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.**

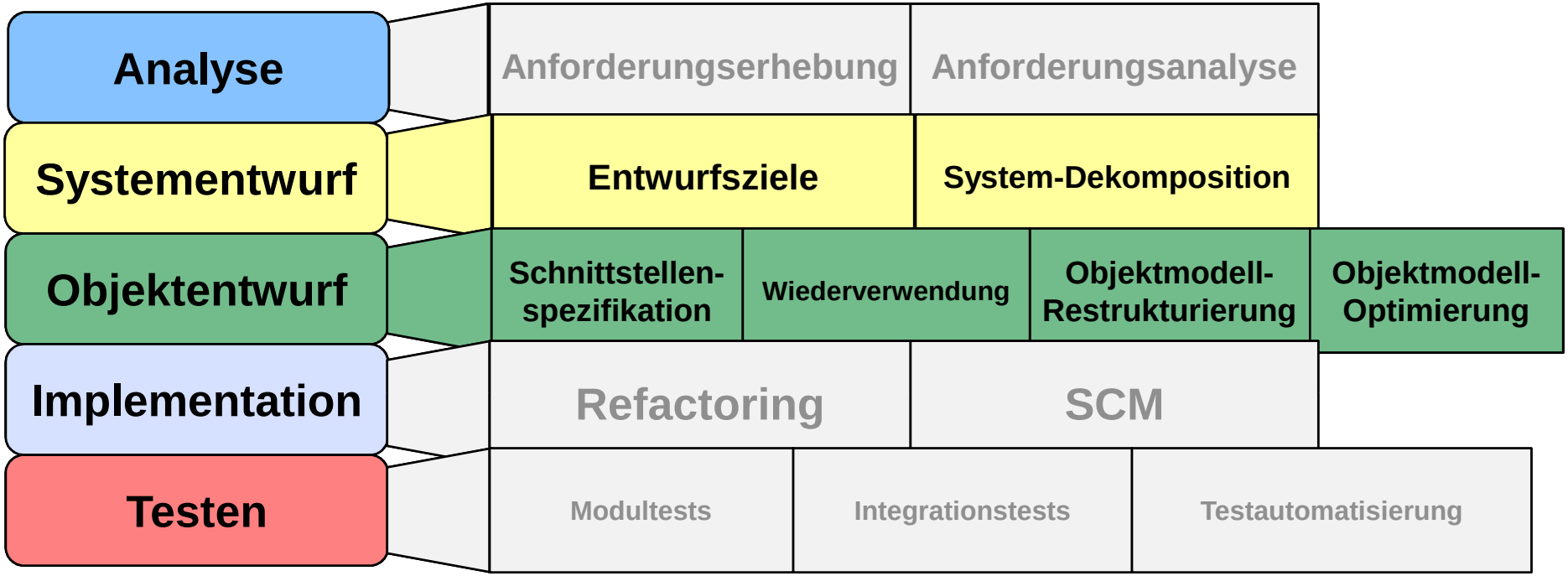
universitätbonn ROOTS

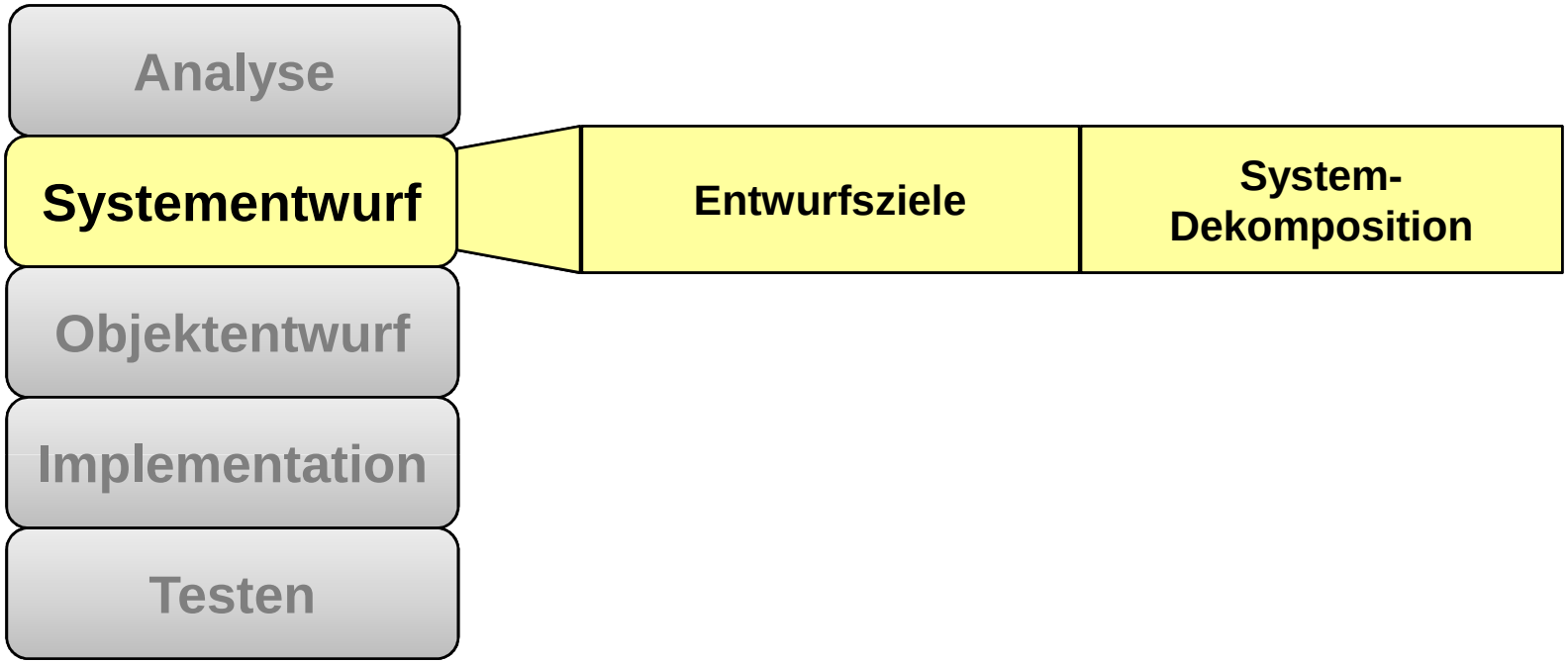
# Systementwurf
# Objektentwurf

von Eva Stöwe und Jan Nonnen

18.03.2010

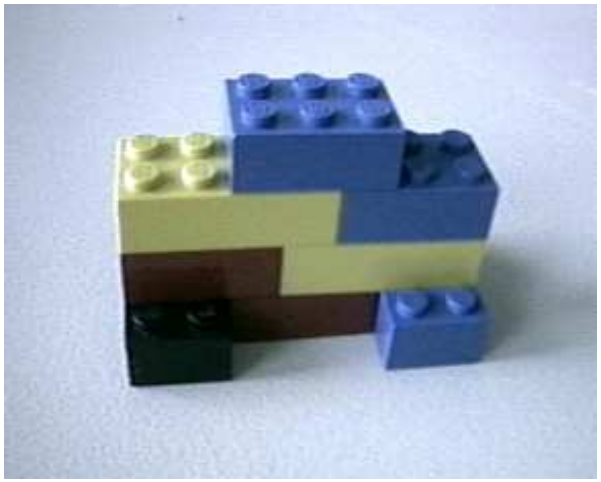
# Entwicklungsphasen Überblick




Analyse
Systementwurf
Objektentwurf
Implementation
Testen
Entwurfsziele
System-
Dekomposition

# Systementwurf ▸ Ziele

- Identifizieren von Entwurfsziele und zerlegen des Systems in Subsysteme mit Hilfe von Diensten
- Verfeinern der Systemzerlegung, bis alle Entwurfsziele erreicht sind
- Ergebnis:
  - **Entwurfsziele**
  - **Subsystemzerlegung**
    - ⇨ Komponentendiagramme
    - ⇨ Spezifikation der Dienste
  - **Systementwurfs-Objektmodel**
    - ⇨ Verteilungsdiagramme



# Systementwurf ▸ Entwurfsziele

- Definieren systemübergreifende Punkte, die von den Entwicklern beachtet werden müssen
