Software Entwicklung 1

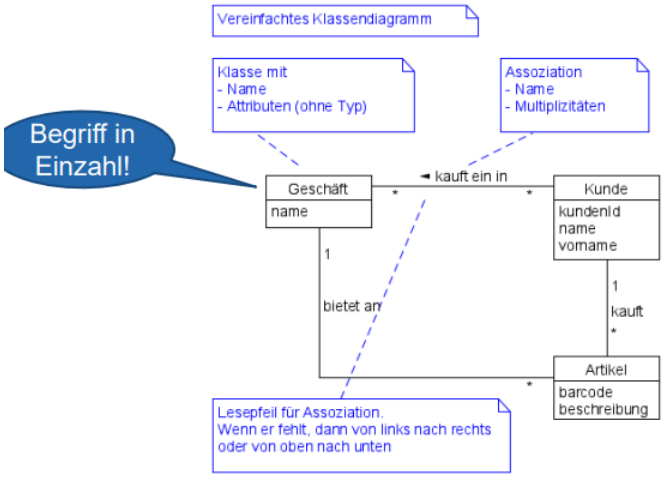
Jil Zerndt, Lucien Perret
January 2025

Domänenmodellierung

Domänenmodell

Ein Domänenmodell ist ein vereinfachtes UML-Klassendiagramm zur Darstellung der Fachdomäne:

- Konzepte als Klassen
- Eigenschaften als Attribute (ohne Typangabe)
- Beziehungen als Assoziationen mit Multiplizitäten



Domänenmodell Erstellung

Schritt 1: Konzepte identifizieren

- Substantive aus Anforderungen extrahieren
- Kategorien prüfen:
  - Physische Objekte (Produkte, Geräte)
  - Kataloge und Listen
  - Container (Warenkorb, Lager)
  - Externe Systeme
  - Rollen von Personen
  - Artefakte (Pläne, Dokumente, Verträge)
  - Zahlungsinstrumente (Transaktionen)
  - **Wichtig:** Keine Softwareklassen modellieren!
- Irrelevante Konzepte ausschließen
- Synonyme vereinheitlichen

Schritt 2: Attribute definieren

- Nur wichtige/zentrale Attribute
- Typische Kategorien:
  - Transaktionsdaten
  - Teil-Ganzes Beziehungen
  - Beschreibungen
  - Verwendungszwecke
- **Wichtig:** Beziehungen als Assoziationen, nicht als Attribute!
- Keine technischen IDs
- Keine abgeleiteten Werte

Schritt 3: Beziehungen modellieren

- Assoziationen zwischen Konzepten identifizieren
- Multiplizitäten festlegen
- Art der Beziehung bestimmen
- Richtung der Assoziation falls nötig
- Rollen an Assoziationsenden benennen

Domänenmodell Zweck

- Visualisierung der Fachdomäne für alle Stakeholder
- Grundlage für das spätere Softwaredesign
- Gemeinsames Verständnis der Begriffe und Zusammenhänge
- Dokumentation der fachlichen Strukturen
- Basis für die Kommunikation zwischen Entwicklung und Fachbereich

Analysemuster im Domänenmodell

Analysemuster im Überblick

Details siehe nächste Seite

Bewährte Strukturen für wiederkehrende Modellierungssituationen:

- **Beschreibungsklassen:** Trennung von Typ und Instanz
- **Generalisierung:** ïst-ein"Beziehungen
- **Komposition:** Starke Teil-Ganzes Beziehung
- **Zustände:** Eigene Zustandshierarchie
- **Rollen:** Verschiedene Funktionen eines Konzepts
- **Assoziationsklasse:** Attribute einer Beziehung

Musterauswahl und Kombination

Systematisches Vorgehen bei der Anwendung von Analysemustern:

