

Bachelor of Science (BSc) in Informatik
Modul Software-Entwicklung 1 (SWEN1)

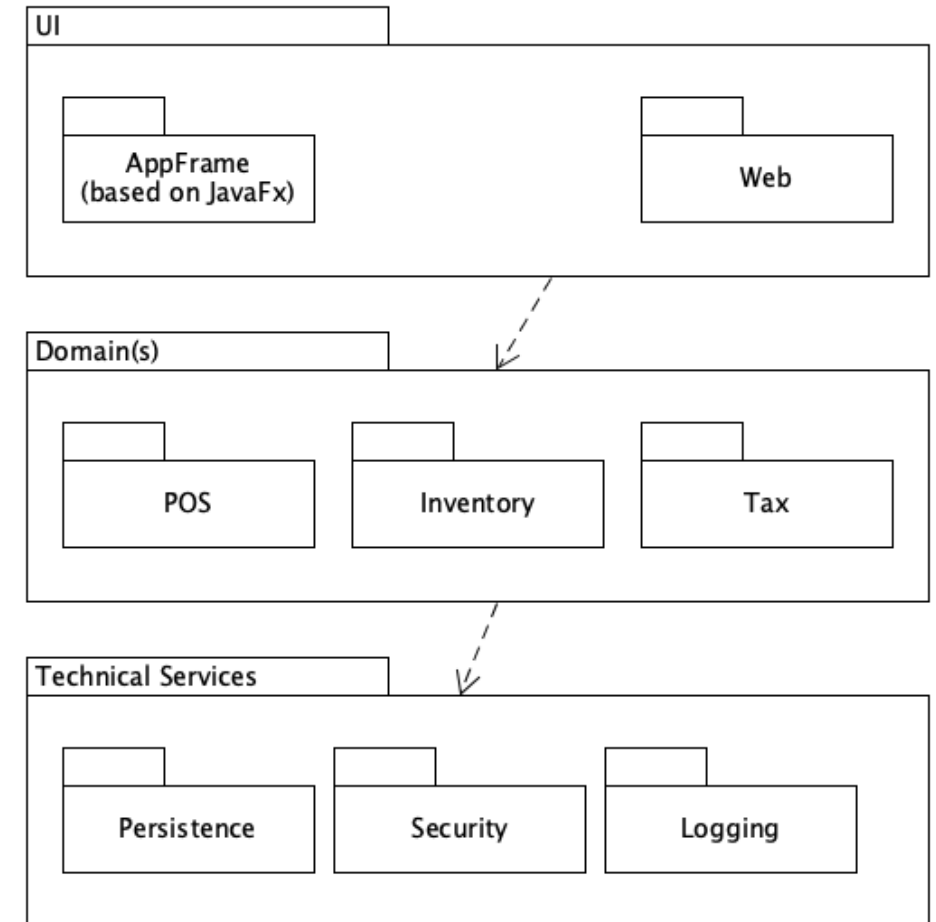
LE 05 – Softwarearchitektur und Design I

SWEN1/PM3 Team:
R. Ferri (feit), D. Liebhart (lieh), K. Bleisch (bles), G. Wyder (wydg)

Ausgabe: HS24

Um was geht es?

- Wie kann ich eine **logische Architektur** aus den Anforderungen ableiten?
- Welche Architekturpatterns gibt es?
- Wie kann ich Architekturentscheide herleiten und dokumentieren
- Wie modelliere ich meine logische Architektur mit der UML, um sie diskutieren und evaluieren zu können?



Lernziele LE 05 – Softwarearchitektur und Design I

- Sie sind in der Lage,
 - die Bedeutung der **logischen Architektur** zu erläutern,
 - die **Einflussfaktoren** aus den nicht-funktionalen Anforderungen für die logische Architektur abzuleiten,
 - den Aufbau von **UML-Paketdiagrammen** zu erklären,
 - das **Schichten-Entwurfsmuster** zu beschreiben,
 - die wichtigsten **Architekturpatterns** zu nennen.