- Werden mit den **Nicht-Funktionalen-Anforderungen** hergeleitet
- Beispiele:
  - Hardware:
    - ⇨ Wie sieht die Hardwarekonfiguration des Systems aus?
    - ⇨ Muss es auch auf einem iPhone laufen?
    - ⇨ …
  - Datenverwaltung:
    - ⇨ Welche Daten sollen persistent sein?
    - ⇨ …
  - Zugriffskontrolle
    - ⇨ Wer darf alles auf diese zugreifen und mit welchen Rechten?
    - ⇨ …

# Systementwurf ▸ Subsystemzerlegung

- **Dienst**: Menge von Operationen mit gemeinsamem Zweck
- **Dienstspezifikation**: Vollständig typisierte Menge von Operationen

## 1. Subsystem-Identifikation

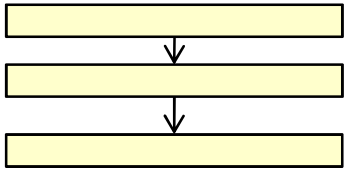
- Definition von Diensten
- Gruppierung von zusammenhängenden Diensten zu einem Subsystem
  - ⇨ Hohe Kohäsion  innerhalb eines Subsystems
  - ⇨ Minimale Kopplung zwischen Subsystemen

## 2. Subsystem-Anordnung

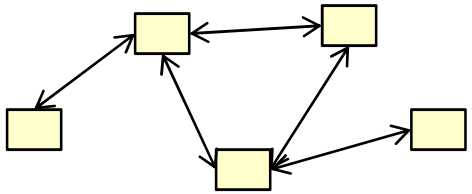
- Wie interagieren die Subsysteme miteinander
- Auswahl der Software Architektur

