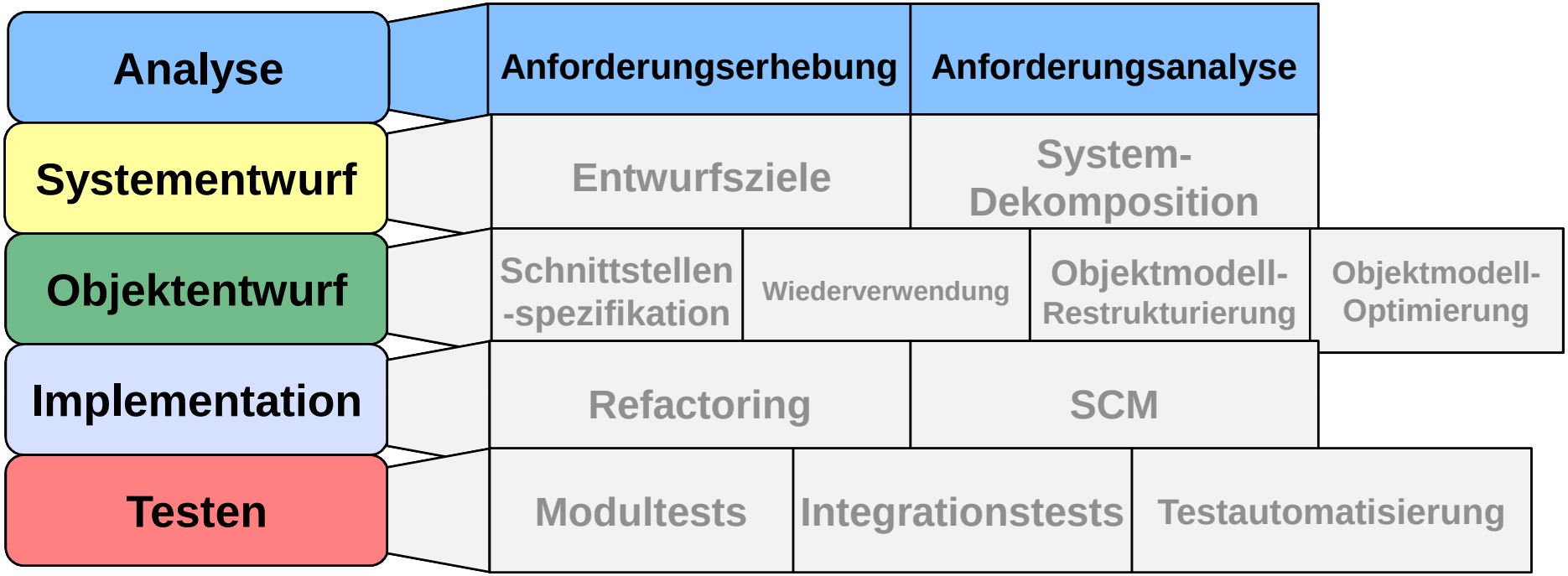
universitätbonn ROOTS

Analysephase

von Eva Stöwe und Jan Nonnen

18.03.2010

Entwicklungsphasen Überblick



Analyse ▸ Ziele

- Ziele (Was?)
  - Kennenlernen der **Anwendungsdomäne**
  - Bestimmung der **Anforderungen an das System**

- Workflows (Wie?)
  - **Anforderungserhebung**
    - ⇨ Definition des Systems in einer Form, die Kunden und Entwickler verstehen
  - **Anforderungsanalyse**
    - ⇨ Technische Spezifikation des Systems in einer für die Entwickler verständlichen und handhabbaren Form

universitätbonn ROOTS

Anforderungserhebung

Erhebung ▸ Anforderungen

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

- Herausforderung:
  - Zusammenarbeit von
    - ⇨ Kunden (Anwendungsdomäne)
    - ⇨ Entwickler (Implementierungsdomäne)
  - Verbindung zwischen den Domänen durch **Szenarios** und **Use-Cases**

