

C.7 Objektorientierte Software-Entwicklung

- Objektorientiertes Software -Engineering
- Phasen der Software-Entwicklung
 - Anforderungsanalyse
 - Objektorientierte Analyse
 - Objektorientierter Entwurf
 - Implementierung
 - Test
- UML: The Unified Modeling Language — ein kurzer Abriss
 - Use-Case Diagramme
 - Klassendiagramme
 - Interaktionsdiagramme

1 Objektorientiertes Software-Engineering

- 1980 - 1990: Entwicklung von OO Programmiersprachen
 - Smalltalk
 - C++
 - Eiffel
 - LOOPS, Flavors
- ab 1990: Verbreitung von objektorientierten Software-Engineering Methoden
 - Sally Shlaer & Steve Mellor
 - Peter Coad & Ed Yourdon
 - Grady Booch
 - Jim Rumbaugh et al. (OMT: Object Modeling Technique)
 - Ivar Jacobson (OOSE, Objectory)
- 1995: Booch, Rumbaugh und Jacobsen beginnen die Entwicklung der UML

2 Warum objektorientiertes Software-Engineering?

- Optimaler Einsatz des objektorientierten Paradigmas
- Übergang auf OO Programmiersprachen ist einfach
- Das Problem ist der **Paradigmen-Wechsel**
 - Vorteile der Objektorientierung hängen davon ab, wie ein Problem angegangen wird
 - sonst Gefahr, dass OO Konzepte falsch eingesetzt werden
 - ➔ C++
- Objektorientierte Software-Engineering-Methoden helfen, objektorientierte Konzepte richtig anzuwenden
 - richtige Bestimmung der Klassen
 - richtiger Einsatz von Vererbung
 - ...

2 Warum objektorientiertes Software-Engineering? (2)

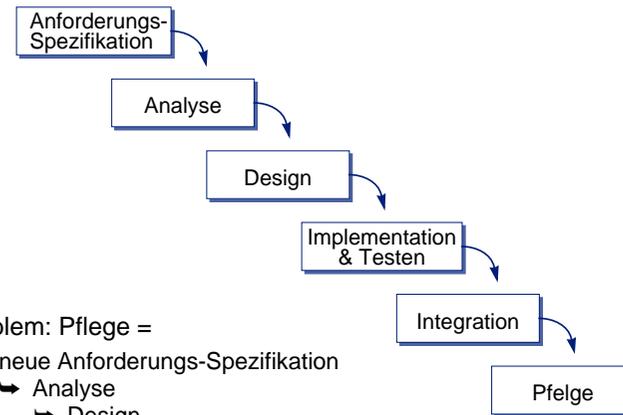
- ▲ Software häufig zu komplex um als Ganzes überblickbar zu sein
- ➔ Modelle um von Details des Systems zu abstrahieren
 - verschiedene Blickrichtungen — z. B.
 - Spezifikationsicht
 - Entwurfssicht
 - verschiedene Abstraktionsebenen
 - Abstraktionen für Systemteile
 - verschiedene Aspekte des Systems
 - Anforderungen
 - statische Aspekte — Struktur
 - dynamische Aspekte — Verhalten

2 Warum objektorientiertes Software-Engineering? (3)

- ▲ Kontrolle des Entwicklungs-Prozesses
- Unkontrollierte Entwicklung führt zu
 - nicht ausreichender Analyse der Anforderungen
 - ↳ Entwicklung stimmt nicht mit den Anforderungen des Auftraggebers überein
 - zu früher Start der Implementierung
 - ↳ Analyse und Entwurf sind noch nicht ausgereift
 - ↳ Fehler in Analyse und Entwurf werden erst später entdeckt
 - ↳ hohe Kosten für die daraus folgenden Änderungen
- ▲ Software-Engineering-Methoden geben Anleitungen
 - Komplexität heutiger Software zu beherrschen
 - Software wirtschaftlich zu entwickeln
 - die Software-Entwicklung zu einem kontrollierten, kontrollierbaren und kalkulierbaren Prozess zu machen