# Systementwurf ▸ Architekturen

- Schichten-Architektur
  - Client/Server Architektur
  - N-Tier

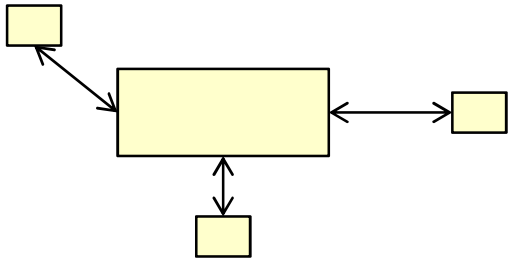


- Peer-to-Peer Architektur

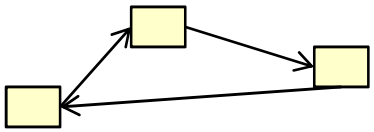


# Systementwurf ▸ Architekturen

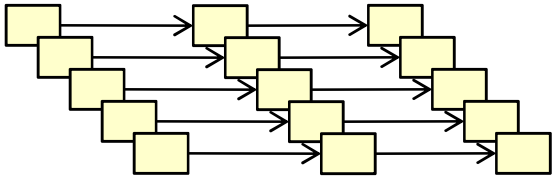
- Repository Architektur



- View/Controller/Model Architektur



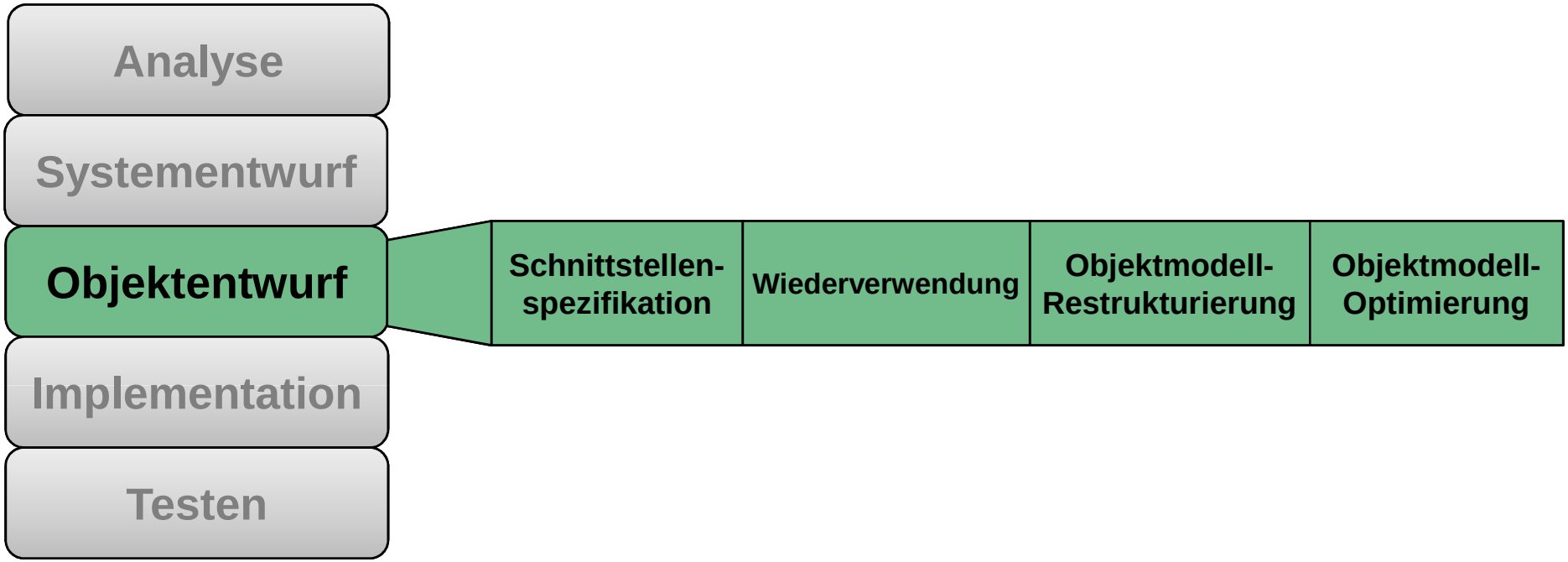
- Pipes & Filter Architektur



# Systementwurf ▸ Grenzfälle

- Die meiste Zeit beschäftigt man sich beim Systementwurf mit dem Verhalten im Betriebszustand.