Agenda

- 1. Was ist eine Softwarearchitektur**
2. Architektur aus den Anforderungen ableiten
3. Modulkonzept
4. Architekturen beschreiben
5. UML-Paketdiagramme
6. Ausgewählte Architekturpatterns und Beispielarchitekturen
7. Aufgaben eines Software-Architekten
8. Wrap-up und Ausblick

Was ist Softwarearchitektur?

- Gesamtheit der wichtigen Entwurfs-Entscheidungen
 - Programmiersprachen, Plattformen
 - Aufteilung des Gesamtsystems in **Teilsysteme**, **Bausteine** samt deren **Schnittstellen**
 - Verantwortlichkeiten der **Teilsysteme** und **ihre Abhängigkeiten**
 - Einsatz einer Basis-Technologie oder eines Frameworks, z.B. Java EE
 - Besondere Massnahmen, um Anforderungen erfüllen zu können
 - Z. B. redundante Datenspeicherung
- Grundlagen
 - Anforderungen (vor allem nicht-funktionale)
 - Systemkontext mit Schnittstellen
- Top Level View (*das grosse Ganze*)



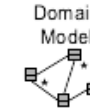
Übersicht Business Analyse vs. Architektur vs. Entwicklung

1 Domänenmodell (Business Modelling) Kontext Diagramm (Business Analyst)

2 Requirements (Business Analyst)

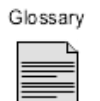
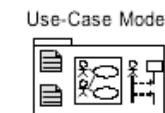
- Liste der Stakeholder
- Vision
- Funktionale Anforderungen:
Use Cases oder User Stories
- Nichtfunktionale Anforderungen:
Supplementary Specification
- Randbedingungen
- Glossar

1 Business Modeling



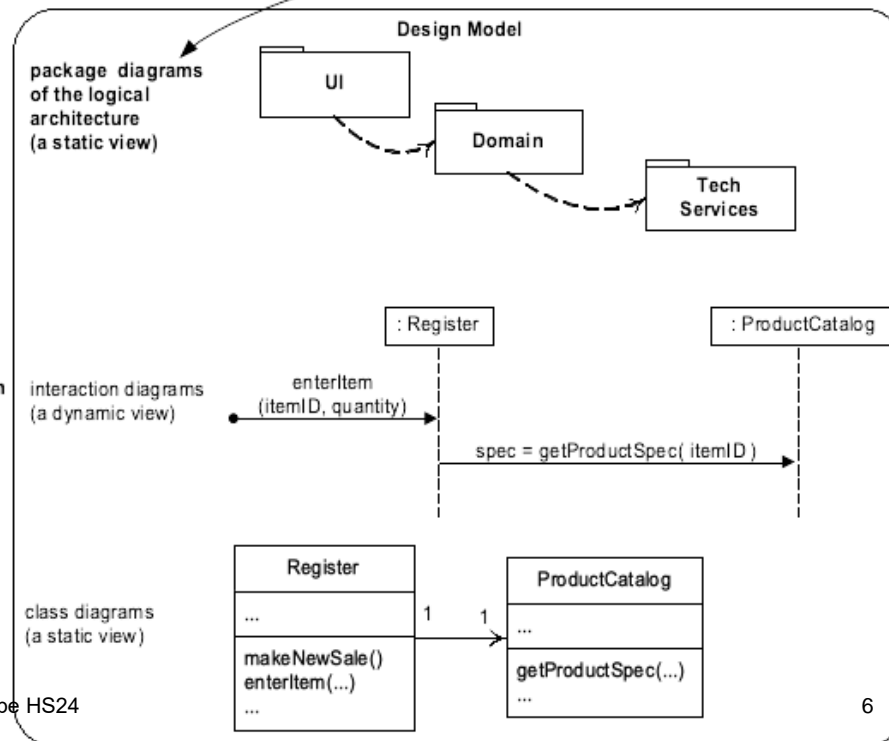
2 Requirements

Requirements



The logical architecture is influenced by the constraints and non-functional requirements captured in the Supp. Spec.

