Moduldokumentation

Modul Software Architecture (swa)

Simon Wächter

2017

Inhalt

[1 Einleitung 3](#_Toc476763111)

[1.1 Einleitung 3](#_Toc476763112)

[1.2 Lernziele 3](#_Toc476763113)

[1.3 Prüfungen 3](#_Toc476763114)

[2 Woche 1 4](#_Toc476763115)

[2.1 Einführung Software-Architektur 4](#_Toc476763116)

[2.1.1 Motivation 4](#_Toc476763117)

[2.1.2 Lernziele 4](#_Toc476763118)

[2.1.3 Drei Aussagen zur Software-Architektur 4](#_Toc476763119)

[2.1.4 Aufgabe: Was ist die Essenz jedes dieser Texte 5](#_Toc476763120)

[2.1.5 Aufgabe: Ihre eigene Erfahrung mit Software-Architektur 6](#_Toc476763121)

[2.1.6 Warum Software-Architektur 6](#_Toc476763122)

[2.1.7 Software-Architektur als Interacting Components 7](#_Toc476763123)

[2.1.8 Architectural Views 7](#_Toc476763124)

[2.1.9 Beispiel: Logical View versus Development View einer einfachen Linked List 8](#_Toc476763125)

[2.2 Ergänzung 9](#_Toc476763126)

[3 Woche 2 10](#_Toc476763127)

[3.1 Motivation des objektorientierten Entwurfes – Logische Sicht 10](#_Toc476763128)

[3.2 Lernziele 10](#_Toc476763129)

[3.3 Software: Vom Programmierer zur Wirkung 10](#_Toc476763130)

[3.4 Prinzipien und Methoden zum Entwurf von Strukturen in der Logical View 11](#_Toc476763131)

[3.4.1 Entwurfsprinzip Low Representational Gap (LRG) 11](#_Toc476763132)

[3.4.2 Entwurfsmethode Domain-Driven Design (DDD) 11](#_Toc476763133)

[3.4.3 Beispiel einer Domain-Driven Design Adressenverwaltung 12](#_Toc476763134)

[3.5 Abläufe und Operationen in der Logical View 15](#_Toc476763135)

[4 Woche 3 17](#_Toc476763136)

[4.1 Motivation 17](#_Toc476763137)

[4.2 Lernziele 17](#_Toc476763138)

[4.3 Vorgehensmodell 17](#_Toc476763139)

[4.3.1 Objektorientierter Entwurf: Logical View 17](#_Toc476763140)

[4.3.2 Objektorientierter Entwurf: Development View 17](#_Toc476763141)

[4.3.3 Methode zur Umsetzung eines objektorientierten Entwurfs 18](#_Toc476763142)

[4.4 Beispiel: Entwurf zum Beispiel der Restaurantsimulation 18](#_Toc476763143)

[4.4.1 Bisherige Restaurantsimulation 18](#_Toc476763144)

[4.4.2 Umsetzung: Liste der zu programmierenden Klassen 19](#_Toc476763145)

[4.4.3 Umsetzung: Kleine Erweiterung der Aufgabe 19](#_Toc476763146)

[4.4.4 Umsetzung: Schnittstellen der zu programmierenden Klassen 20](#_Toc476763147)

[5 Woche 4 21](#_Toc476763148)

# Einleitung

## Einleitung

Dieses Dokument stellt die Moduldokumentation für das Modul swa dar. Allfällige Unterlagen sind im Modulordner zu finden.

## Lernziele

Das Modul beinhaltet folgende Lernziele:

* Sie kennen wichtige Architektur-Sichten auf die Struktur von Software-Systemen
* Sie können solche Sichten für vorhandene oder geplante Systeme darstellen und gefällte bzw. zu fällende Entwurfs-Entscheidungen darin aufzeigen
* Sie erkennen Vor-und Nachteile gegebener Software-Strukturen sowie einzelner Entwurfs-Entscheidungen
* Sie können Entwurfs-Entscheidungen begründet fällen und daraus eine System-Architektur entwickeln
* Sie bewerten Software-Strukturen anhand konkreter Anforderungen und schlagen ggf. Verbesserungen vor.
* Sie können Software-Strukturen auf abstrakterem Niveau als Programm-Code darstellen.
* Sie können solche Software-Strukturen bewusst und begründet entwerfen.
* Sie können solche Software-Strukturen analysieren und ggf. kritisieren und verbessern.
* Sie können solche Software-Strukturen in Programm-Code umsetzen, so dass die abstrakten Strukturen erhalten und nutzbar bleiben.

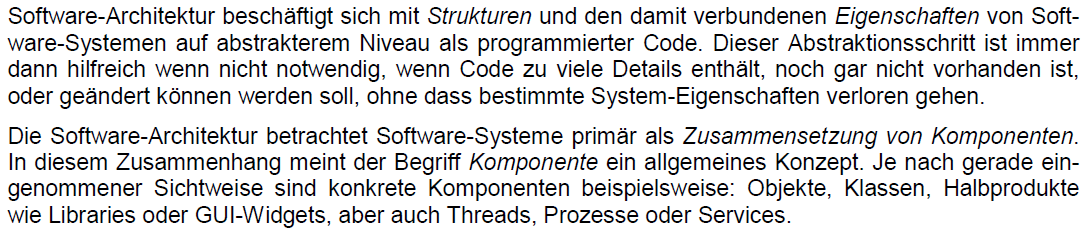
## Prüfungen

Die Modulnote setzt sich aus einer Semesterprüfung zu 50% und einer Modulschlussprüfung zu 50% zusammen.