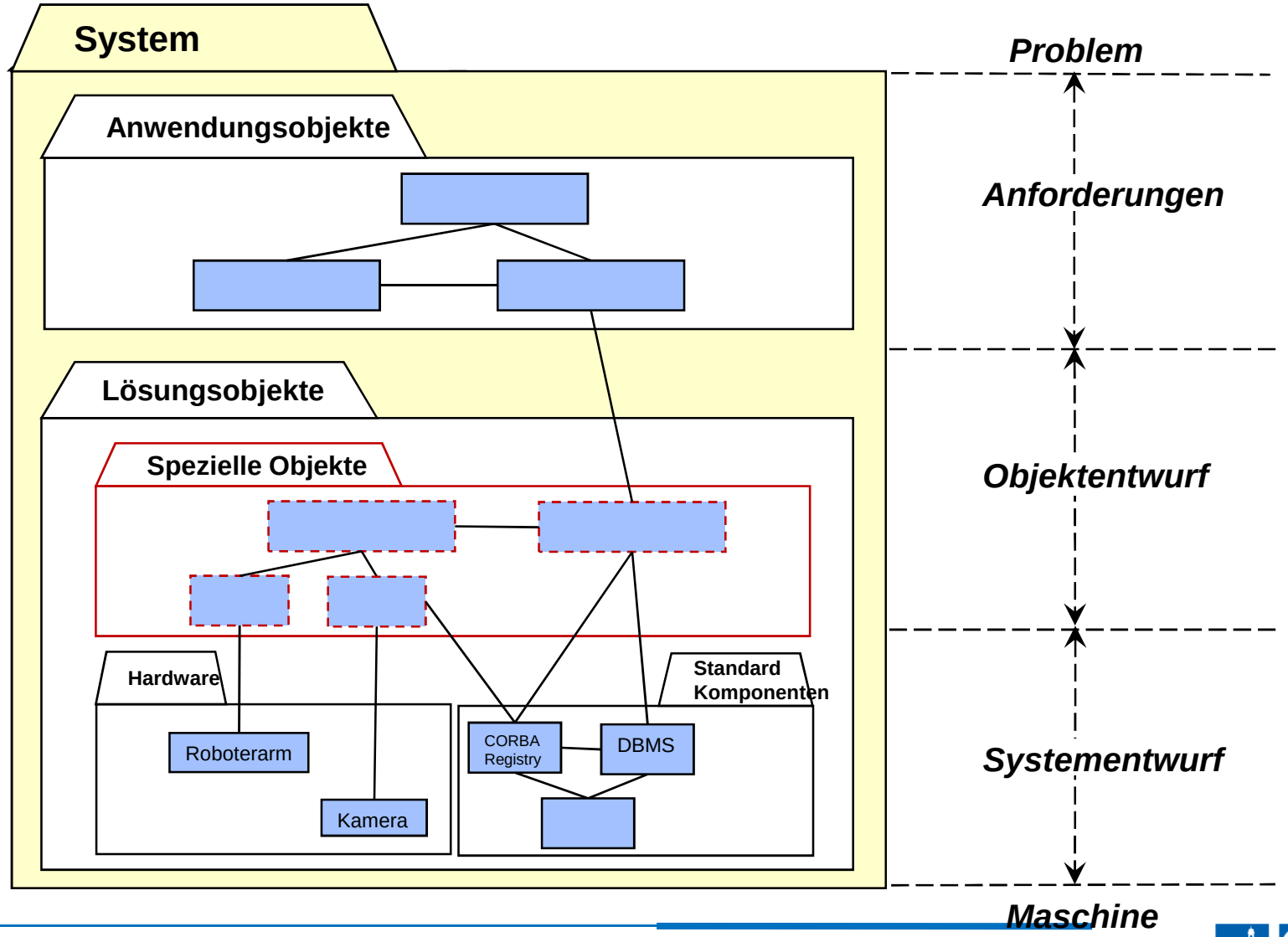
- Abschliessend muss man sich aber auch mit Grenzfällen befassen.
  - Initialisierung
    - ⇨ Beschreibt, wie das System aus einem nicht initialisierten Zustand in einen Betriebszustand gebracht wird ("startup use cases”).
  - Terminierung
    - ⇨ Beschreibt, welche Ressourcen vor der Beendigung aufgeräumt werden und welche Systeme benachrichtigt werden (“Terminierungs-Use Cases").
  - Fehler
    - ⇨ Viele mögliche Gründe: Programmierfehler, externe Probleme (Stromversorgung).
    - ⇨ Guter Systementwurf sieht fatale Fehler voraus (“Fehler-Use Cases”).


Analyse
Systementwurf
Objektentwurf
Implementation
Testen
Schnittstellen-spezifikation
Wiederverwendung
Objektmodell-Restrukturierung
Objektmodell-Optimierung

# Objektentwurf ▸ Ziele

- In der **Analyse** wurde spezifiziert, was das System zu leisten hat
- In dem **Systementwurf** wurde festgelegt, wie die grundlegende Architektur aussieht und die System-Zerlegung
- Jetzt wird das Analyse-Objektmodell erweitert, bis es eine Grundlage für eine konkrete objektorientierte Implementation darstellt
- Basis:
  - **Analyse-Objektmodell, Analyse-Verhaltensmodell**
  - **System-Objektmodell, Entwurfsziele, Subsystemzerlegung**
- Erstellt:
  - **Objektmodell**
    - ⇨ Wichtigster Grundstein für den weiteren Entwurf

# Vom Objektentwurf zur Implementierung



# Objektentwurf ▸ Schritte

1. Schnittstellenspezifikation
   - Spezifikation der Schnittstellen und des Verhaltens aller Typen
2. Nutzung von existierenden Standardkomponenten
   - Bestimmung von Algorithmen-Bibliotheken
3. Restrukturierung des Objektmodells
   - Umformen des Objektmodells zur Verbesserung von Verständlichkeit und Erweiterbarkeit
     - ⇨ z.B. Anpassung der Vererbung aus Verhaltenssicht
   - Realisierung von UML-Konzepten, die keine Entsprechung der gewählten Programmiersprache haben
     - ⇨ z.B. Realisierung von Assoziationen und Multiple-Vererbung
4. Optimierung des Objektmodells
   - Umformen des Objektmodells unter Berücksichtigung von Effizienzfragen

universitätbonn ROOTS

# Schnittstellenspezifikation

# Objektentwurf ▸ Schnittstellen

- Im **Systementwurf** werden Typsignaturen für **Dienste** festgelegt.
- Im **Objektentwurf** werden Typsignaturen für alle **Typen** festgelegt.