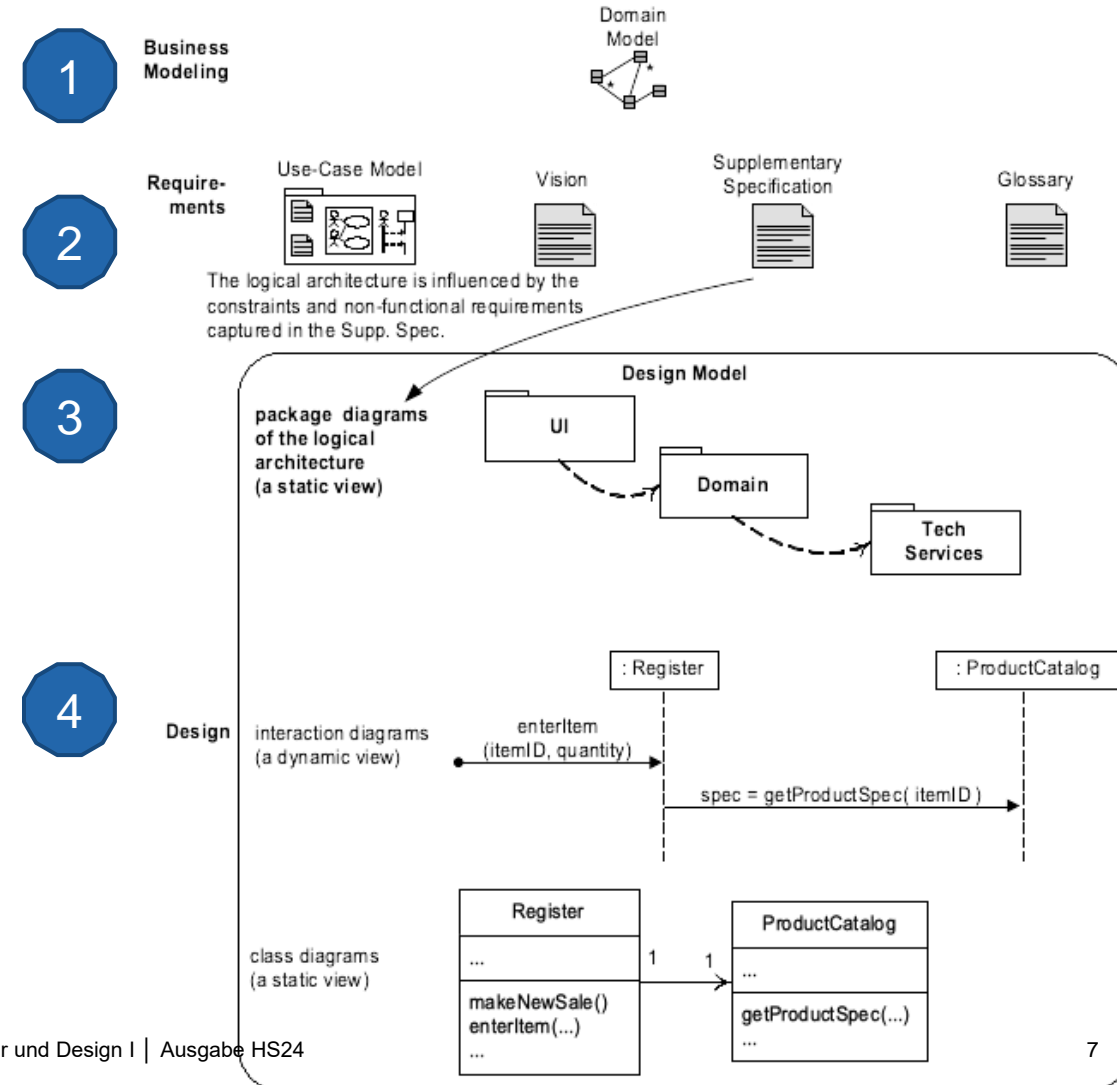
3 Design Model



4 Design

Übersicht Business Analyse vs. Architektur vs. Entwicklung

- 3 Logische Architektur (Software Architekt)
- 4 Umsetzung (Entwicklung)
 - Use Case / User Story Realisierung
 - Anwendung von GRASP
 - DCD – Design-Klassen-Diagramm
 - Interaktionsdiagramme
 - Programmierung
 - Erstellen der Unit- / Integrations-Tests