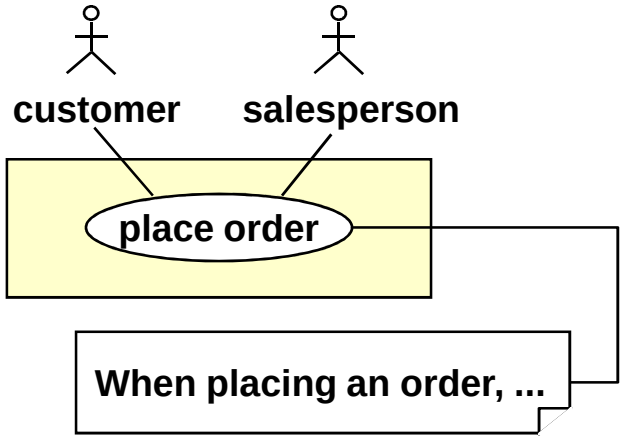
- Ergebnis der Anforderungserhebung:
  - **Use Case Model (Analyse-Verhaltensmodell)**
  - **Domain Object Model (DOM)**
  - **Nicht-Funktionale-Anforderungen**
  - **Glossar**

Erhebung ▸ Anforderungen

- Funktionale Anforderungen
  - Beschreiben die Interaktionen zwischen System und Umgebung

- Nichtfunktionale Anforderungen
  - Für den Nutzer sichtbare Aspekte des Systems, die nicht direkt mit dem funktionalen Verhalten in Beziehung stehen.
    - ⇨ Reaktionszeit
    - ⇨ Ressourcenbedarf
    - ⇨ Genauigkeit
    - ⇨ Sicherheitsanforderungen
    - ⇨ Skalierbarkeit
    - ⇨ Zuverlässigkeit
    - ⇨ Look & Feel

Use Case Modell



Erhebung ▸ Szenarios und Use-Cases

- **Szenario**

  Konkrete, fokussierte und informelle Beschreibung

  - einer einzelnen Funktionalität eines Systems,
  - aus Sicht eines einzelnen Akteurs

- **Use Case**

  Abstraktion, die eine Menge von Szenarien beschreibt

  - Modellieren ein System aus Sicht des Nutzers
  - Bilden Basis für den gesamten Entwicklungsprozess
  - Textuelle Beschreibung mit Hilfe von UML Diagrammen

Erhebung ▸ Szenarien finden

- Fragen an den Kunden:
  - 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, beobachten, wie Aufgaben damit bearbeitet werden
  - Gespräch mit dem Endbenutzer, nicht nur mit Auftraggeber

Erhebung ▸ Use Case - Schema

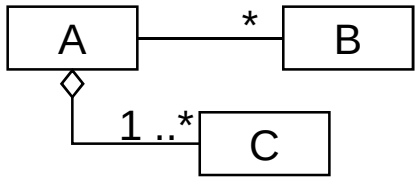
- Name
- Kurzbeschreibung
- Akteure
- Vorbedingung
- Ereignisfluss
  - Schritte im Standardablauf des Use Cases
- Nachbedingung (bei Erfolg)
- Sonderfälle (Alternativen und Fehlerfälle)
  - ⇨ Name
  - ⇨ Verzweigungspunkt im Standardablauf („extension point“)
  - ⇨ auslösende Bedingung
  - ⇨ Ereignisfluss im Sonderfall
  - ⇨ Nachbedingung
- Nichtfunktionale Anforderungen

Erhebung ▸ Übung: Use-Case Entwurf

- Szenario 1:
  - Wenn der Kunde sich für ein Urlaubsland entschieden hat, gibt die Mitarbeiterin die vom Kunden angedachte Dauer der Reise und einen Preisrahmen ein. Anschließend legt sie die gewünschten Aktivitäten (Ballon-Fahren, Kamel-Reiten, Tauchkurs, Tempelrundfahrten) fest. Danach bekommt sie vom Programm eine Anzeige passender Pauschal-Angebote. Am Ende werden die für den Kunden interessanten Angebote automatisch ausgedruckt.

- Szenario 2:
  - Wenn sich der Kunde für ein Urlaubsland entschieden und die Mitarbeiterin Dauer ,Preis und Aktivitäten festgelegt hat, findet der Kunde keine für ihn interessanten Angebote bei den angezeigten Pauschal-Angeboten. Dann soll die Möglichkeit bestehen, Angebote individuell zusammenzustellen. Dafür gibt zeigt das System eine Liste aller buchbaren Optionen an, die die Mitarbeiterin beliebig kombinieren kann. Am Ende werden alle für den Kunden interessanten Individual-Angebote automatisch ausgedruckt.