1. **Analyse der Situation**
   - Konzepte und Beziehungen identifizieren, Attribute zuordnen
   - Probleme im einfachen Modell erkennen (Bei Kombination)
2. **Passende Muster identifizieren**
   - Beschreibungsklassen bei gleichartigen Objekten
   - Generalisierung bei ïst-einBeziehungen
   - Komposition bei existenzabhängigen Teilen
   - Zustände bei Objektlebenszyklen
   - Rollen bei verschiedenen Funktionen
   - Assoziationsklassen bei Beziehungsattributen
   - Wertobjekte bei komplexen Werten
3. **Musterauswahl**
   - Vor- und Nachteile abwägen
   - Komplexität vs. Nutzen bewerten
4. **Muster anwenden**
   - Struktur des Musters übernehmen, an Kontext anpassen
   - Konsistenz und fachliche Korrektheit sicherstellen
5. **Muster kombinieren**
   - An Kontext anpassen und mit bestehenden Elementen verbinden
   - Überschneidungen identifizieren und Konflikte auflösen
   - Gesamtmodell harmonisieren
   - Konsistenz und fachliche Korrektheit sicherstellen

Prüfungsaufgabe: Konzeptidentifikation

Aufgabentext: Ën Bibliothekssystem verwaltet Bücher, die von Mitgliedern ausgeliehen werden können. Jedes Buch hat eine ISBN und mehrere Exemplare. Mitglieder können maximal 5 Bücher gleichzeitig für 4 Wochen ausleihen. Bei Überschreitung wird eine Mahngebühr fällig."

Identifizierte Konzepte: Buch (Beschreibungsklasse), Exemplar (Physisches Objekt), Mitglied (Rolle), Ausleihe (Transaktion), Mahnung (Artefakt)

Begründung:

- Buch/Exemplar:
  Trennung wegen mehrfacher Exemplare (Beschreibungsmuster)
- Ausleihe: Verbindet Exemplar und Mitglied, hat Zeitbezug
- Mahnung: Entsteht bei Fristüberschreitung

Review eines Domänenmodells

Checkliste für die Überprüfung:

- **Fachliche Korrektheit**
  - Alle relevanten Konzepte vorhanden?
  - Begriffe aus der Fachdomäne verwendet?
  - Beziehungen fachlich sinnvoll?
- **Technische Korrektheit**
  - UML-Notation korrekt?
  - Multiplizitäten angegeben?
  - Assoziationsnamen vorhanden?
- **Modellqualität**
  - Angemessener Detaillierungsgrad?
  - Analysemuster sinnvoll eingesetzt?
  - Keine Implementation vorweggenommen?

Typische Modellierungsfehler vermeiden

- **Keine Softwareklassen modellieren**
  - Manager-Klassen vermeiden
  - Keine technischen Helper-Klassen
- **Keine Operationen modellieren**
  - Fokus auf Struktur, nicht Verhalten
  - Keine CRUD-Operationen
- **Richtige Abstraktionsebene wählen**
  - Nicht zu detailliert oder abstrakt
  - Fachliche Begriffe verwenden
- **Assoziationen statt Attribute**
  - Beziehungen als Assoziationen modellieren
  - Keine Objekt-IDs als Attribute

Typische Modellierungsfehler

Fehler 1: Technische statt fachliche Klassen

- **Falsch:** CustomerManager, OrderController, DatabaseHandler
- **Richtig:** Kunde, Bestellung, Produkt

Fehler 2: IDs als Attribute statt Assoziationen

- **Falsch:** customerId: String, orderId: Integer
- **Richtig:** Direkte Assoziation zwischen Kunde und Bestellung

Fehler 3: Implementierungsdetails

- **Falsch:** saveToDatabase(), validateInput(), createPDF()
- **Richtig:** Keine Operationen im Domänenmodell

Typische Prüfungsaufgabe: Modell verbessern

Fehlerhaftes Modell:

- Klasse 'UserManager' mit CRUD-Operationen
- Attribute 'customerID' und 'orderID' statt Assoziationen
- Operation 'calculateTotal()' in Bestellung
- Technische Klasse "DatabaseConnection"

Verbesserungen:

- 'UserManager' entfernen, stattdessen Beziehungen modellieren
- IDs durch direkte Assoziationen ersetzen
- Operationen entfernen (gehören ins Design)
- Technische Klassen entfernen

Detaillierte Analysemuster

1. Beschreibungsklassen

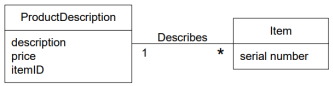
Trennt die Beschreibung eines Typs von seinen konkreten Instanzen.