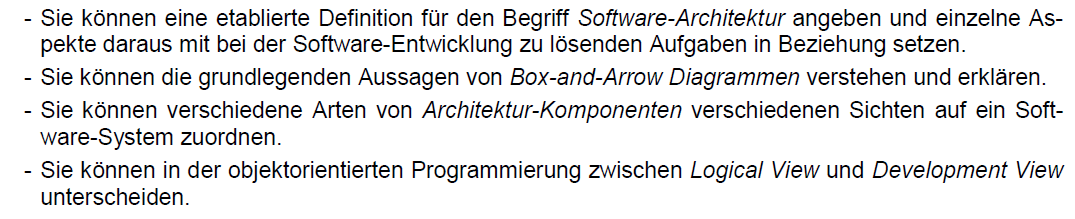
# Woche 1

## Einführung Software-Architektur

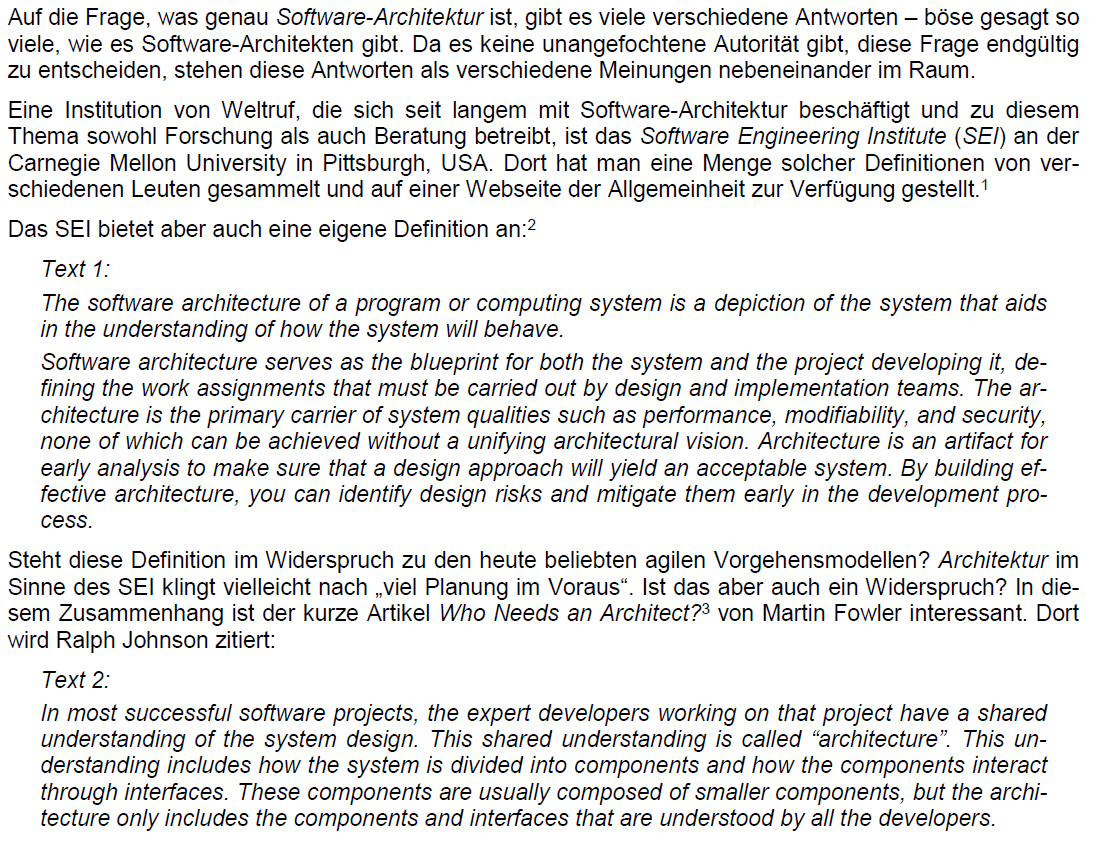
### Motivation

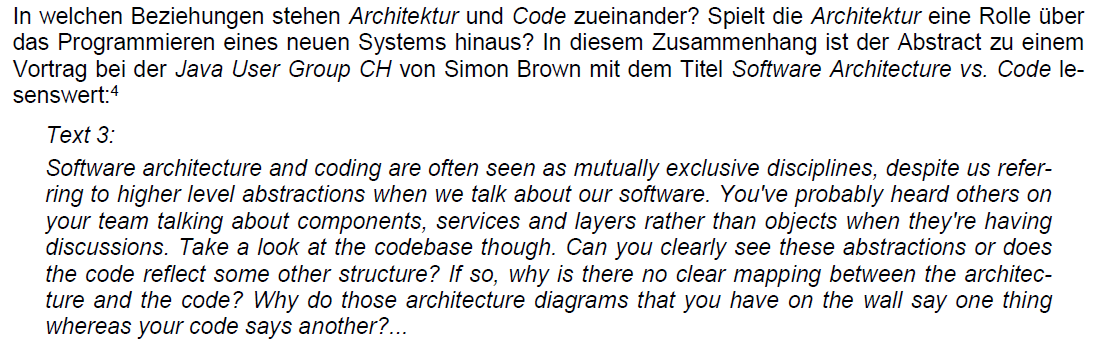


### Lernziele

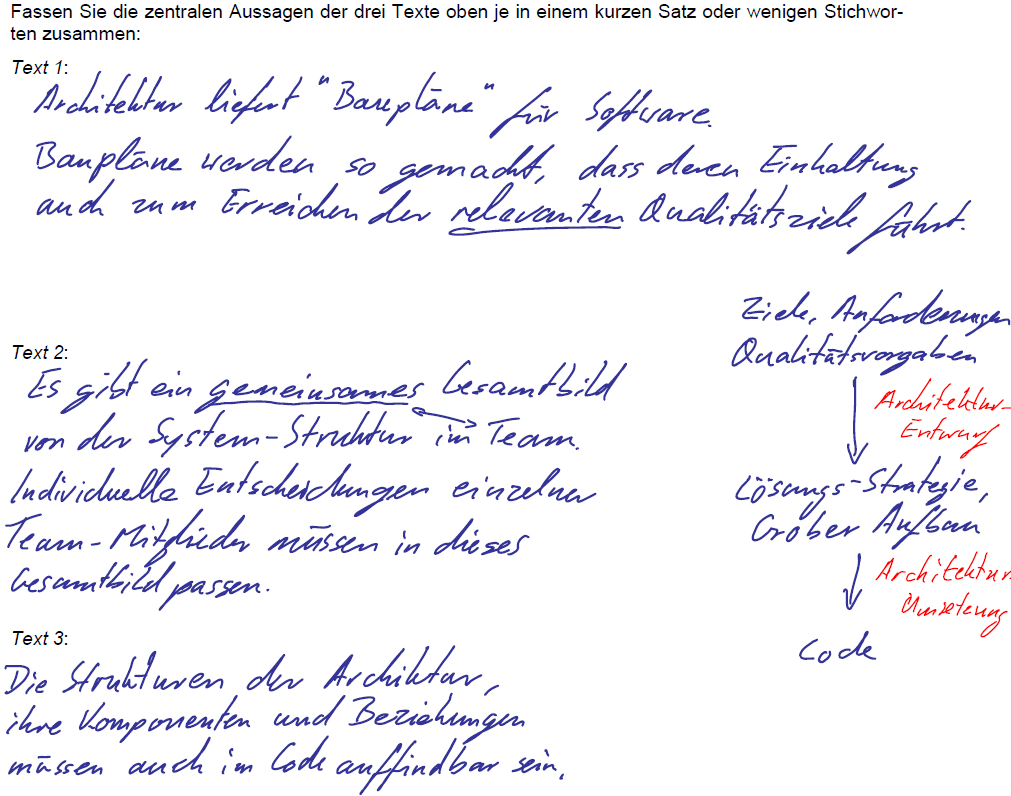


### Drei Aussagen zur Software-Architektur

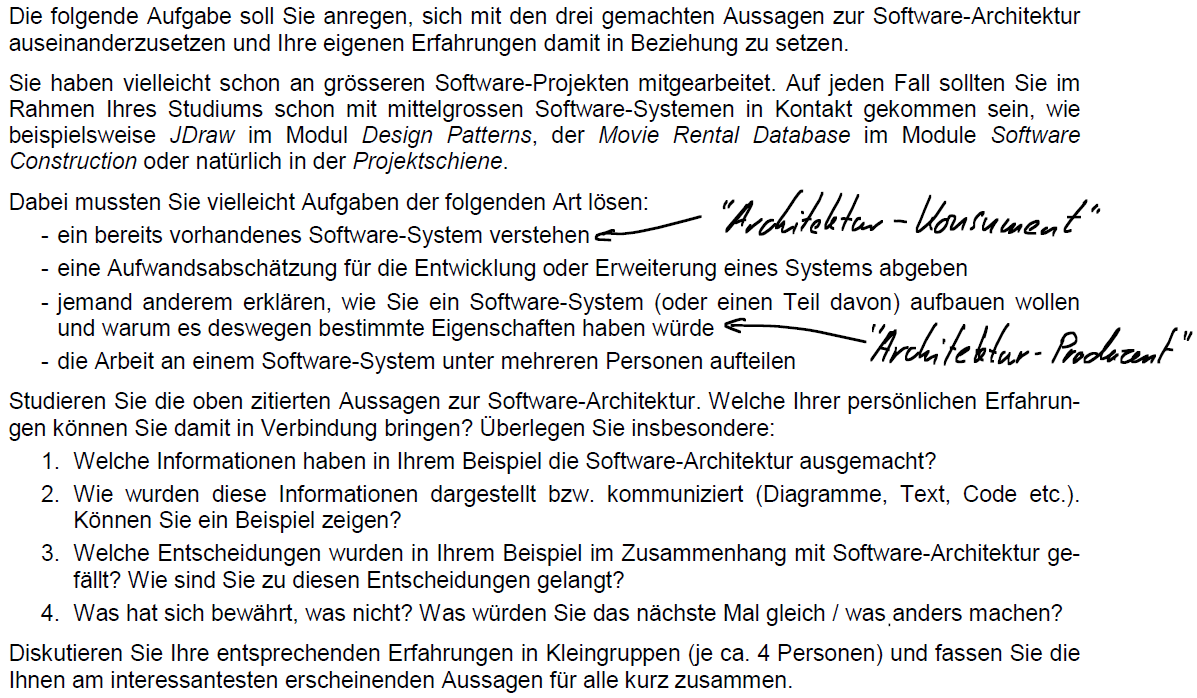




### Aufgabe: Was ist die Essenz jedes dieser Texte



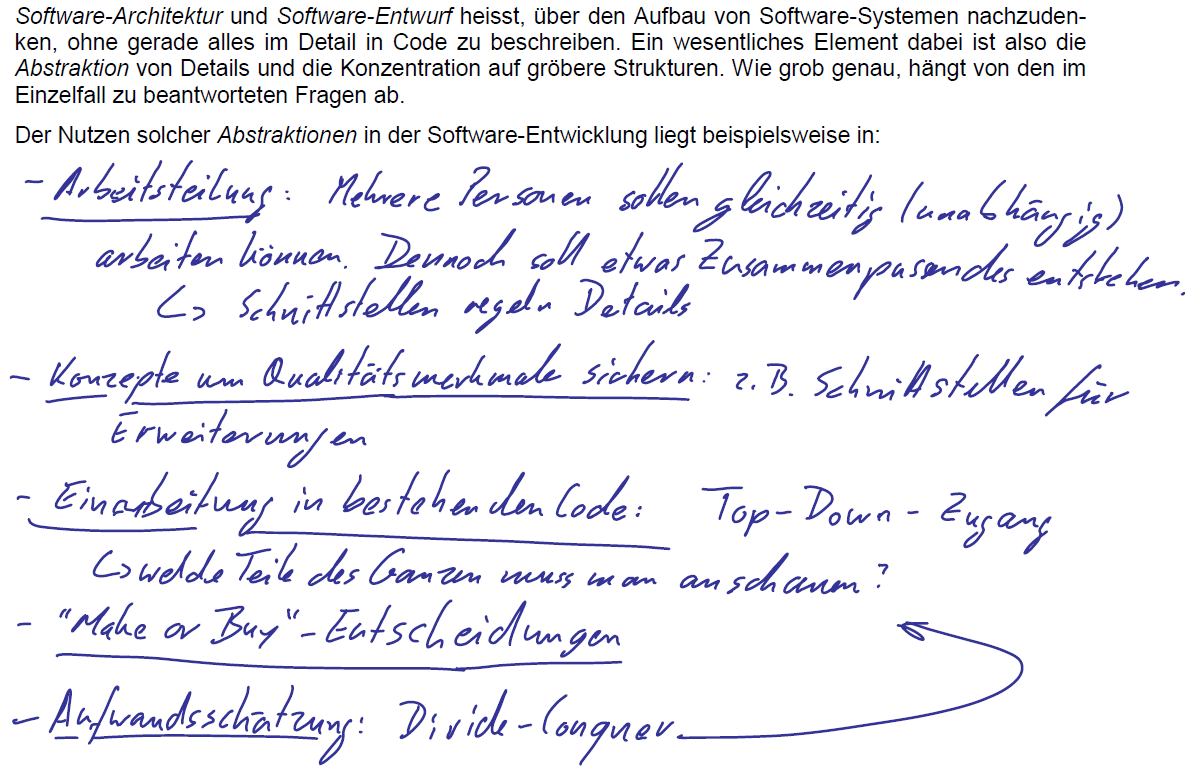
### Aufgabe: Ihre eigene Erfahrung mit Software-Architektur



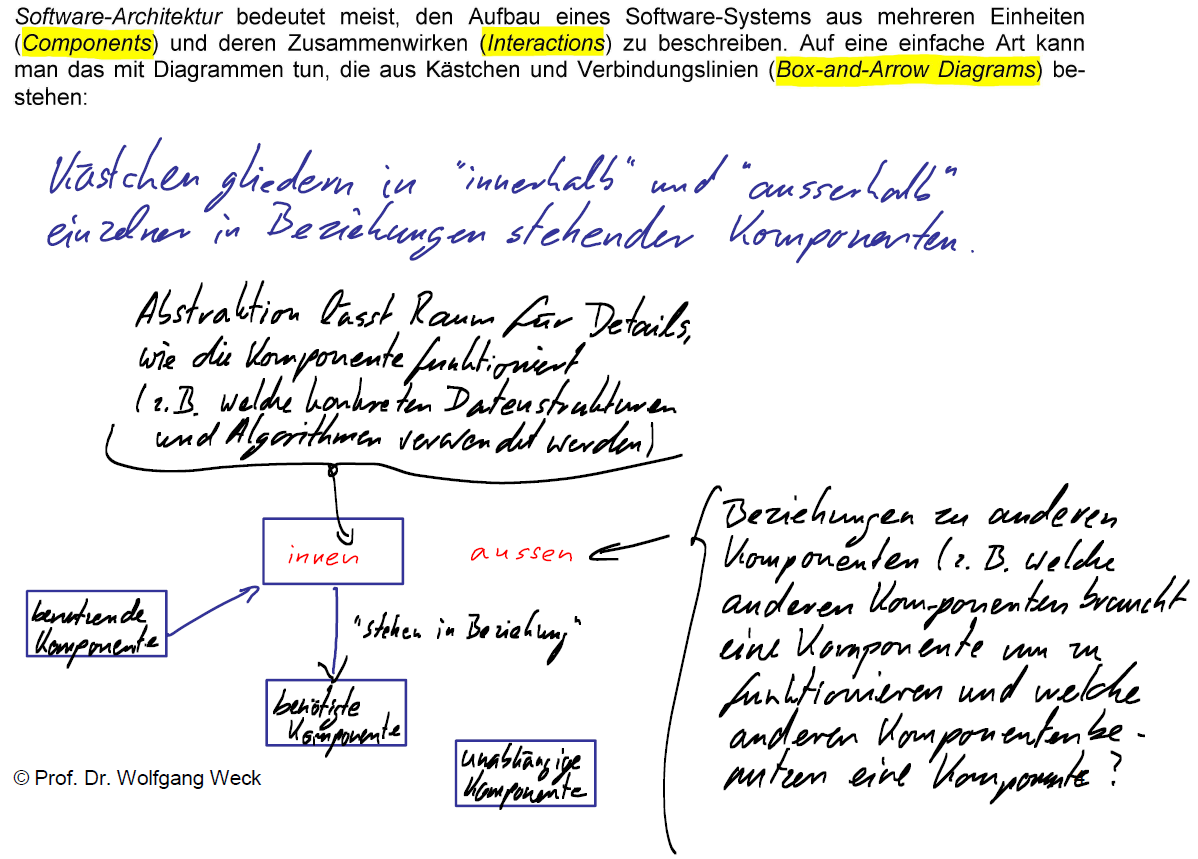
Meine Antworten:

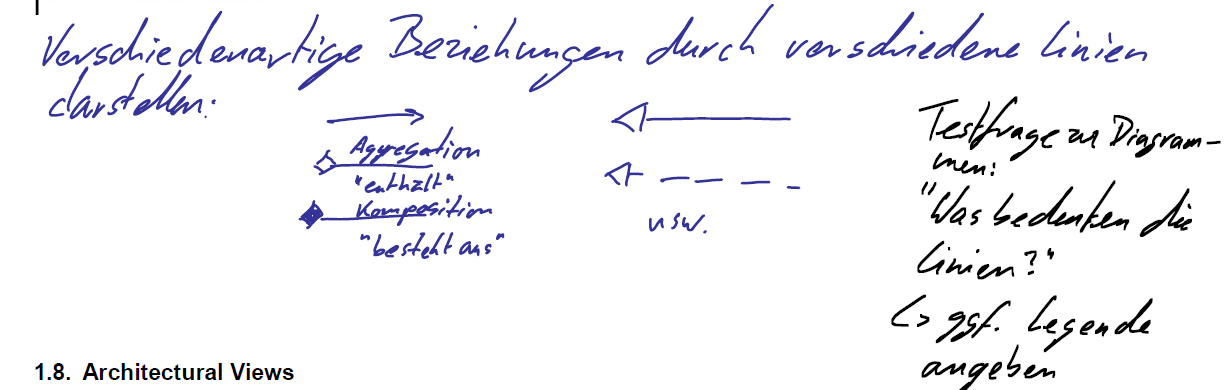
* Ein Bild (Diagramm) sagt mehr als tausend Worte
* Diskussionen kommen der Architektur zu Gute und erhöhen die Chance, die Qualitätskriterien zu erreichen

### Warum Software-Architektur

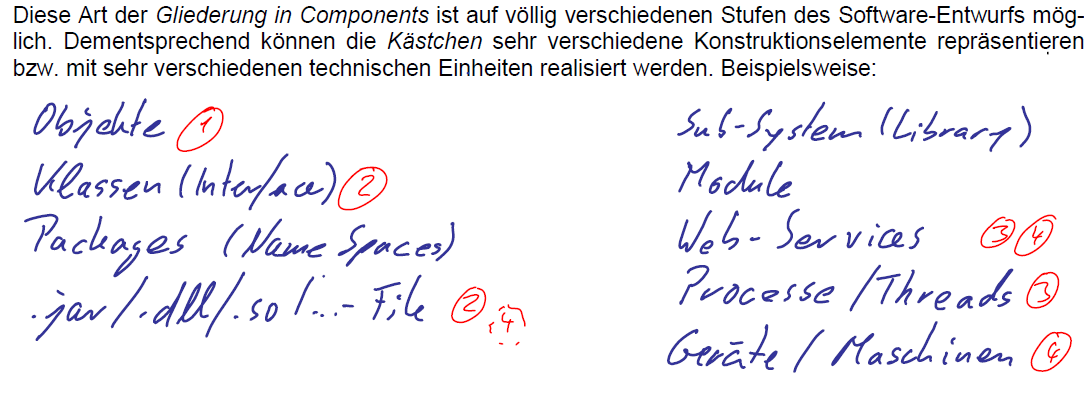


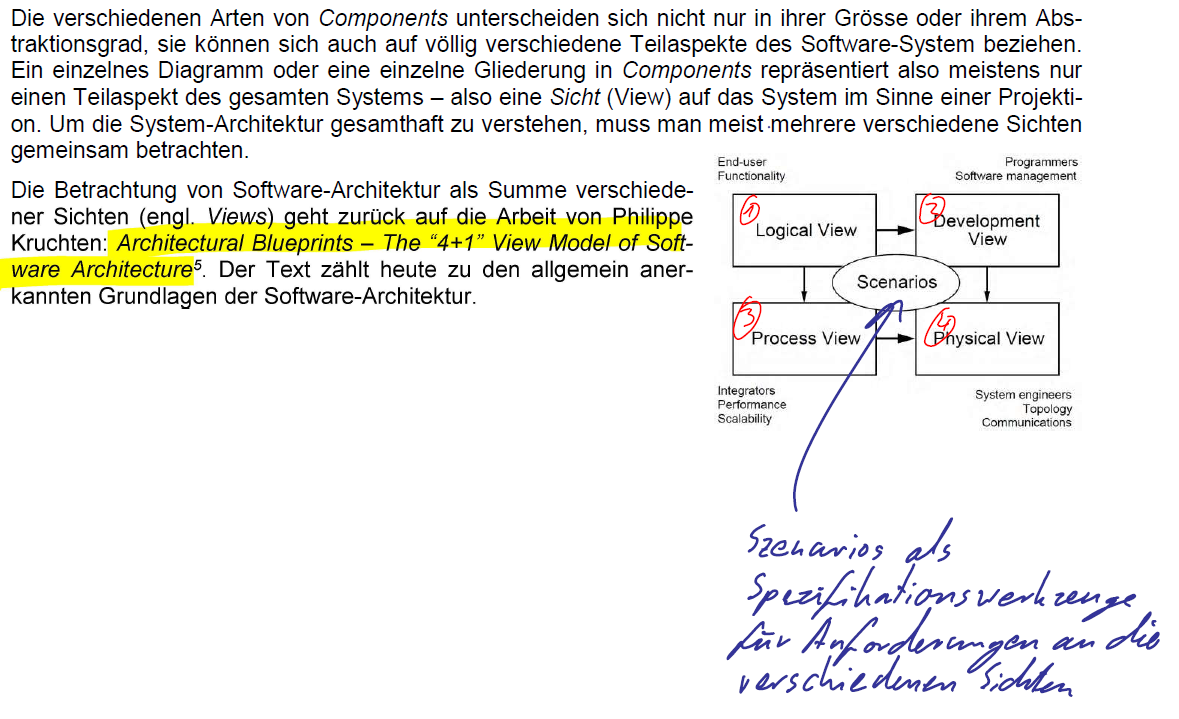
### Software-Architektur als Interacting Components



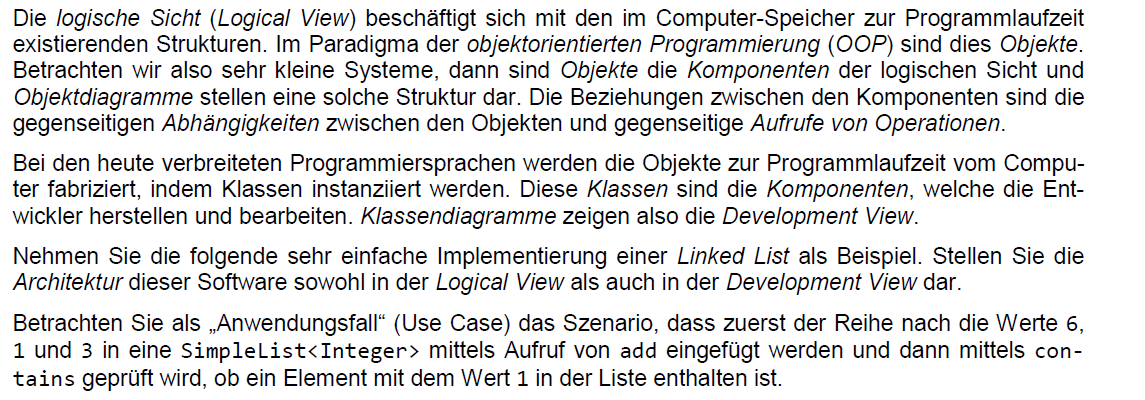


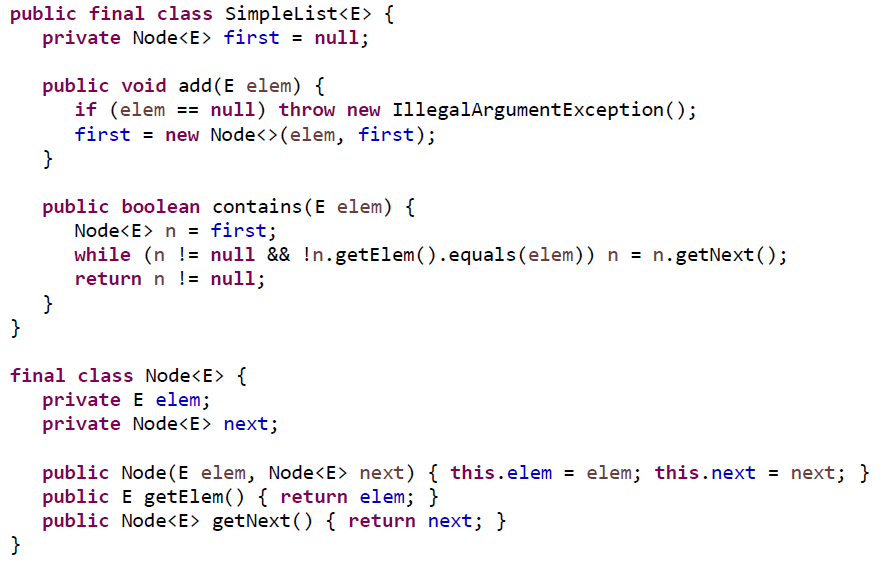
### Architectural Views





### Beispiel: Logical View versus Development View einer einfachen Linked List





## Ergänzung

Gemäss Wikipedia kann das 4+1 View Model der Software-Architektur wie folgt beschrieben werden:

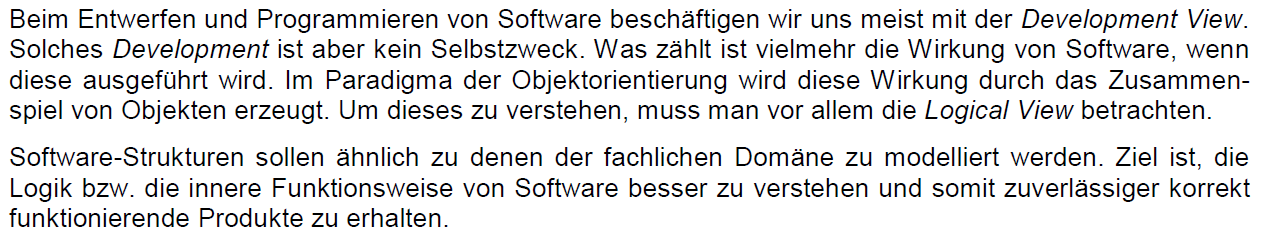
1. Logical View: Enthält die grundlegenden Informationen zum System aus Endbenutzersicht. Dabei werden Objektdiagramme, State-Machines und Sequenzdiagramme verwendet. Die Diagramme dienen einzig dem Entwickler.
2. Development View: Enthält die grundlegenden Informationen zur Projektstruktur mit Modulen und Systemen. Dabei wird auf Klassen-, Package- und Komponentendiagramme zurückgegriffen. Die Diagramme deinen einzig dem Entwickler.
3. Process View: Enthält die grundlegenden Informationen zu Prozessen im System und beschreibt auch Eigenschaften wie Performance und Verfügbarkeit. Dabei werden Aktivitätsdiagramme verwendet.
4. Physical View: Enthält die grundlegenden Informationen zum physischen Deployment des Systems, zum Beispiel wie viele Nodes verwendet werden oder in welcher Form das System deployt wird. Die Diagramme erhalten auch nicht funktionale Anforderungen wie Skalierbarkeit und Verfügbarkeit. Dabei werden Deploymentdiagramme verwendet.
5. Use Case View: Enthält die grundlegenden Informationen zum Erscheinungsbild des Systems von aussen (Es beschreibt die Funktionalität aus Blackbox-Perspektive). Dabei werden Use Case Diagramme verwendet.