- Spezifikation der Signaturen und Sichtbarkeiten
- Identifizieren von fehlenden Attributen und Operationen

- Verfeinerung:
  - Verhaltens-Spezifikation mit **Design-by-Contract**
  - Interaktions-Spezifikation durch **Behaviour-Protocols**
  - Spezifikation der jeweils benötigten vorhandenen Standard-Komponenten und Adaptions von Schnittstellen

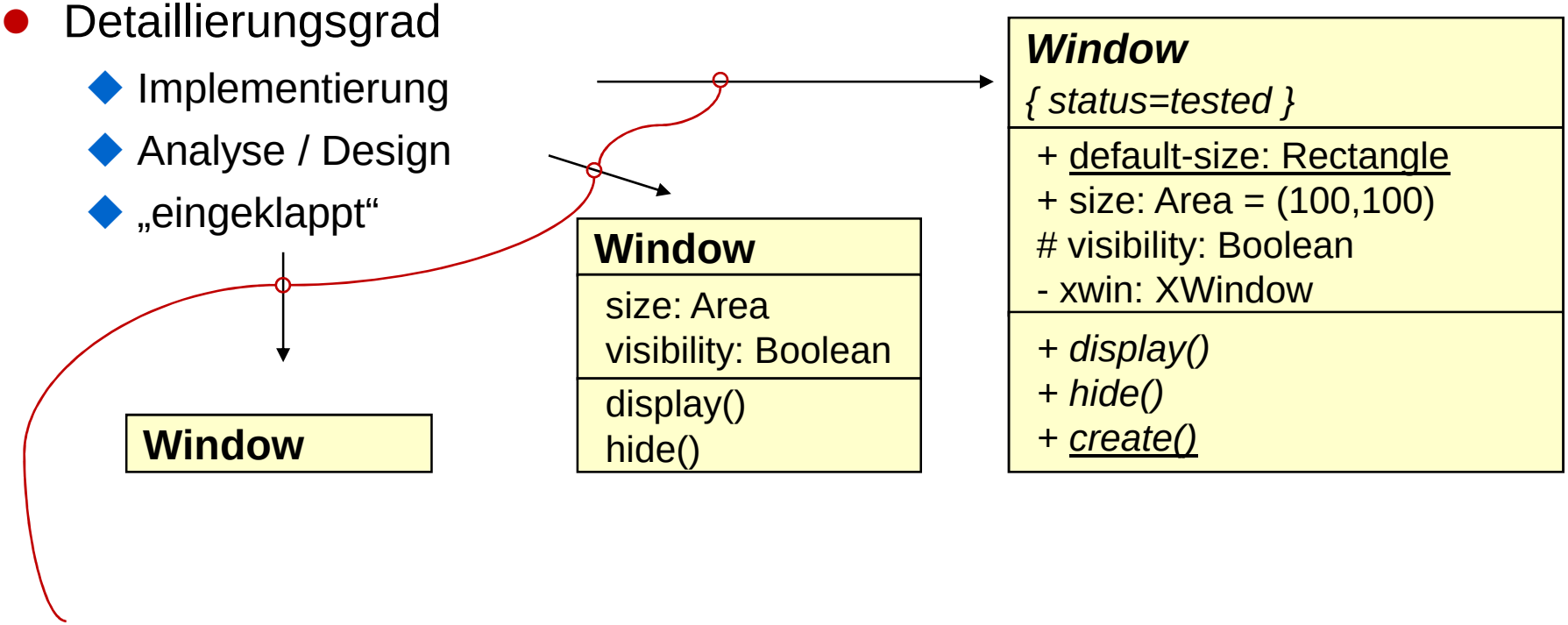
# Typen ▸ Fehlende Attribute/Operationen

- Alle Subsystemdienste und Analyseobjekte untersuchen
  - Ergänzung der  zur Realisierung benötigten
    - ⇨ Attribute
    - ⇨ Operationen
- Iteration über das Modell während der gesamten Phase
- Erfordert
  - hohen Informationsaustausch zwischen Entwicklern
  - gute Dokumentation von Designentscheidungen
- **Wichtig:**