Anwendung:

- Bei mehreren gleichartigen Objekten
- Gemeinsame unveränderliche Eigenschaften
- Vermeidung von Redundanz

Beispielstruktur:

- ProductDescription (Typ)
  - price, description, itemID
- Product (Instanz)
  - serialNumber



Beschreibungsklassen in der Praxis

Szenario: Bibliothekssystem

Problem: Ein Buch kann mehrere physische Exemplare haben, die alle dieselben Grunddaten (Titel, Autor, ISBN) aber unterschiedliche Zustände (ausgeliehen, verfügbar) haben.

Lösung:

- **Book** (Beschreibungsklasse)
  - title, author, isbn, publisher
- **BookCopy** (Instanzklasse)
  - inventoryNumber, status, location
- Assoziation: BookCopy "beschrieben durch"Book

2. Generalisierung/Spezialisierung

Regeln:

- 100% Regel: Jede Instanz der Spezialisierung ist auch Instanz der Generalisierung
- 'IS-A' Beziehung
- Gemeinsame Attribute/Assoziationen als Grund für Generalisierung
- Gemeinsame Eigenschaften in Basisklasse
- Spezifische Eigenschaften in Unterklassen

Beispiele:

- Person → Student, Dozent
- Zahlung → Barzahlung, Kreditkartenzahlung
- Dokument → Rechnung, Lieferschein

Generalisierung im Online-Shop

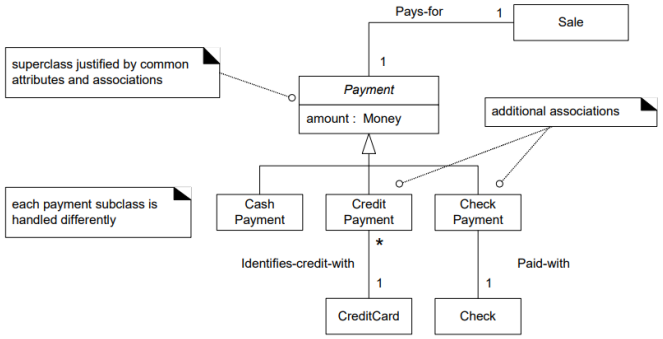
Szenario: Versch. Zahlungsarten

Struktur:

- **Payment** (abstrakt)
  - amount, date, status
- **CashPayment**
  - receivedAmount, changeAmount
- **CreditCardPayment**
  - cardType, authorizationCode

Begründung:

- Gemeinsame Attribute in Payment
- Spezifische Attribute in Unterklassen
- Jede Zahlung ist genau ein Typ



3. Komposition

Modelliert eine starke Teil-Ganzes Beziehung mit Existenzabhängigkeit der Teile.

Eigenschaften:

- Teile können nicht ohne Ganzes existieren
- Teil gehört zu genau einem Ganzen
- Löschen des Ganzen löscht alle Teile

Notation:

- Ausgefüllte Raute am "GanzesEnde
- Multiplizität am "TeilEnde

Komposition im Bestellsystem

Szenario: Bestellung mit Bestellpositionen

Struktur:

- **Order**
  - orderDate, status
- **OrderItem**
  - quantity, price
- Komposition von Order zu OrderItem (1 zu *)

Begründung:

- OrderItems existieren nur im Kontext einer Order
- Löschen der Order löscht alle OrderItems
- Ein OrderItem gehört zu genau einer Order