Hinzu kommt, was Architektur und was Design ist:

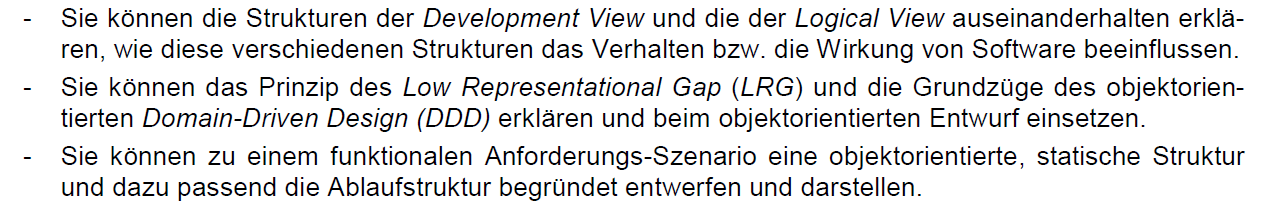
1. Architektur ist immer Design, aber nicht alles Design muss auch Architektur sein 🡪 Nach dem Motto, dass alles designt, aber nicht immer architekturkonform implementiert werden muss (was schlecht ist)
2. Die Architektur beschreibt was und wo etwas getan wird, das Design hingegen wie
3. Kann etwas in der Architektur weggelassen werden, gehört es überspitzt gesagt nicht zur Architektur (z.B. wie wird eine Liste intern implementiert), sondern kann den Designern überlassen werden (Sofern es die im Rahmen des 4+1 Modelles modellierten Anforderungen erfüllt, z.B. die Liste muss genug schnell sein) 🡪 Trägt eine Komponente nichts zum Overall Goal der Architektur bei, ist es eine sogenannte «Nonarchitectural Element»

# Woche 2

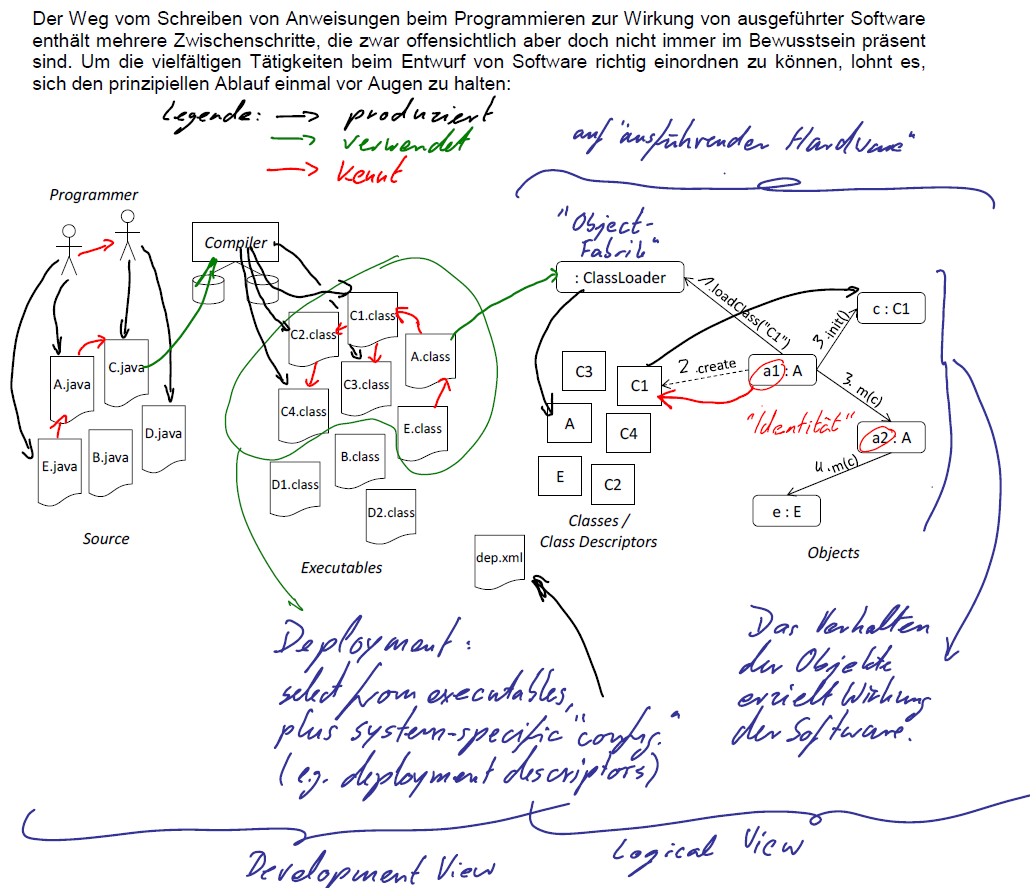
## Motivation des objektorientierten Entwurfes – Logische Sicht

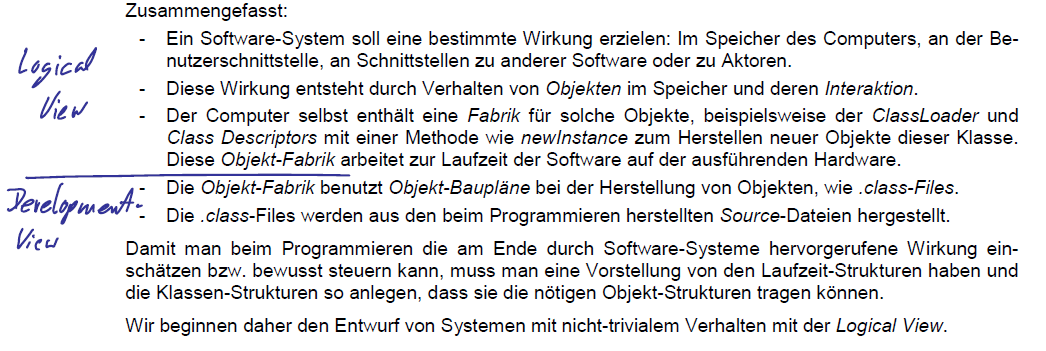


## Lernziele



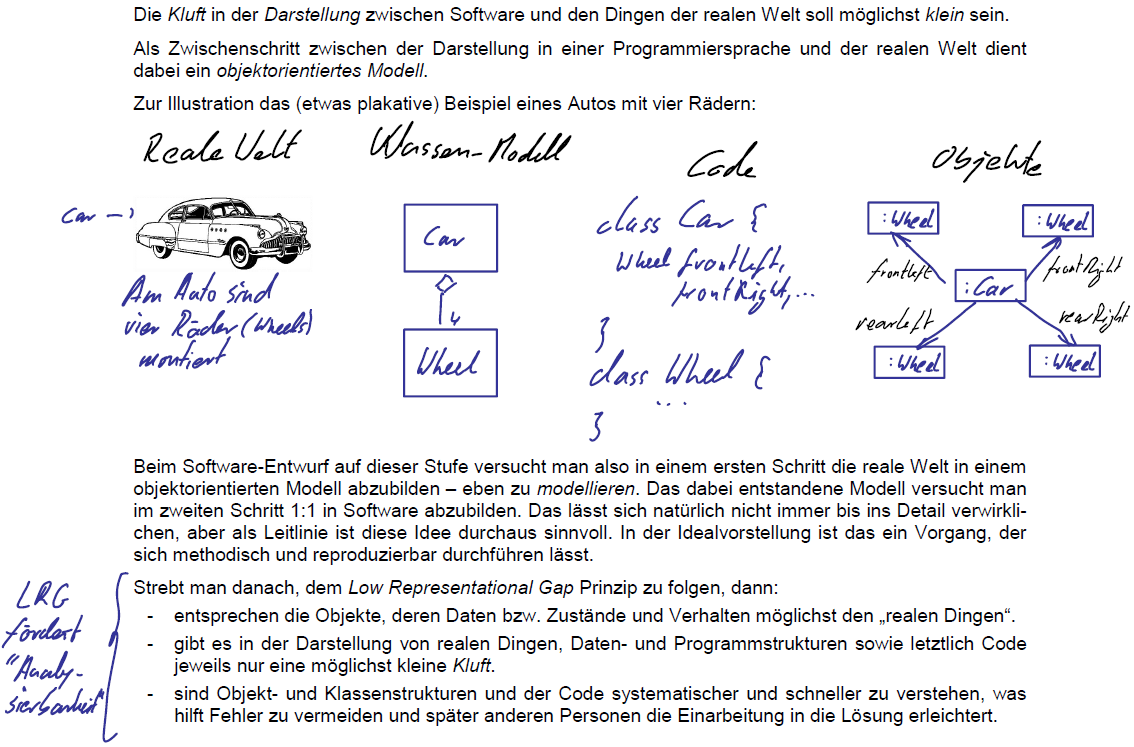
## Software: Vom Programmierer zur Wirkung



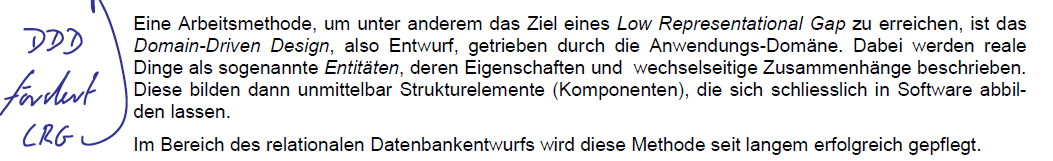


## Prinzipien und Methoden zum Entwurf von Strukturen in der Logical View

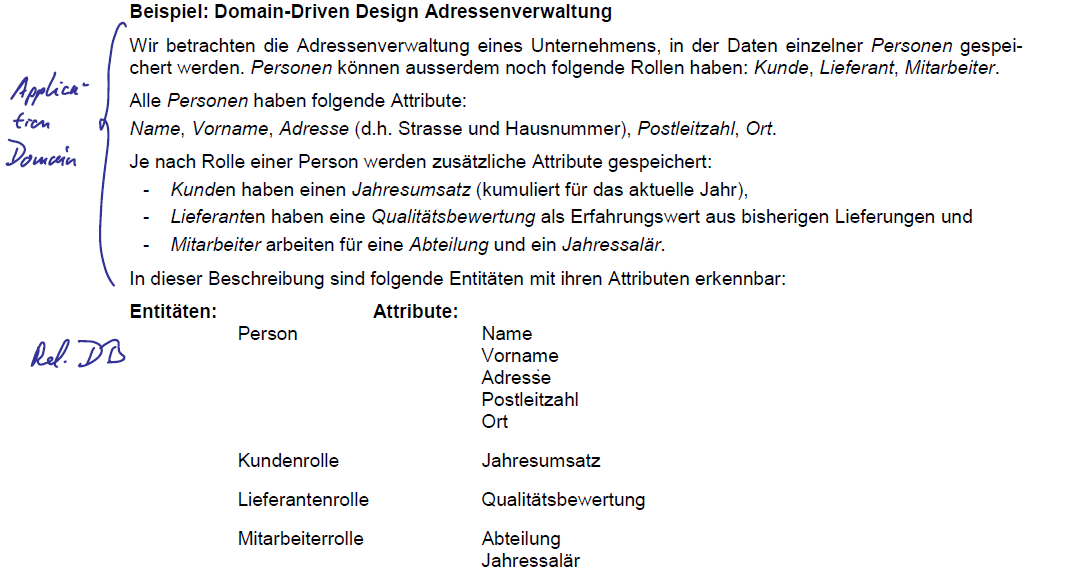
### Entwurfsprinzip Low Representational Gap (LRG)