Domain Object Model (DOM)



Erhebung ▸ Domain Object Model (DOM)

- Menge „einfacher“ Klassendiagramme, die **Konzepte der Anwendungsdomäne** beschreiben
  - Klassen
  - Attribute
  - Beziehungen

- **Ziel:**
  - finden **wichtiger Abstraktionen** des Systems
  - soweit sie für **Anwender beobachtbar** sind

- Techniken:
  - Abbotts Textanalyse
  - CRC-Cards
  - …

Erhebung ▸ Textanalyse von Abbott

Sprachelement	Modellelement
Eigenname	Objekt
Nomen	Klasse
„ist“	Generalisierung
„hat“, „enthält“, …	Aggregation
Modalverb („müssen“, „können“, „dürfen“, „sollen“)	Einschränkung (*Constraint*)
Adjektiv	Attribut
Transitives Verb	Methode
Intransitives Verb	Methode (Event)

Erhebung ▸ Class-Responsibility-Collaboration Karten

- Class
  - Welchen Typ betrachten wir?
- Responsibility
  - Beschreibt die Aufgaben des Typs
- Collaboration
  - Welche anderen Typen werden für die Aufgabe gebraucht?
- Nutzen
  - Lenkt den Blick auf das Wesentliche
  - Hilft auf Schnittstelle statt Daten zu fokussieren
  - Beugt Konzentration von zu vielen Verantwortlichkeiten in einem Typ vor
- “Schreib nie mehr auf, als auf eine Karte paßt!”

Bestellung	
Prüfe ob Artikel auf Lager	Lager
Bestimme Preis	Artikel
Prüfe Zahlungseingang	Kunde, Kasse
Ausliefern	Logistik

universitätbonn ROOTS

Anforderungsanalyse

Analyse ▸ Anforderungsanalyse

- Technische Spezifikation des Systems in einer für die Entwickler verständlichen und handhabbaren Form
- Weiterentwicklung des **DOM** in Form des **Analyse-Objektmodells**
  - Klassifizierung von **Boundary/Control/Entity**
- Verfeinerung des **Analyse-Verhaltensmodells** durch
  - Interaktionsdiagrammen
  - Zustandsdiagrammen
- Wiederholtes, wechselseitiges Verfeinern des
  - **Objektmodells**
  - **Verhaltensmodells**

Analyse ▸ Abgrenzung der Artefakte

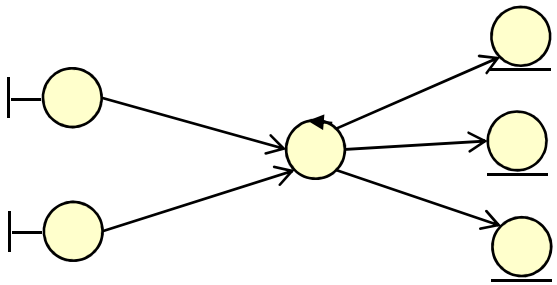
Erhebungs Modell

- Terminologie der Anwender
- Zum Verständnis der Anforderungen an das System
- externe Sicht des Systems (was tut es?)
- darf redundant / inkonsistent sein
- definiert Use Cases

Analyse Modell

- Terminologie der Entwickler
- Zum Verständnis der Struktur des System
- interne Sicht des Systems (wie tut es das?)
- darf keine Redundanzen / Inkonsistenzen enthalten
- definiert Use-Case-Realisation

Analyse-Objektmodell (BCE-Modell)



Analyse ▸ Analyse-Objektmodell (BCE)

- Weiterentwicklung des „Domain Object Model“

- Typen im Analyse-Objektmodell
  - realisieren essentielle funktionale Anforderungen
  - abstrahieren Konzepte oder Subsysteme mit
    - ⇨ konzeptuellen Attributen die evtl. später zu eigenen Klassen werden
    - ⇨ noch relativ wenigen Operationen

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

Layout

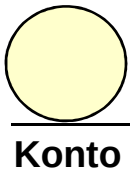
- Boundaries links - Controller in der Mitte - Entities rechts

Analysetyp ▸ Entity

- relevantes **Konzept** der **Anwendungsdomäne**

- Heuristik für Entity-Kandidaten
  - Typen im DOM
  - Begriffe im Glossar
  - In verschiedenen Use Cases wiederkehrende Substantive

«entity»
Konto



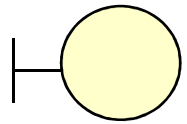
- oft persistente Informationen der Anwendung
  - ergibt sich aus der Use Case übergreifenden Bedeutung
  - aber oft keine reinen Daten-Klassen
  - können komplexes Verhalten besitzen