3 Phasen der Software-Entwicklung

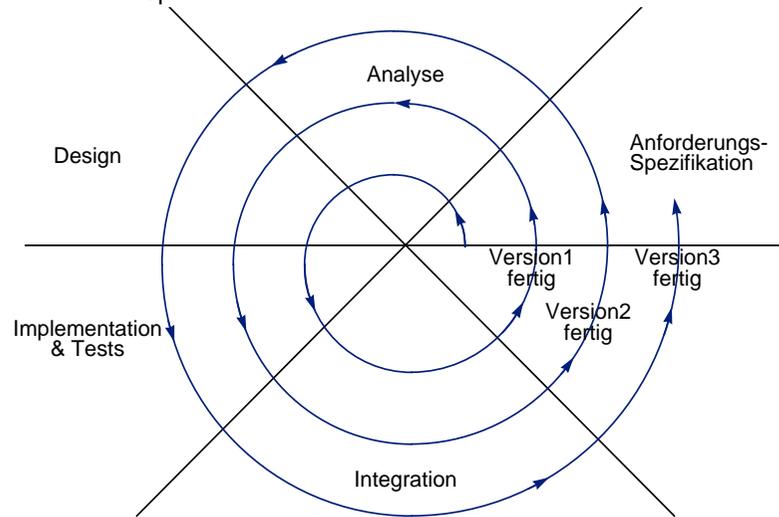
- Traditionell: Wasserfall-Ansatz



- Problem: Pflege =
 - neue Anforderungs-Spezifikation
 - ↳ Analyse
 - ↳ Design
 - ↳ neue Implementation
 - ↳ ...

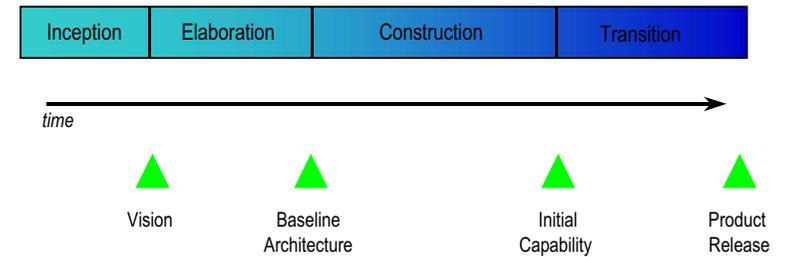
3 Phasen der Software-Entwicklung (2)

- Besser: Spiral-Modell



3 Phasen der Software-Entwicklung (3)

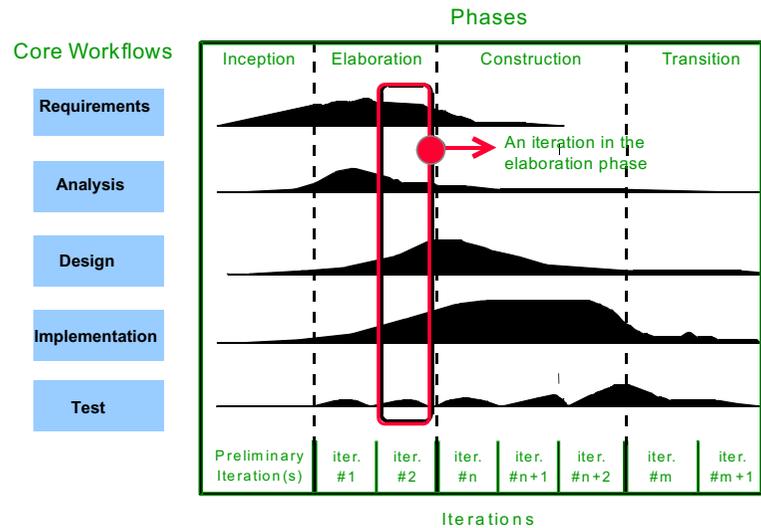
- Weiter verbessert: Iterativer Ansatz
- ▲ Vier Lebenszyklus-Phasen