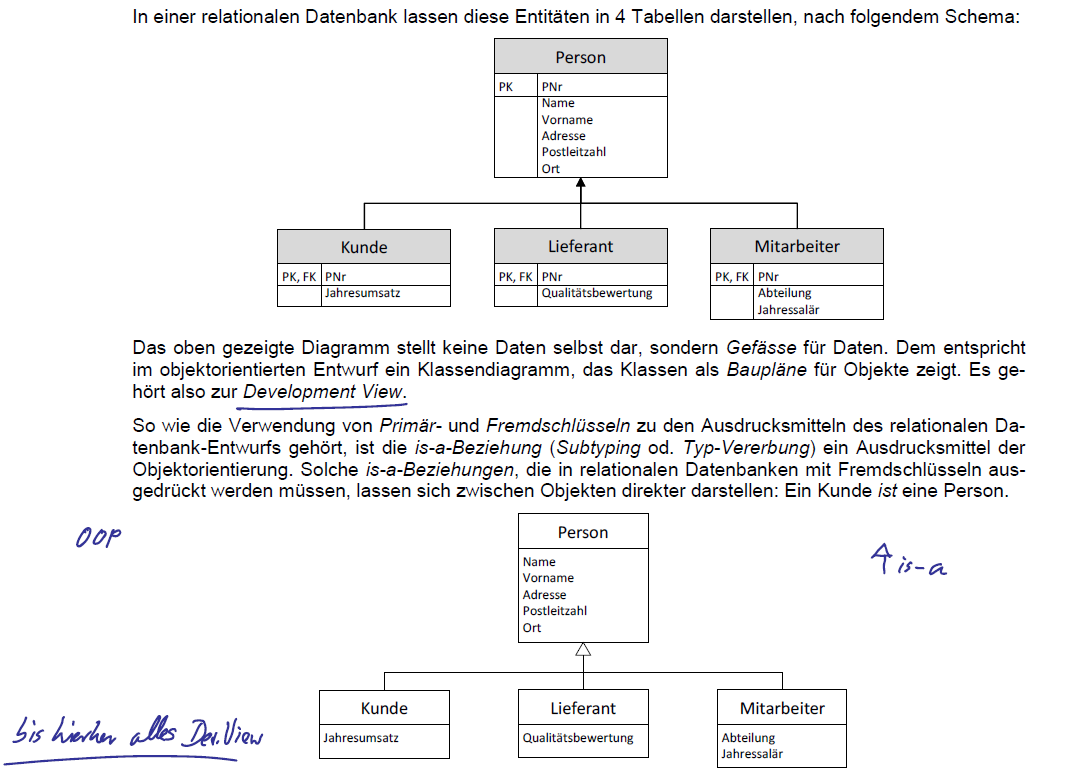


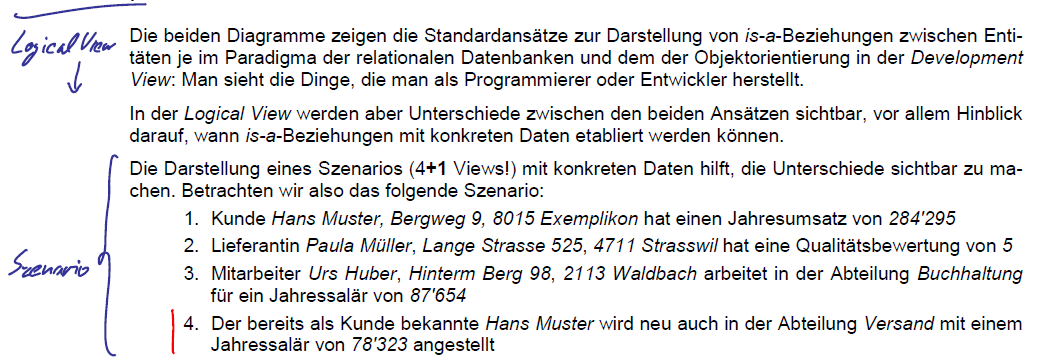
### Entwurfsmethode Domain-Driven Design (DDD)

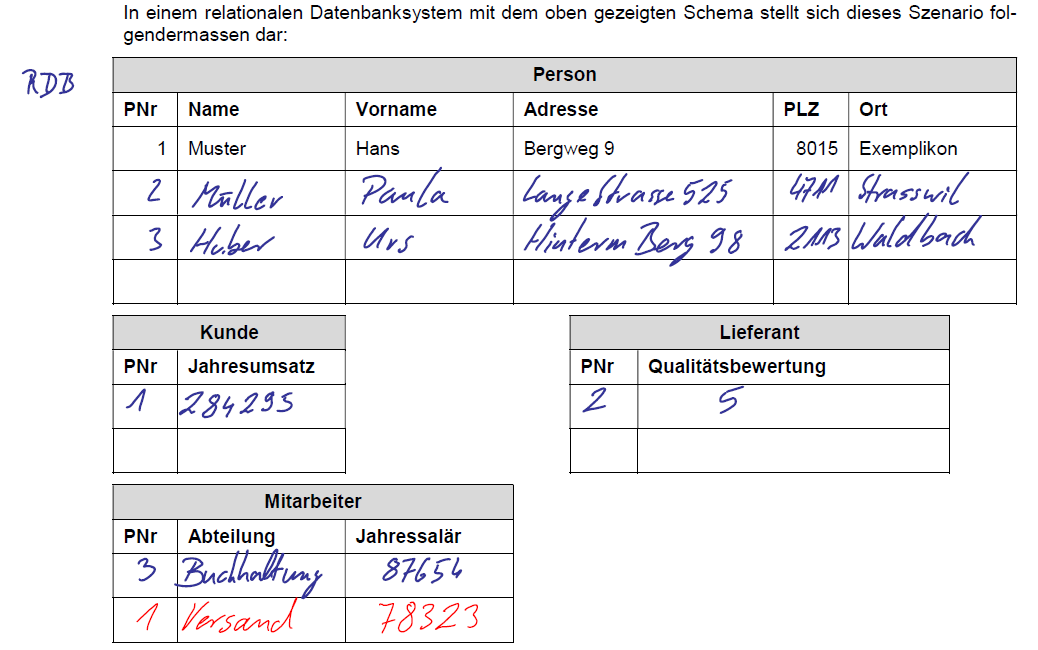


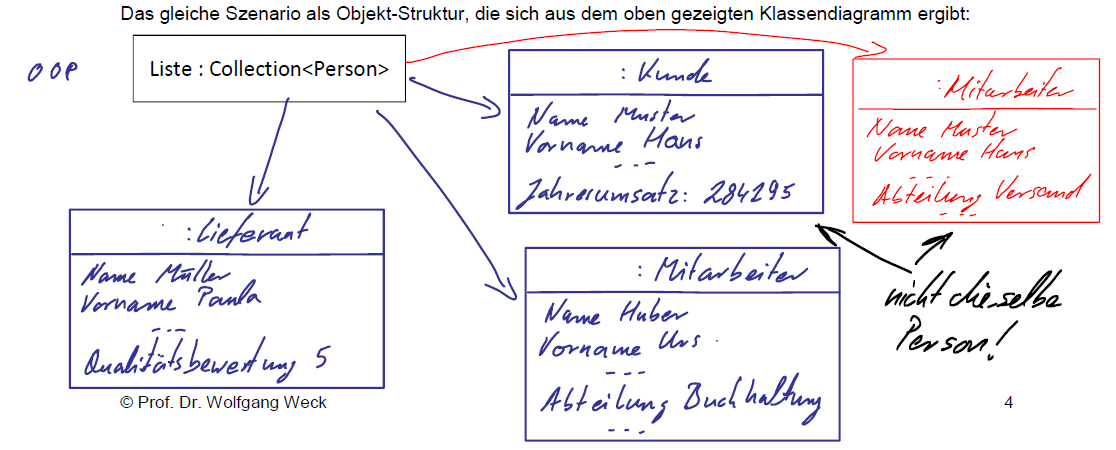
### Beispiel einer Domain-Driven Design Adressenverwaltung