  Konsistenz von neuen Entscheidungen und Entwurfszielen mit alten

# Objektentwurf ▸ Detailstufen einer Klasse

- Detaillierungsgrad
  - Implementierung
  - Analyse / Design
  - „eingeklappt“



universitätbonn ROOTS

# Design by Contract

# DbC ▸ Übersicht

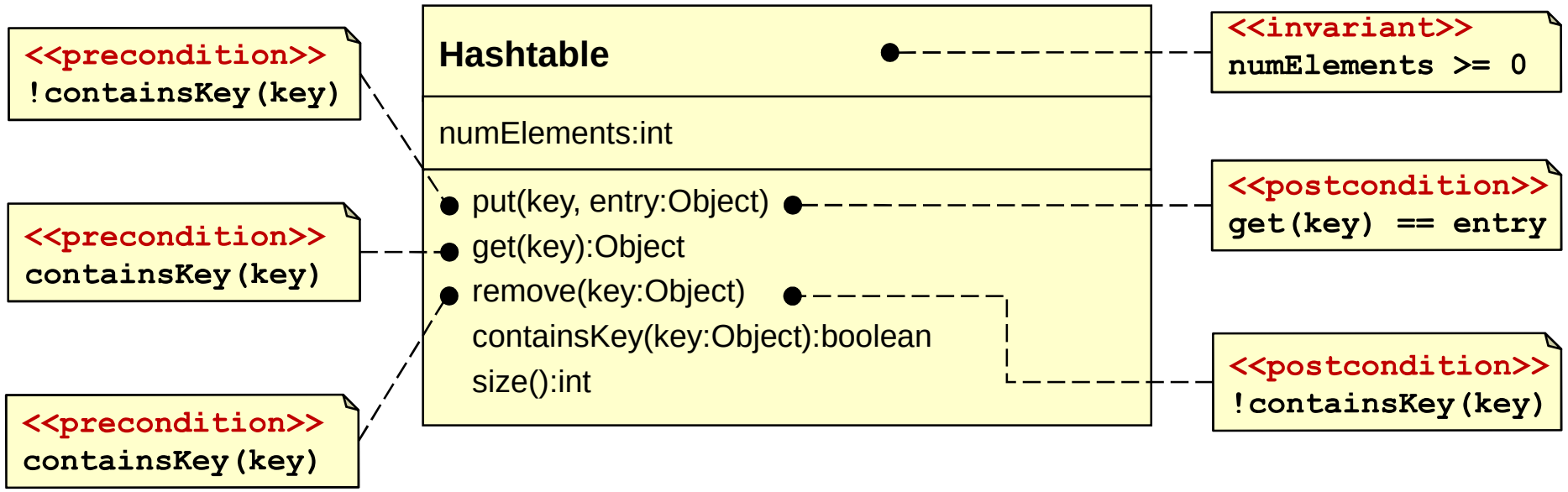
- Vertrag (Contract)
  - Menge aller Assertions, die festlegen, wie zwei Partner interagieren
- Contract besteht aus
  - Precondition
  - Postcondition
  - Class invariant
- Überprüfung per Assertion
  - Logische Aussage, die wahr sein muss
  - Macht die Annahmen explizit, unter denen ein Design funktioniert

# DbC ▸ Übersicht

- Precondition
  - Ist eine Voraussetzungen dafür, dass eine Operation korrekt ausgeführt werden kann
- Class Invariant
  - Beschreibt deklarativ legale Zustände von Instanzen einer Klasse
- Postcondition
  - Beschreibt deklarativ das Ergebnis eines korrekten Aufrufs

# DbC ▸ Formulierung in UML

- Jede Assertion kann auch als Notiz dargestellt und an des jeweilige UML Element “angehängt” werden.
  - Die Art der Assertion wird als Stereotyp angegeben



# DbC ▸ Assertions in Java

- Anwendung
  - Man kann Assertions an beliebigen Stellen des Quellcodes verwenden um logische Ausdrücke zu überprüfen.
    - ⇨ Liefert ein solcher Ausdruck `false` zurück, wird ein `AssertionError` ausgelöst und die Ausführung des Programms stoppt.
    - ⇨ Die ausgelösten Fehler sind vom Typ `Error` und nicht vom Typ `Exception`