- **Start (Inception):** definiere Umfang des Systems, entwerfe Geschäftsplan
- **Ausarbeitung (Elaboration):** plane Projekt, spezifiziere Eigenschaften, lege Grundlinien der Architektur fest
- **Konstruktion:** baue das Produkt
- **Transition:** übergebe das Produkt an seine Anwender

3 Phasen der Software-Entwicklung (4)

■ Iterationen und Workflows



C.8 Objektorientierte Analyse

▲ Was soll mein System tun?

■ Basis: Analyse der "realen Welt"

- Komponenten, Begriffe, Aufgaben
- Anforderungen und Einschränkungen

■ Abstraktion von unwichtigen Aspekten und Implementierungsdetails

■ Abstraktion von Implementierungsdetails

- momentan unwichtig: wie implementiere ich Dinge?
- + zu berücksichtigen: Aspekte der Implementierungs-umgebung und Ausführungsplattform
welche Mittel habe ich zur Verfügung?

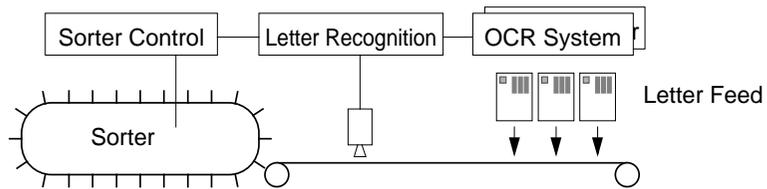
1 Der Prozess

- Analysiere Anforderungen
 - Beschreibe Use Cases
 - Lege Begriffe des "problem domain" fest
 - Finde Objekte
 - Organisiere Objekte
 - Lege erste interne Strukturen der Objekte fest
 - Beschreibe Intaktionen der Objekte
 - Lege Operationen der Objekte fest
- } Anforderungs-Modell
- } Analyse-Modell

- Fokus liegt auf dem "problem domain"
- Lege Ziele und Aufgaben fest
 - Basis für alternative Lösungen
 - Basis für Überprüfungen und Bewertungen
- Sicht des Kunden
- Performance-, Benutzer- and Architektur-bezogene Anforderungen
- Informelle Beschreibung

3 OOA — Beispiele einer Anforderungsanalyse

▲ Briefsortieranlage



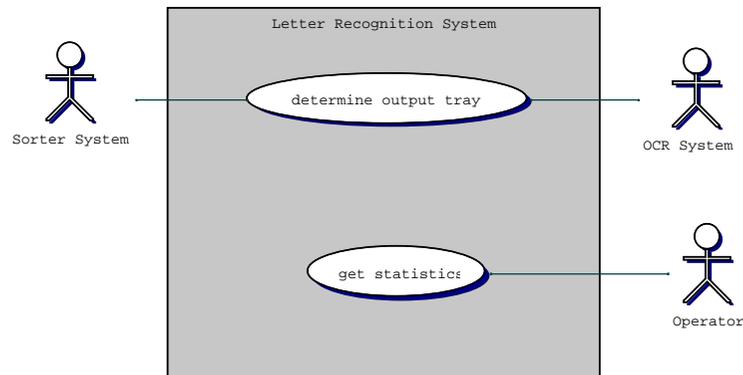
- Problem domain: Postleitzahlenerkennung
- Aufgabe: erkenne PLZ auf einem Brief und melde an Sorter Control die Nummer des Ausgabefachs
- Anforderungen:
 - Erkennung darf max. 1 Minute dauern
 - eigene OCR-Rechner für Schrifterkennung
 - Statistiken über PLZ-Häufigkeit und Fehler