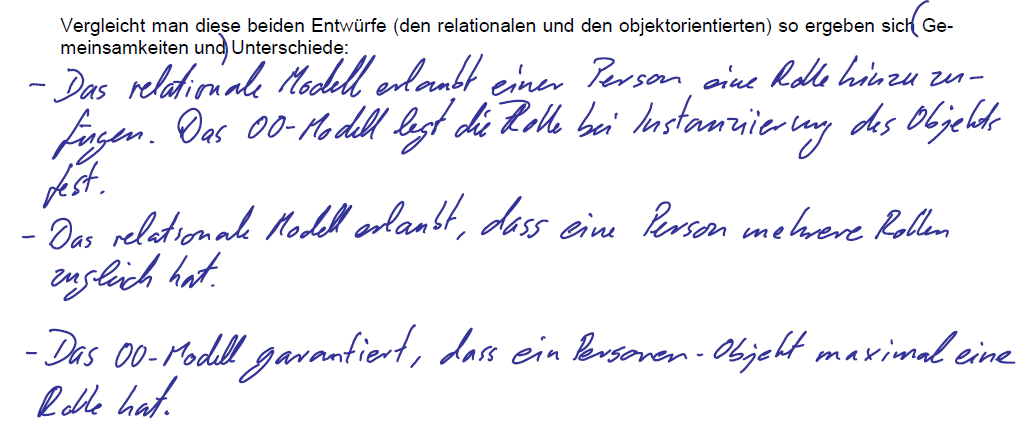


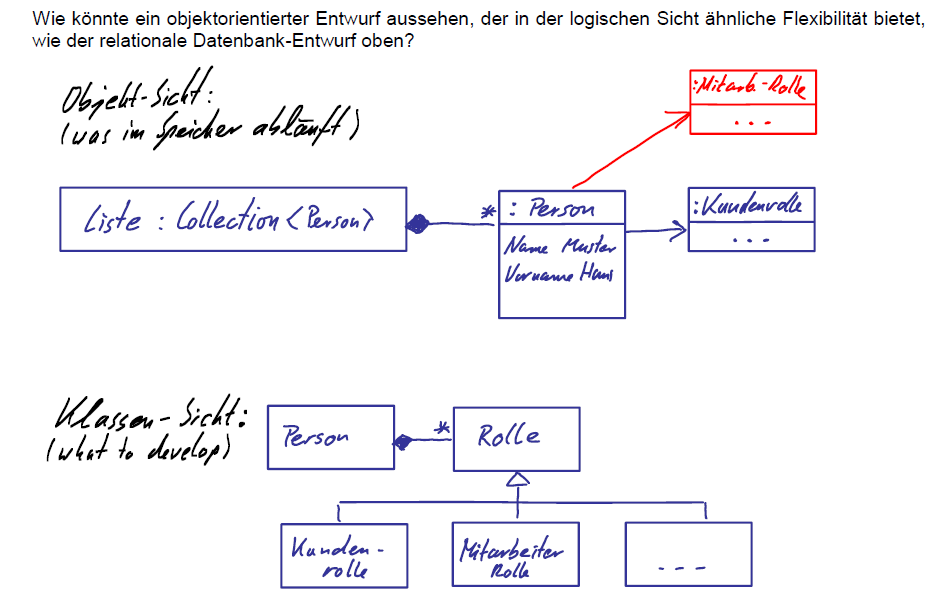


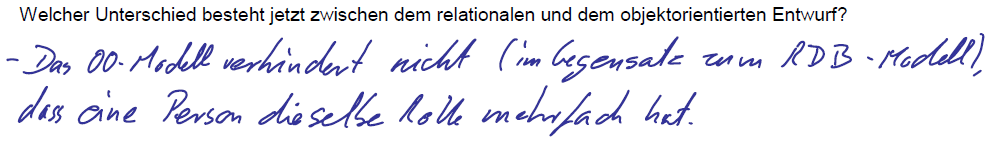




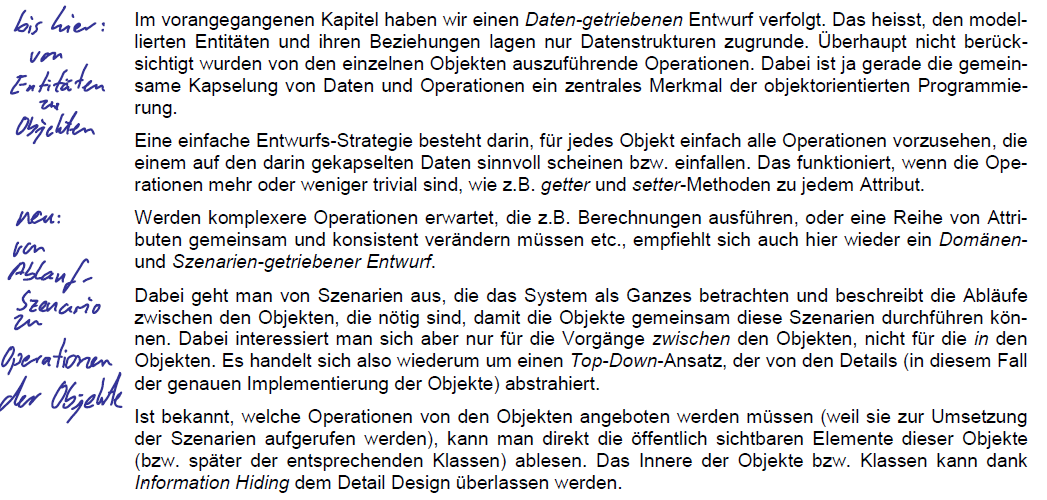


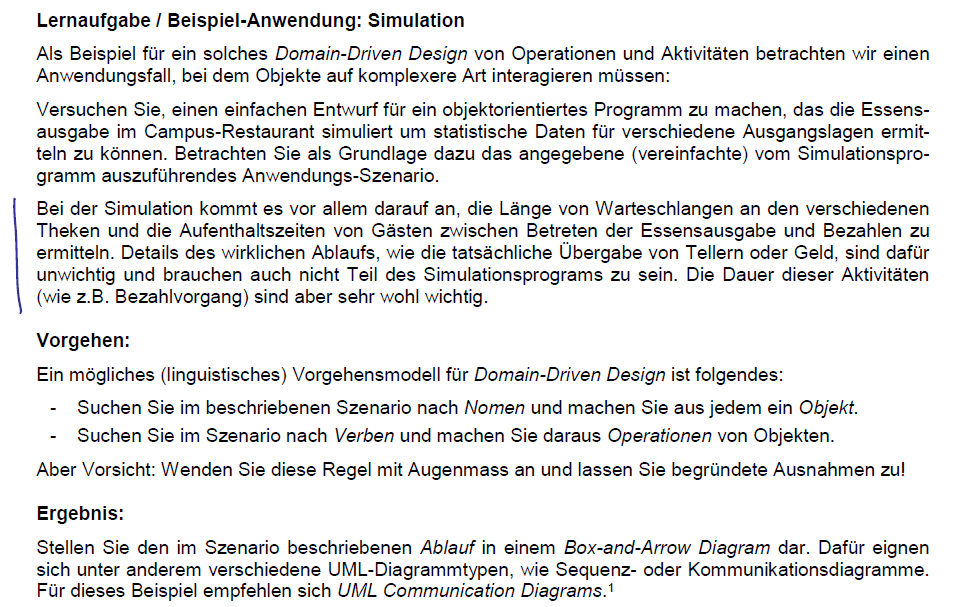


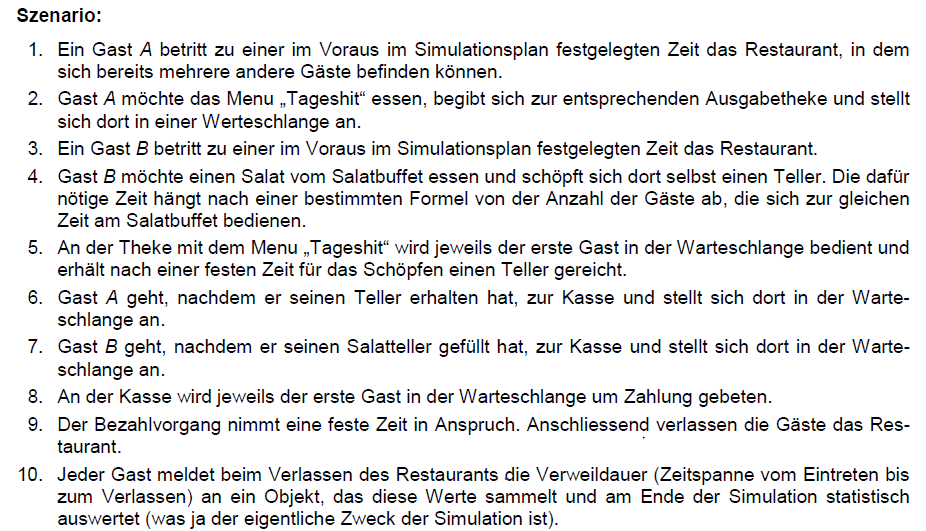


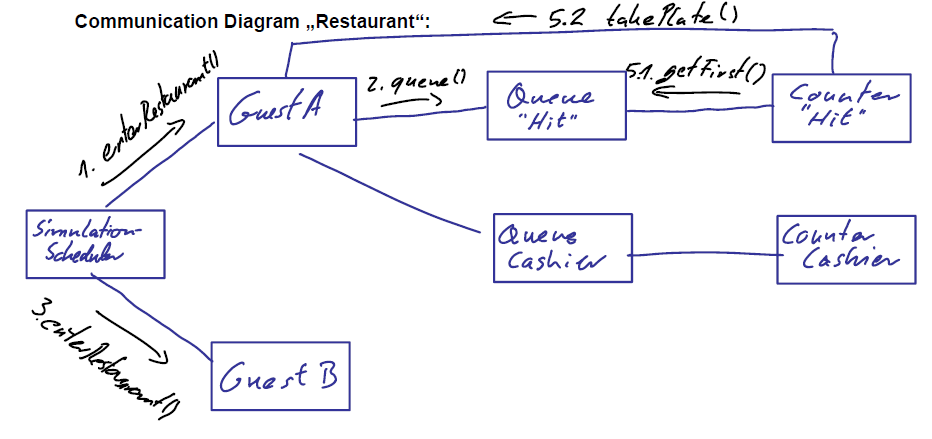


## Abläufe und Operationen in der Logical View



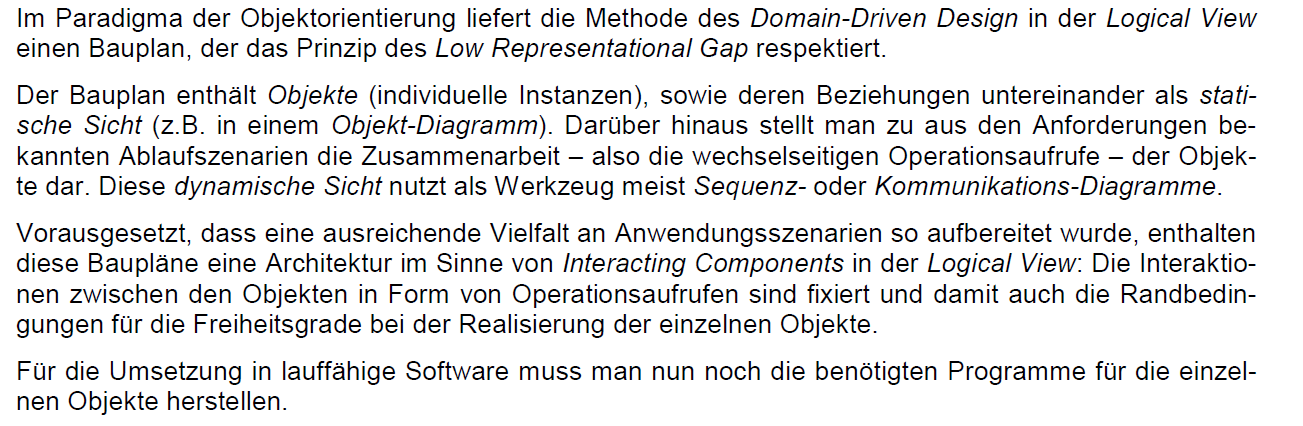




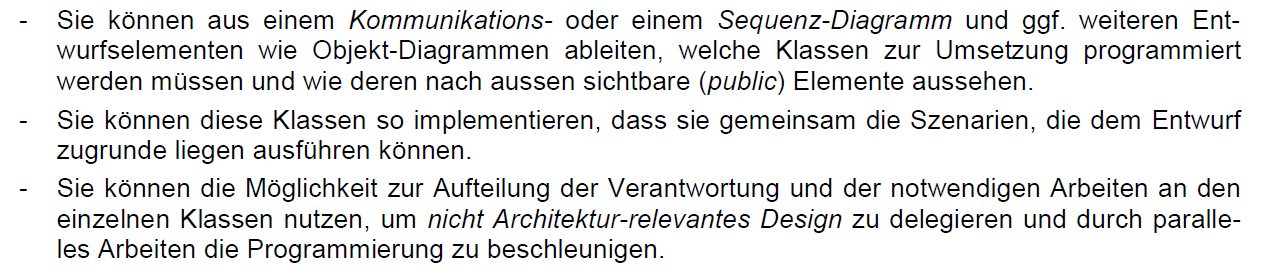