- Syntax

```
assert <boolean expression>;
assert (c != null);
```

**oder:**

```
assert <boolean expression> : <String>;
assert (c != null) : "Customer is null“;
```

universitätbonn ROOTS

# Nutzung von existierenden Standardkomponenten

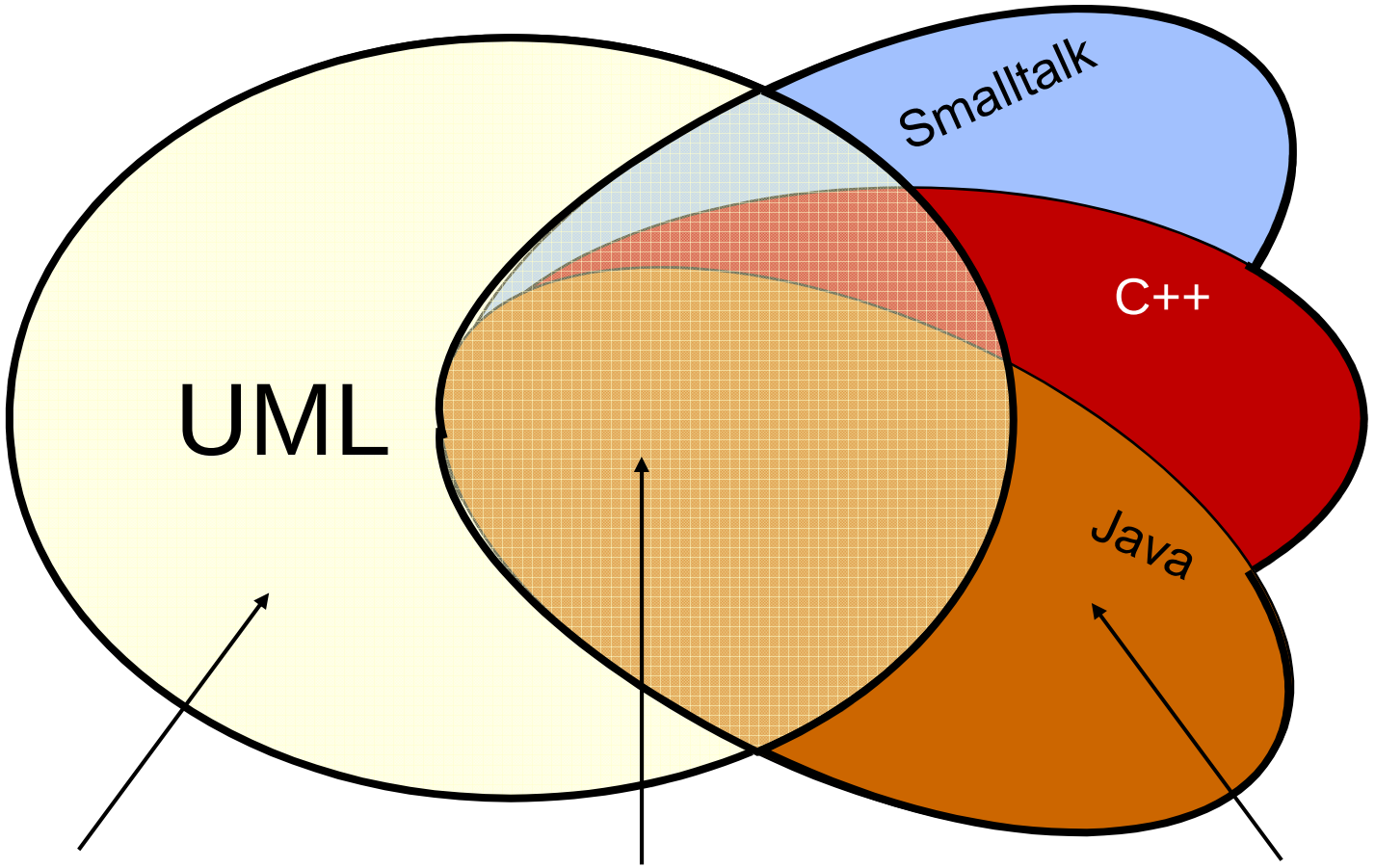
# Objektentwurf ▸ Standardkomponenten

- Komponenten-Suchmaschinen
  - Bsp: Google Code, SourceForge, GitHub
- Auswahl existierender Standardklassenbibliotheken, Frameworks oder Komponenten
- Anpassung der Standardklassenbibliotheken, Frameworks oder Komponenten
  - Nutzung vorgesehener Anpassungsmöglichkeiten
    - ⇨ Parametrisierung
    - ⇨ Bildung von Unterklassen
  - Unvorhergesehene Anpassung
    - ⇨ Änderungen der API, falls der Quellcode verfügbar ist
    - ⇨ Sonst: Adapter oder Bridge Pattern
    - ⇨ Sonst: Aspektorientierte Programmierung