Analysetyp ▸ Boundary

- Beschreibt Interaktionen zwischen System und Akteuren

- Heuristik für Boundaries-Kandidaten
  - Akteure sollte nur mit Boundaries kommunizieren
  - Boundary sollte mit mindestens einem Akteur kommunizieren
  - Dateneingabeformulare und Fenster
  - Fragen, Nachrichten an den Nutzer

«boundary»
Touchscreen



Touchscreen

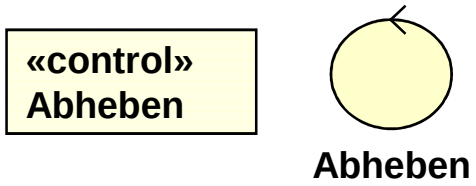
Granularität

- So fein wie nötig
  - Einheiten trennen, die im Ereignisfluss als unterschiedlich erkennbar sein müssen
- So grob wie möglich
  - Möglichst große logische Einheiten zusammenfassen

Analysetyp ▸ Control

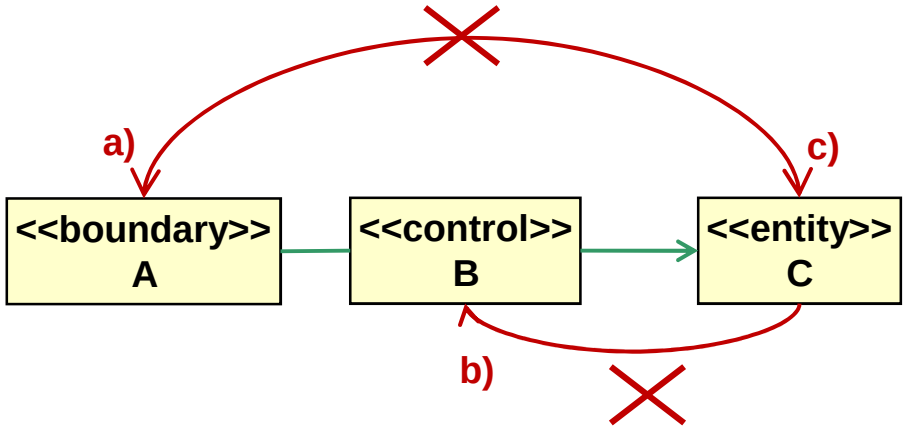
- Kapselt den **Ereignisfluss** eines Use-Case
  - Vermittlungsstelle zwischen Entities und Boundaries
  - Komplexe Berechnungen oder logische Zusammenhänge, die keiner Entity zugeordnet werden können

- Heuristik für Control-Kandidaten
  - Ein Kontrollobjekt pro Use Case



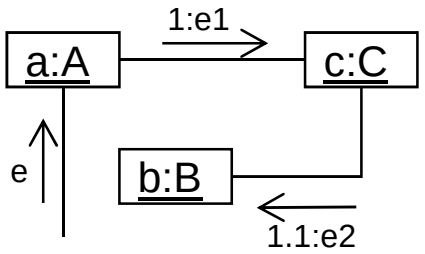
- Ausnahmen
  - um problemgebundene physikalische Verteilung auszudrücken
  - oder für sehr komplexe Use Cases

Analyse ▸ Verbotene Abhängigkeiten



- zu c)
  - Boundaries sollen keine Kontrollaufgaben übernehmen
  - Wartbarkeit und automatische Codeerzeugung für graph. Oberflächen
- zu a,b)
  - Entity-Typen sind unempfindlich gegen Änderungen in anderen Typen
  - Nutzung eines Entity-Typs in mehreren Use Cases (= Control-Typen) möglich