# Woche 3

## Motivation

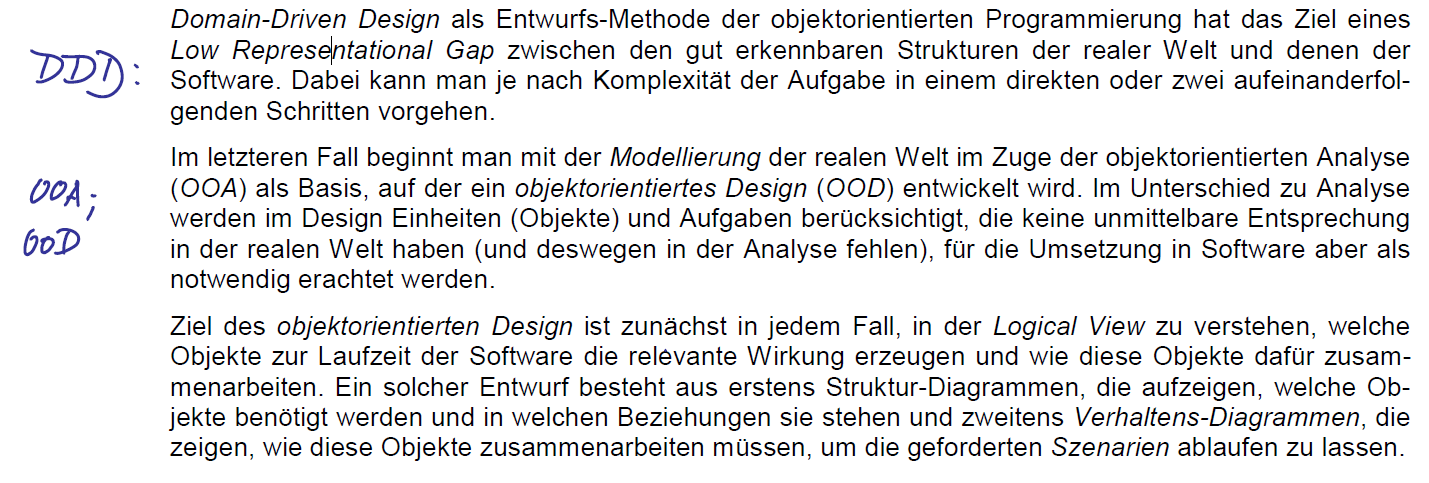


## Lernziele

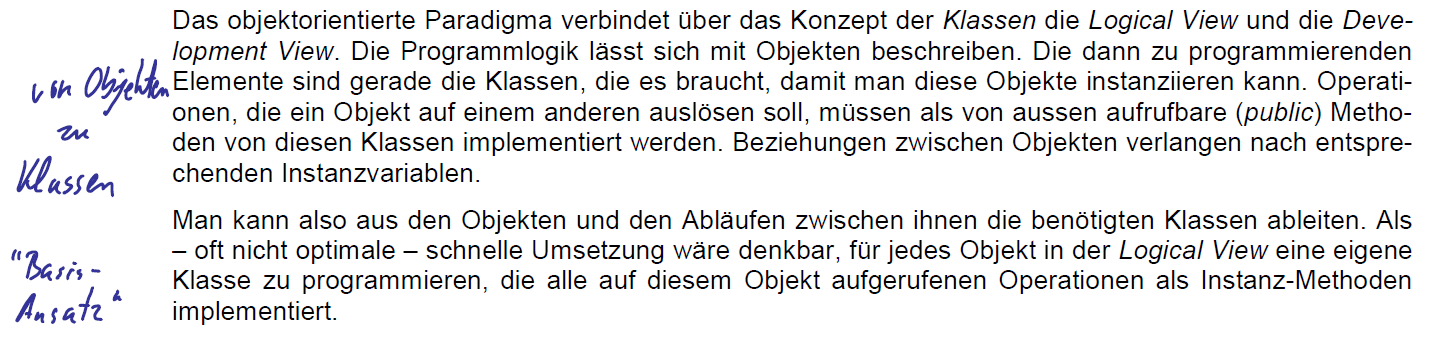


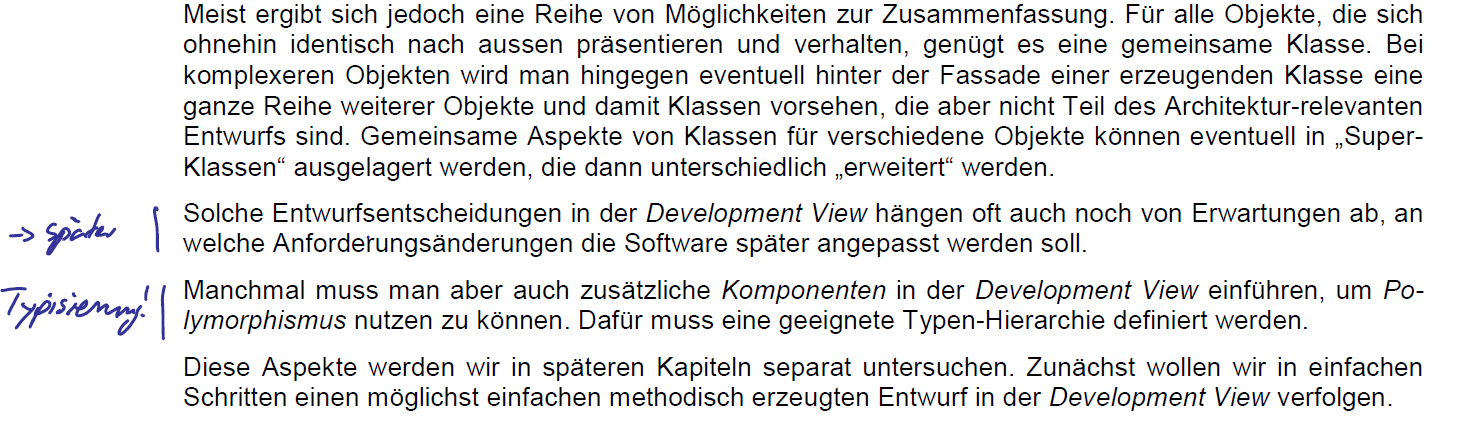
## Vorgehensmodell

### Objektorientierter Entwurf: Logical View

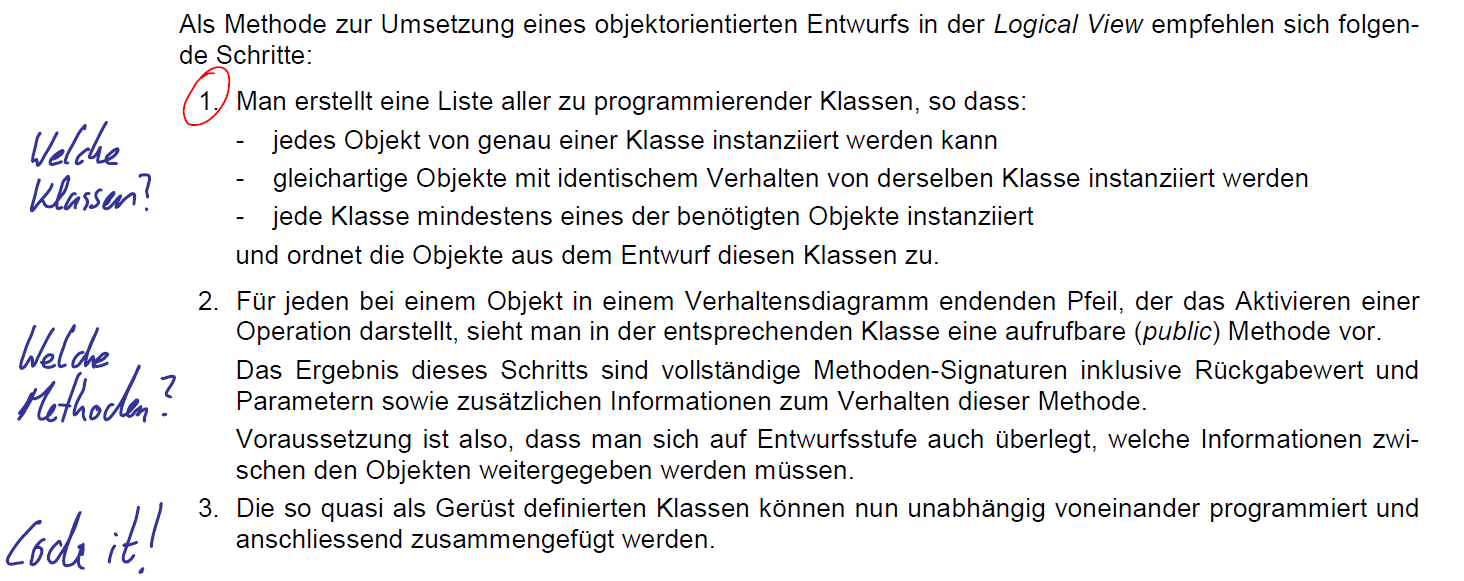


### Objektorientierter Entwurf: Development View



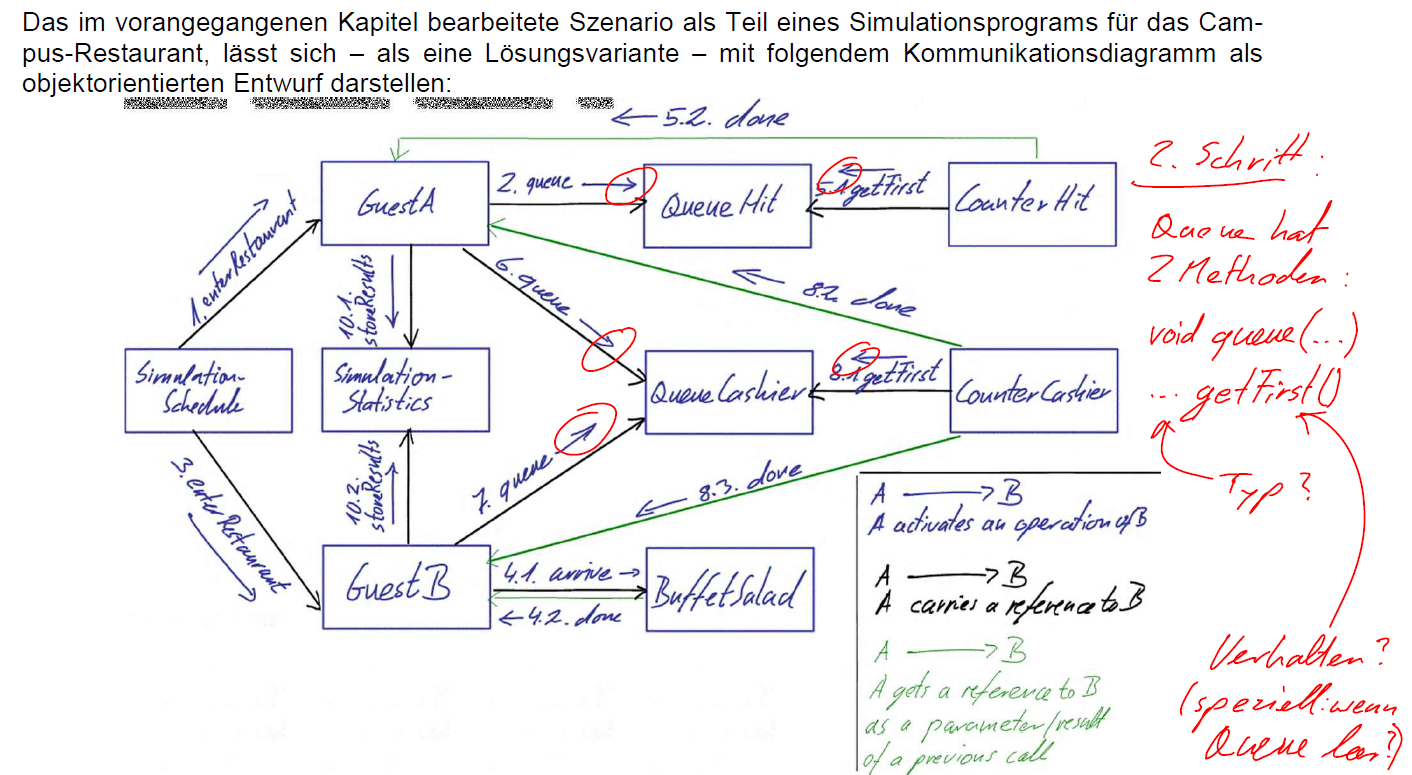