universitätbonn ROOTS

# Erweiterung und Restrukturierung des Objektmodells

# Objektentwurf ▸ Kern-UML




Konzepte ohne direkter Entsprechung in gängigen Sprachen

direkt ineinander abbildbarer gemeinsamer „Kern“

sprachspezifische Details

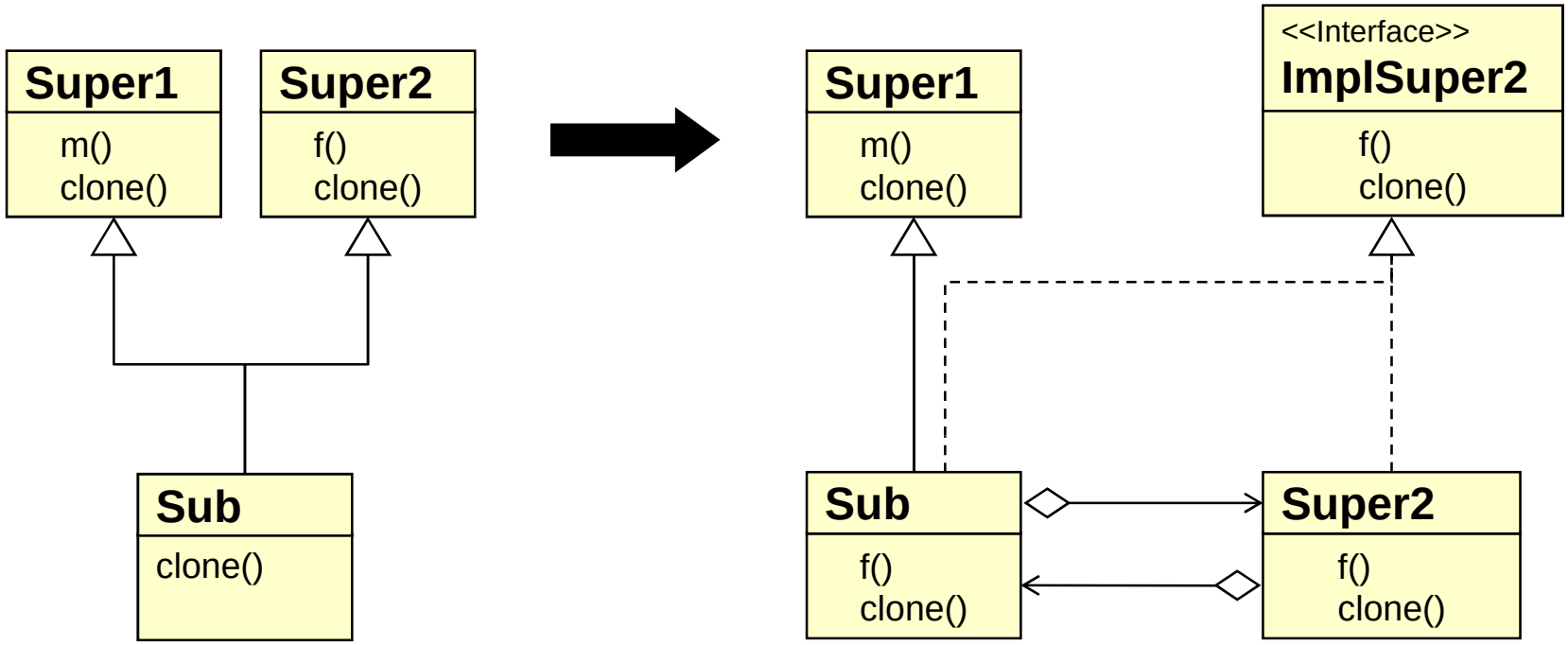
# UML Konzepte ▸ Umsetzungsbeispiele

- Multiple Vererbung
  - Wiederverwendung: durch Aggregation und Forwarding
  - Subtyping: durch Implementation eines gemeinsamen Interfaces
  - Overriding: durch „Rückreferenzen“ (als Parameter oder Feld)
- Multiple Instantiierung / Multiple Sichten
  - Decorator (evtl. plus obige Simulation von Subtyping und Overriding)
- Dynamische Klassifikation / Dynamische Änderung der Klassenzugehörigkeit
  - Strategy
- Jetzt Fokus auf Multiple Vererbung

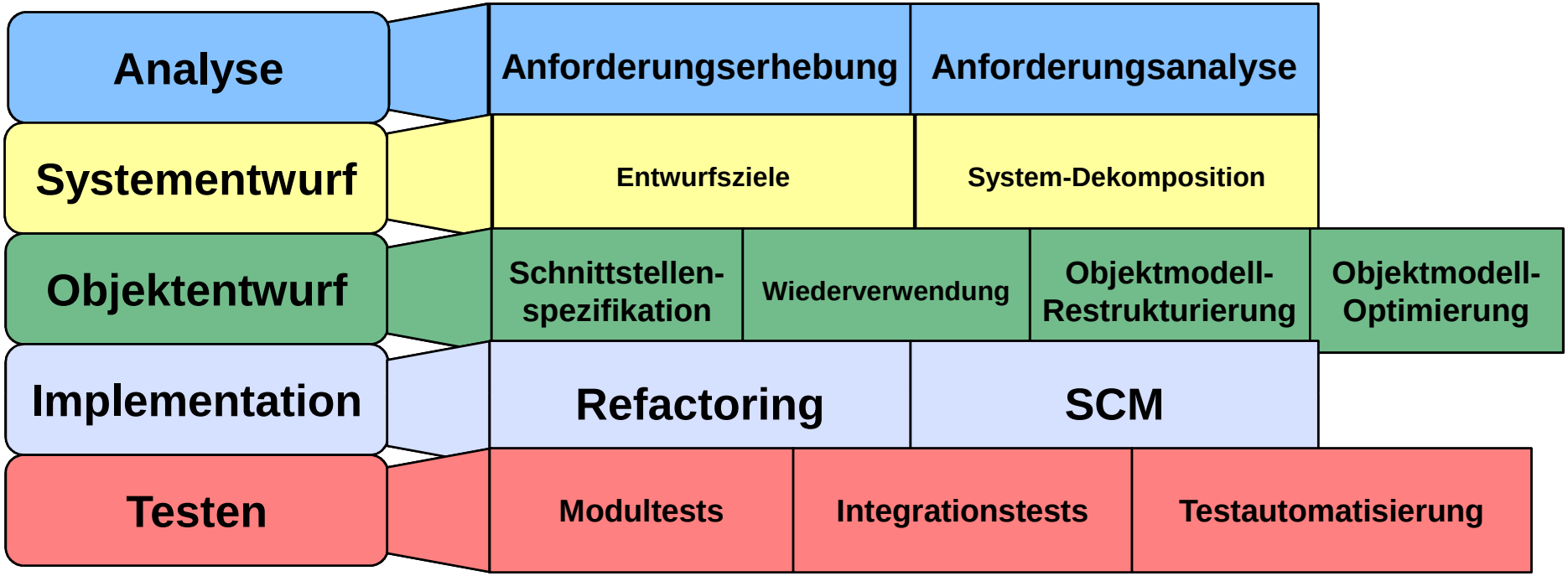
# UML Konzepte ▸ Multiple Vererbung in Java

- Absicht
  - Interface und Code mehrerer "Oberklassen" wiederverwenden
  - ... obwohl Java nur einfache Vererbung erlaubt
- Motivation
  - komplexe Klassen nicht neu implementieren
- Anwendbarkeit
  - keine semantischen Konflikte zwischen "Oberklassen"-Methoden

# UML Konzepte ▸ Multiple Vererbung in Java



# Entwicklungsphasen Überblick



# Entwicklungsphasen und ihre Modelle