4 OOA — Use Cases

- Beschreibe Interaktionen zwischen "Benutzern" und System
- Beteiligte:
 - Aktor: Person oder eine andere Komponente des Systems
! in einer bestimmten Rolle
 - Use case: "Dialog" zwischen Aktor und System um einen bestimmten Zweck zu erreichen
- weitere Festlegung der Anforderungsanalyse
- erster Schritt zur Modularisierung des Problems

4 OOA — Use Cases (2)

- Briefsortierer
 - Sorter Control fordert Nummer des Ausgabefachs an
 - Benutzer fragt Statistiken ab
- UML Diagram



5 OOA — Objekte finden

- Begriffe des "problem domain" identifizieren
 - nach Substantiven in der Terminology des "problem domain "suchen
- BEISPIEL**
- Sorter Control
 - OCR
 - Letter
 - Camera
- Strategie:
- Auftraggeber skizziert seine Sicht
 - Systemanalyst schreibt Begriffe mit
- Ziel: Basis für die weitere Arbeit erstellen
 - NICHT das ganze System beschreiben

6 OOA — Objekte organisieren

- ➔ Modell strukturieren ➔ Analysemodell
- verschiedene Kategorien von Objekten
 - ➔ Schnittstellenobjekte
 - ➔ Objekte, die Dinge repräsentieren (Entity objects)
 - ➔ Kontrollobjekte

6 OOA — Objekte organisieren (2)

Schnittstellenobjekte

- Repräsentieren Aktoren der Use Cases
 - Startpunkte für Aktivitäten im System
 - Schnittstellen des Systems zur "Aussenwelt"
- BEISPIEL
- Sorter Control
 - Operator Panel
 - OCR System

Entity Objects

- Repräsentieren den Systemzustand
 - sind verhältnismäßig langlebig
 - überdauern oft Ausführung eines Use Case
- BEISPIEL
- Statistics
 - Letter
 - Picture

6 OOA — Objekte organisieren (3)

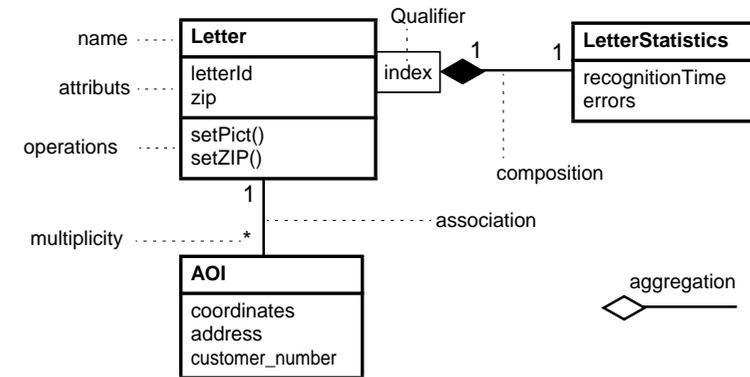
Kontrollobjekte

- Problem: eine Aufgabe oder Aktivität kann keinem der Objekte zugeordnet werden
 - ◆ Haupt-Fokus liegt auf dem Ablauf
 - ◆ solche Abläufe resultieren häufig direkt aus einem Use Case
 - definiere etwas, das für den Ablauf verantwortlich ist
 - ➔ erzeuge ein Objekt dafür
- BEISPIEL
- Aufgabe: führe PLZ-Erkennung durch
 - Bild aufnehmen
 - Adressbereich suchen
 - Auftrag an OCR-Rechner übergeben
 - Ausgabefach an sorter control melden
 - ➔ Objekt *LetterController*