### Methode zur Umsetzung eines objektorientierten Entwurfs

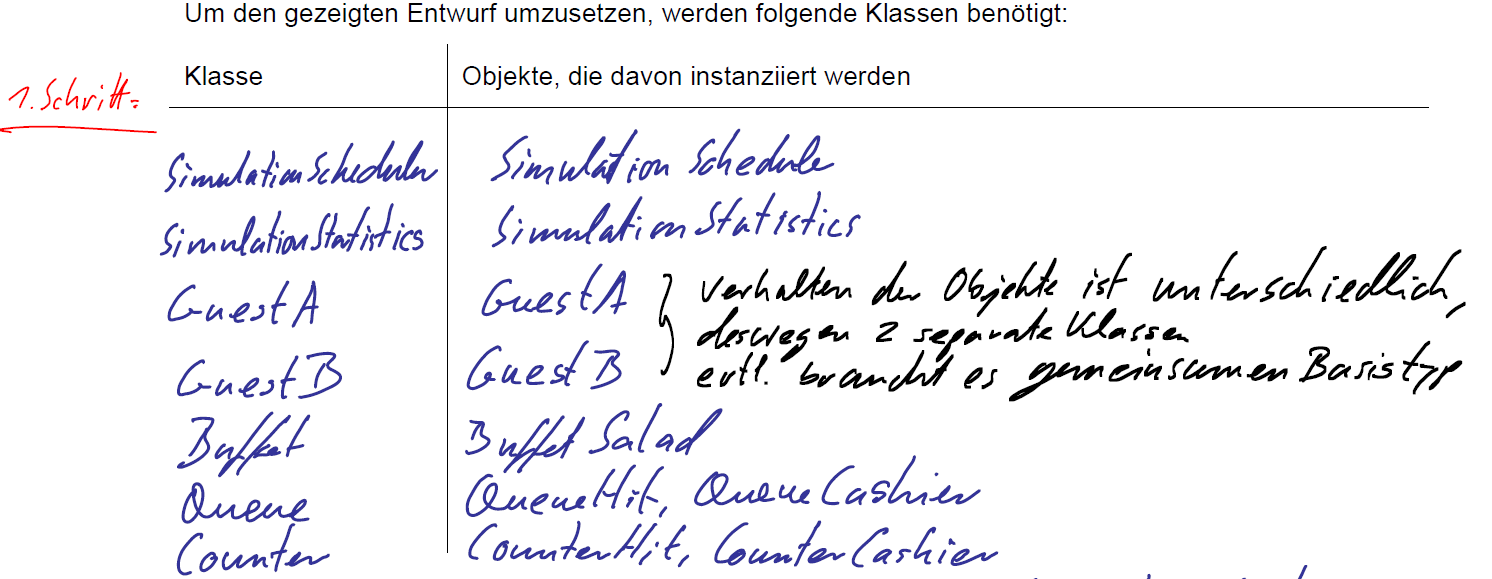


## Beispiel: Entwurf zum Beispiel der Restaurantsimulation

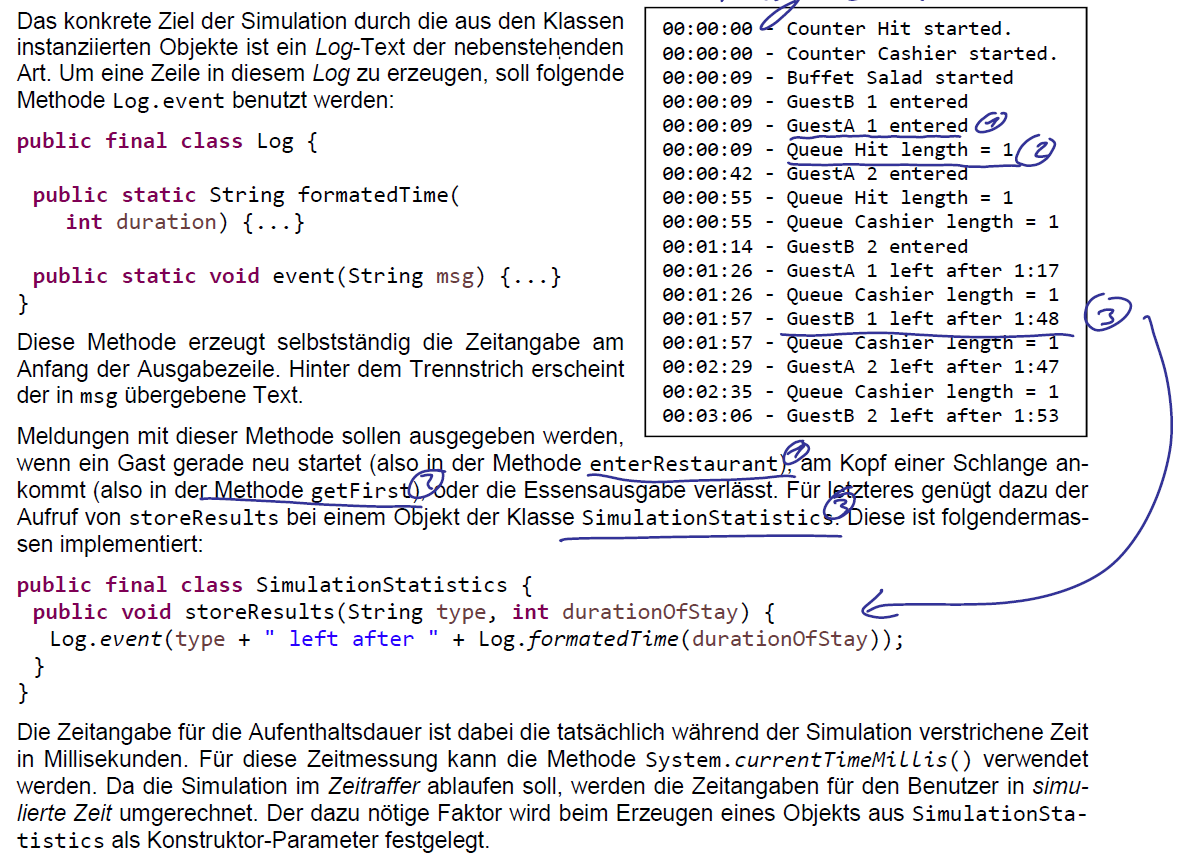
### Bisherige Restaurantsimulation



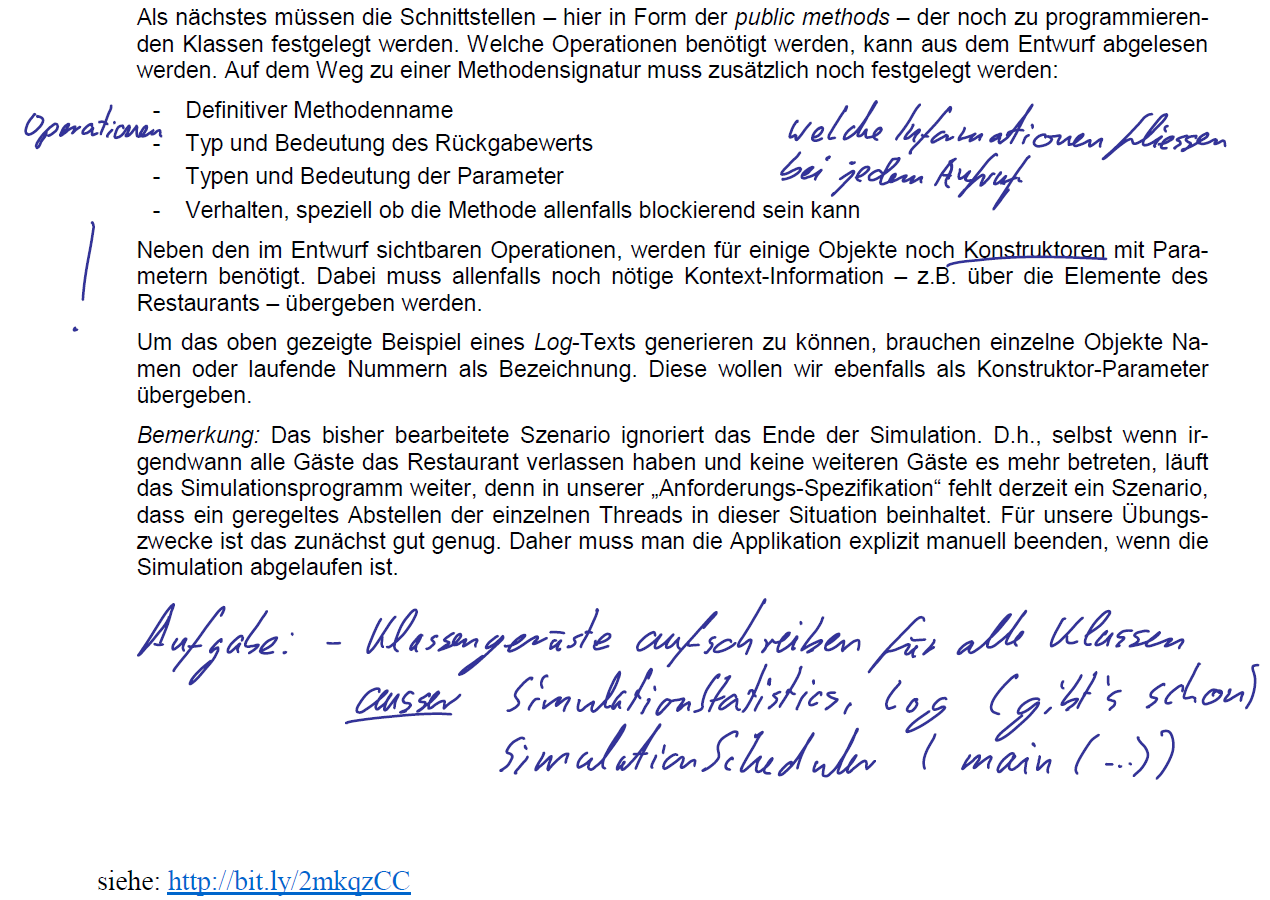
### Umsetzung: Liste der zu programmierenden Klassen



### Umsetzung: Kleine Erweiterung der Aufgabe



### Umsetzung: Schnittstellen der zu programmierenden Klassen



# Woche 4