4. Zustandsmodellierung

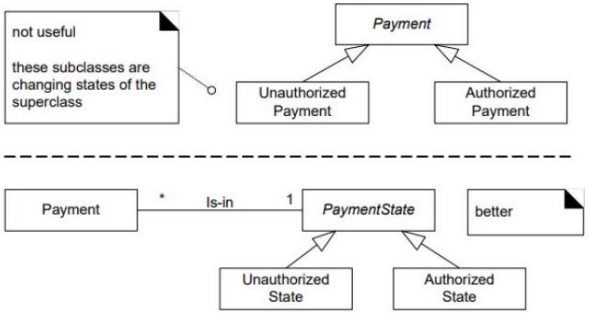
Modelliert Zustände als eigene Konzepthierarchie statt als Attribut.

Vorteile:

- Klare Strukturierung der Zustände
- Vermeidet problematische Vererbung
- Erweiterbarkeit durch neue Zustandsklassen
- Vermeidung von if/else Kaskaden
- Zustandsspezifisches Verhalten möglich

Beispielstruktur:

- OrderState (abstrakt)
  - New, InProgress, Completed
- Order ïst inÖrderState



Zustandsmodellierung: Ticketsystem

Szenario: Support-Tickets mit verschiedenen Status

Falsche Modellierung:

```
class Ticket {
    enum Status {NEW, OPEN, IN_PROGRESS,
        RESOLVED, CLOSED}
    private Status status;
}
```

Bessere Modellierung:

- **TicketState** (abstrakt)
  - timestamp, changedBy
- Konkrete Zustände:
  - NewState: assignedTo
  - OpenState: priority
  - InProgressState: estimatedCompletion
  - ResolvedState: solution
  - ClosedState: closureReason

5. Rollen

Modelliert verschiedene Funktionen eines Konzepts.

Varianten:

- Rollen als eigene Konzepte
- Rollen als Assoziationsenden
- Rollen durch Generalisierung

Anwendung:

- Bei verschiedenen Verantwortlichkeiten
- Wenn Rollen wechseln können
- Bei unterschiedlichen Beziehungen

Rollenmuster: Universitätssystem

Szenario: Person kann gleichzeitig Student und Tutor sein

Variante 1: Rollen als Konzepte

- **Person**
  - name, birthDate, address
- **StudentRole**
  - matriculationNumber, program
- **TutorRole**
  - department, hourlyRate

Variante 2: Generalisierung

- Person als Basisklasse
- Student und Tutor als Spezialisierungen
- Problem: Mehrfachrollen schwierig

6. Assoziationsklassen

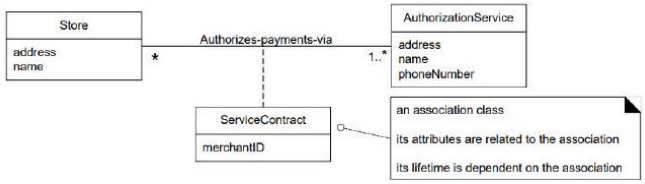
Modelliert Attribute einer Beziehung zwischen Konzepten, eigene Klasse für die Assoziation

Einsatz wenn:

- Attribute zur Beziehung gehören
- Beziehung eigene Identität hat
- Mehrere Beziehungen möglich sind

Notation:

- Gestrichelte Linie zur Assoziation
- Klasse enthält beziehungsspezifische Attribute



Assoziationsklasse: Kursbuchungssystem

Szenario: Studenten können sich für Kurse einschreiben

Struktur:

- **Student** und **Course** als Hauptkonzepte
- **Enrollment** als Assoziationsklasse:
  - enrollmentDate
  - grade
  - attendance
  - status

Begründung:

- Noten gehören zur Einschreibung
- Student kann mehrere Kurse belegen
- Kurs hat mehrere Studenten
- Einschreibungsdaten sind beziehungsspezifisch