Analyse-Verhaltensmodell



Analyse ▸ Analyse-Verhaltensmodel

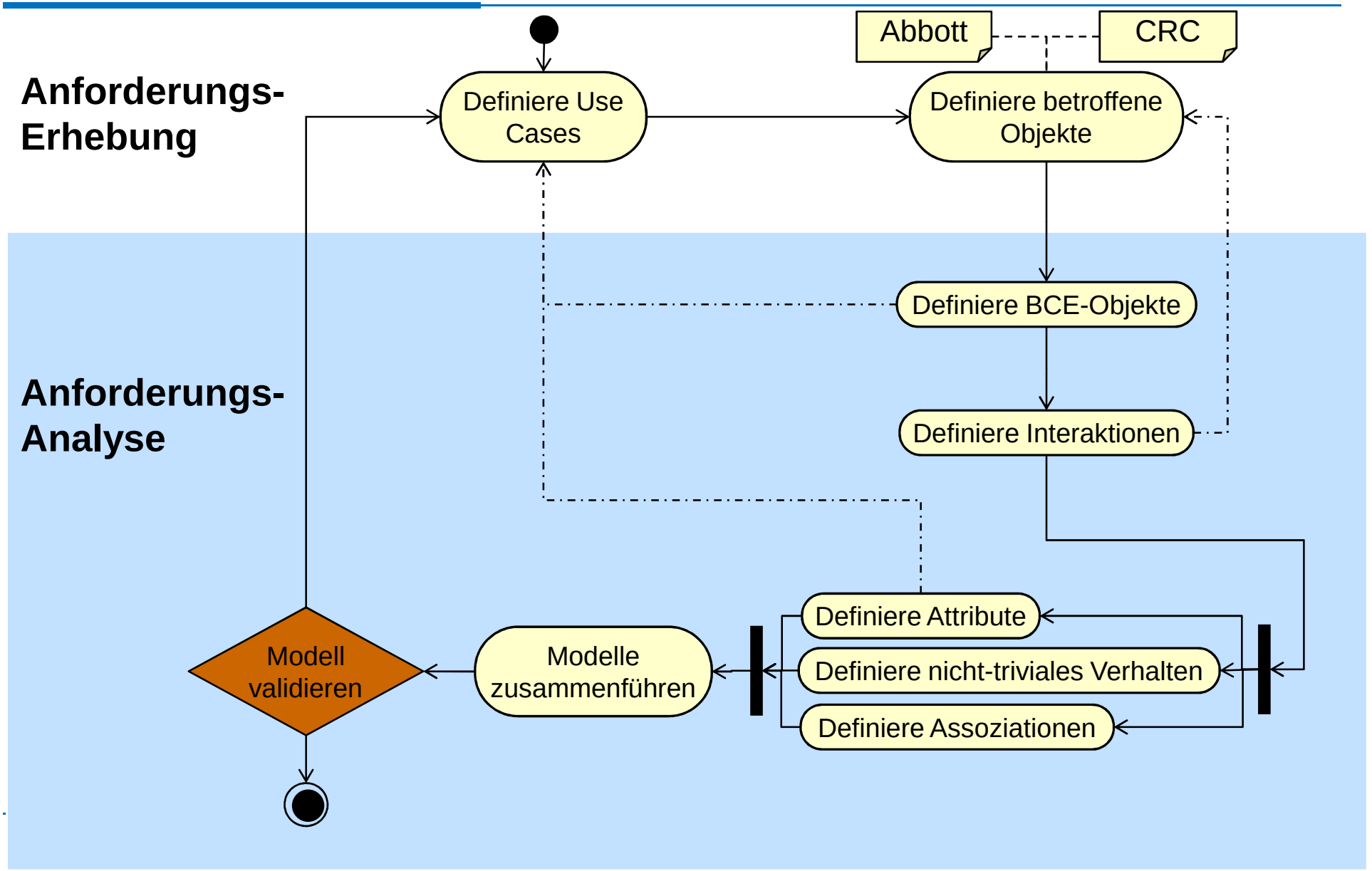
- Sammlung von
  - je einem Interaktionsdiagramm pro Ereignisfluss jedes wichtigen Use Cases
  - je einem Zustandsdiagramm für Typen mit komplexen Verhalten

- Zweck
  - Identifikation von Methoden für das Objektmodell
  - Präzisierung für Methodenimplementierungen
  - Verfeinerung des Objektmodells

Analyse ▸ Verfahren (pro Use Case)

- Bilde Ereignisfluss auf Interaktionen betroffener Analyse-Objekte ab
  - ➔ Kommunikationsdiagramm
  - ◆ informelle Aktionen durch konkrete Nachrichten ersetzen
  - ◆ Verantwortlichkeiten der Objekte überdenken
    - ⇨ Zuordnung von Nachrichten anpassen
- Modelliere Ereignisfluss einzelner Objekte
  - ➔ Zustandsdiagramm
- Methoden der Objekte ergänzen
  - ➔ Anpassen des Objekt-/Klassendiagramms
  - ◆ mehr Zwischenschritte (neue Methoden und Nachrichten)
  - ◆ mehr Strukturdetails (neue Attribute, Parameter, Typen)

Analyse-Workflow



Ende der Analysephase

- 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