6 OOA — Objekte organisieren (4)

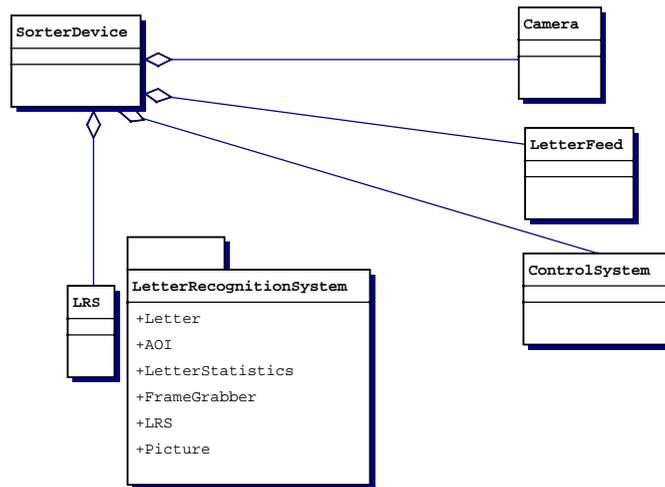
UML-Notation: Klassendiagramme

- Klassen mit ihren Attributen (Variablen) und Operationen (Methoden)
- Beziehungen zwischen Klassen
- Beispiel:



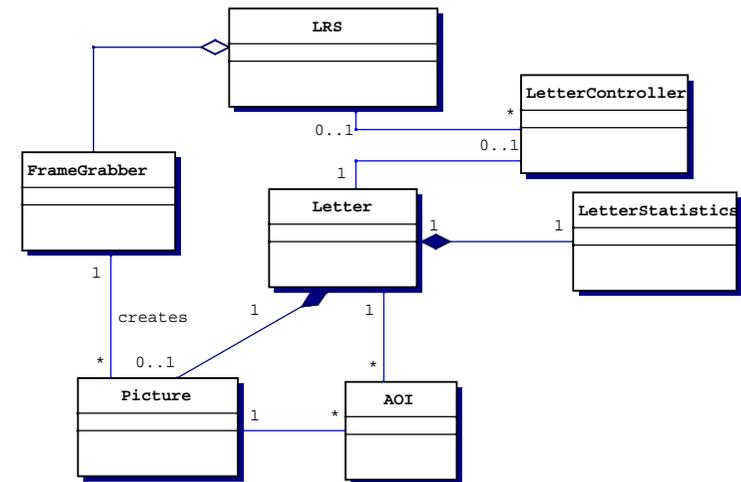
6 OOA — Objekte organisieren (5)

■ Briefsortierer — Systemüberblick: UML-Diagramm



6 OOA — Objekte organisieren (6)

■ Briefsortierer — Brieferkennung (LRS): UML-Diagramm



6 OOA — Objekte organisieren (7)

- Identifiziere Zustand von Objekten
 - Attribute (werden zu Instanzvariablen)
 - Typen
- Identifiziere Verhalten
 - Operationen / Methoden
 - Interaktion zwischen Objekten
- Suche nach Ähnlichkeiten, Gemeinsamkeiten
 - Basis für Vererbungshierarchie
- Suche nach Abhängigkeiten zwischen Objekten
 - Aggregation
 - Komposition

BEISPIEL

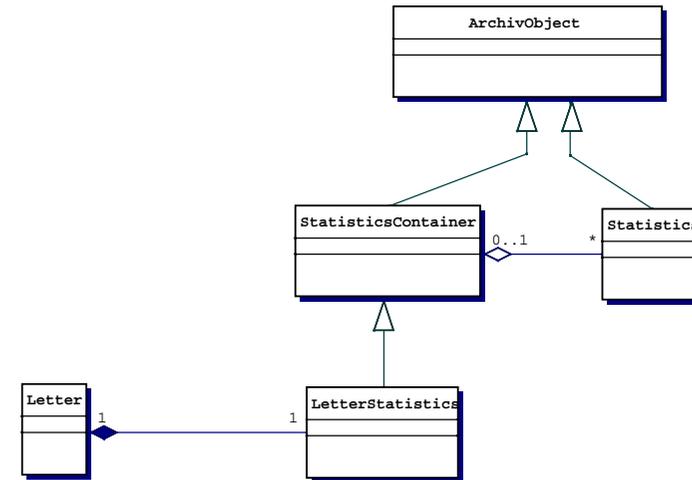
- Letter:
 - Letter ID
 - ZIP Code
 - Output Tray