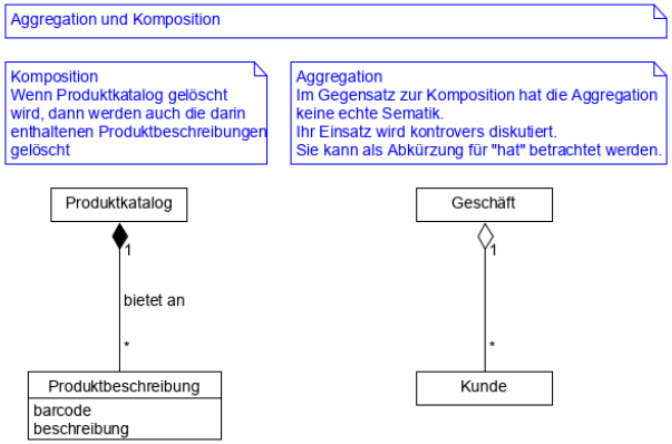
7. Wertobjekte

- Masseinheiten als eigene Konzepte
- Zeitintervalle als Konzepte
- Vermeidet primitive Obsession

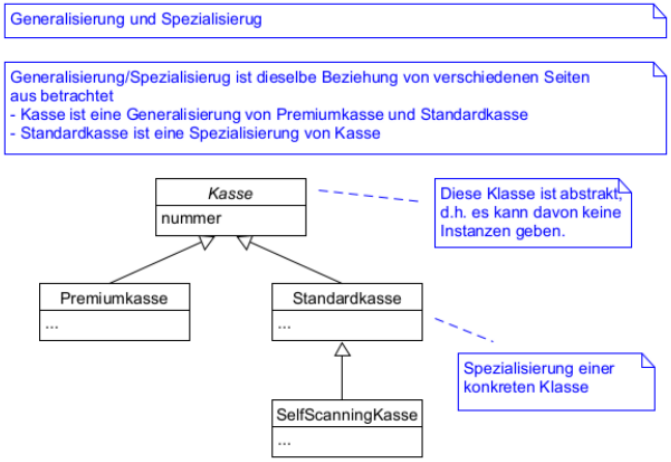
Mehr Info und Beispiele

Optionale Elemente im Domänenmodell

- Optional: Aggregationen/Kompositionen



- Optional: Generalisierung/Spezialisierung



Vollständige Beispiele Domänenmodell

Domänenmodell Online-Shop

Aufgabe: Erstellen Sie ein Domänenmodell für einen Online-Shop mit Warenkorb-Funktion.

Lösung:

- **Konzepte identifizieren:**
  - Artikel (physisches Objekt)
  - Artikelbeschreibung (Beschreibungsklasse)
  - Warenkorb (Container)
  - Bestellung (Transaktion)
  - Kunde (Rolle)
- **Attribute:**
  - Artikelbeschreibung: name, preis, beschreibung
  - Bestellung: datum, status
  - Kunde: name, adresse
- **Beziehungen:**
  - Warenkorb gehört zu genau einem Kunde (Komposition)
  - Warenkorb enthält beliebig viele Artikel
  - Bestellung wird aus Warenkorb erstellt

Komplexes Domänenmodell: Reisebuchungssystem

Anforderung: Modellieren Sie ein System für Pauschalreisen mit Flügen, Hotels und Aktivitäten.

Verwendete Analysemuster:

- **Beschreibungsklassen:**
  - Flugverbindung vs. konkreter Flug
  - Hotelkategorie vs. konkretes Zimmer
  - Aktivitätstyp vs. konkrete Durchführung
- **Zustände:**
  - Buchungszustände: angefragt, bestätigt, storniert
  - Zahlungszustände: offen, teilbezahlt, vollständig
- **Rollen:**
  - Person als: Kunde, Reiseleiter, Kontaktperson
- **Wertobjekte:**
  - Geldbetrag mit Währung
  - Zeitraum für Reisedauer