BEISPIEL

- Letter "has" a:
 - Picture
 - Statistics object

6 OOA — Objekte organisieren (8)

- Briefsortierer — Statistik: UML-Diagramm mit Vererbungsbeziehungen



7 OOA — Beschreibe Interaktionen

- Ausführung von Use Cases



Lege Operationen fest

- die wesentlichen Methoden der Objekte

BEISPIEL

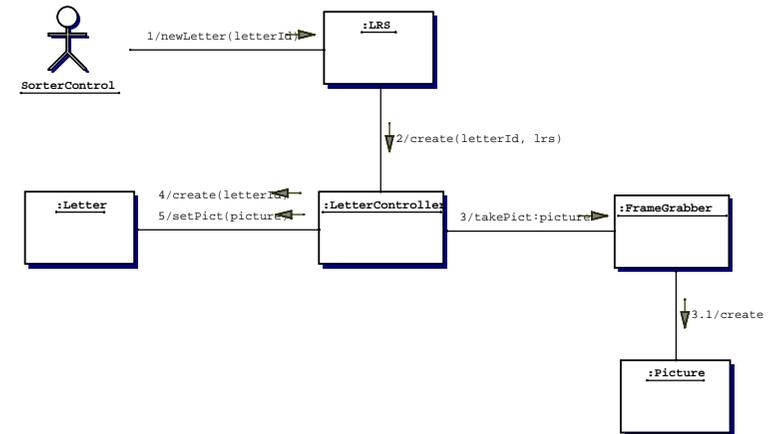
- Brief bearbeiten
- Statistiken abfragen

BEISPIEL

- Sorter Control:
 - Brief in Fach werfen

7 OOA — Describe Interactions (2)

- Briefsortierer — UML Kollaborations-Diagramm



8 OOA — Struktur verfeinern

- Use Cases strukturieren
 - Gemeinsame Abläufe identifizieren
- Objekte detaillierter dokumentieren
 - Attribute
 - Operationen
 - Rollen und Verantwortlichkeiten beschreiben

9 OOA - OOD?

Wo endet die Analyse und wo fängt das Design an?

- Analyse nimmt oft über 50% eines Entwicklungszyklus ein!
- Design beginnt nach 20 - 30%
- Übergänge sind sehr fließend
 - irgendwann können Implementierungsaspekte nicht mehr zurückgestellt werden

C.9 Objektorientiertes Design

- Transformation des Analysemodells zu einem implementierbaren Modell
- Aspekte der Implementierungsumgebung aufnehmen
- Strukturelle und strategische Entscheidungen treffen
 - wo brauche ich eigene Threads
 - Verteilung der Anwendung auf mehrere Rechner
 - welche Interprozesskommunikation wird eingesetzt
 - Datenbankschnittstellen
 - Fehlerbehandlung
 - Garbage collection
- Weitere Verfeinerung der Objektinteraktionen und Schnittstellen

1 Phasen

- Klassen-Entwurf
 - Finde Software-Klassen für die Klassen des Analysemodells
 - Zerteile Analysemodell-Klassen
 - Entferne unnötige Klassen des Analysemodells
 - Füge neue Klassen hinzu (z. B. Listen oder Hashtab. zur Verwaltung)
 - ! **Objektgrenzen müssen erhalten bleiben!**
Analyse → Design → Implementierung (Traceability)
- Systementwurf
 - Nicht-problembezogene Aspekte
(Verteilung, Nebenläufigkeit, Betriebsmittel, Systemschnittstellen, ...)
- Programmentwurf
 - Programmiersprache
 - Fehlerbehandlung (Exceptions, Rückgabewerte)
 - Performance-Aspekte