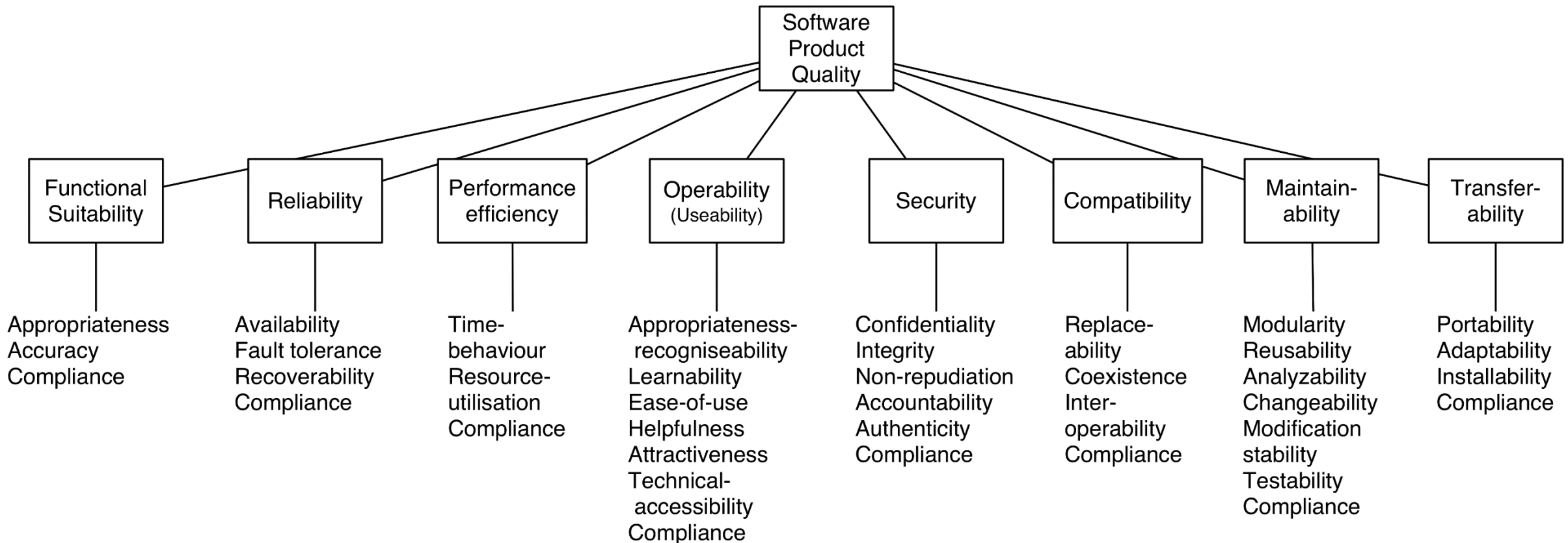
Agenda

1. Was ist eine Softwarearchitektur
- 2. Architektur aus den Anforderungen ableiten**
3. Modulkonzept
4. Architekturen beschreiben
5. UML-Paketdiagramme
6. Ausgewählte Architekturpatterns und Beispielarchitekturen
7. Aufgaben eines Software-Architekten
8. Wrap-up und Ausblick

Architektur aus den Anforderungen ableiten

- Die Architektur muss heutige und zukünftige Anforderungen erfüllen können und Weiterentwicklungen der Software und seiner Umgebung ermöglichen
- Zentrale Aufgabe der **Architekturanalyse**
 - **Analyse** der funktionalen und insbesondere nichtfunktionalen Anforderungen im Hinblick auf die Konsequenzen für die Architektur
 - Unter Berücksichtigung der Randbedingungen und ihrer zukünftigen Veränderungen
 - Dabei müssen **Qualität** und **Stabilität der Anforderungen selbst** überprüft werden.
 - **Lücken in den Anforderungen** müssen aufgedeckt werden.
 - Gerade bei den **nichtfunktionalen Anforderungen** muss hier noch meist nachgebessert werden, da die Anforderungsträger diese häufig als selbstverständlich verstehen.

Nichtfunktionale Anforderungen gemäss ISO 25010



Hauptziele der Architektur

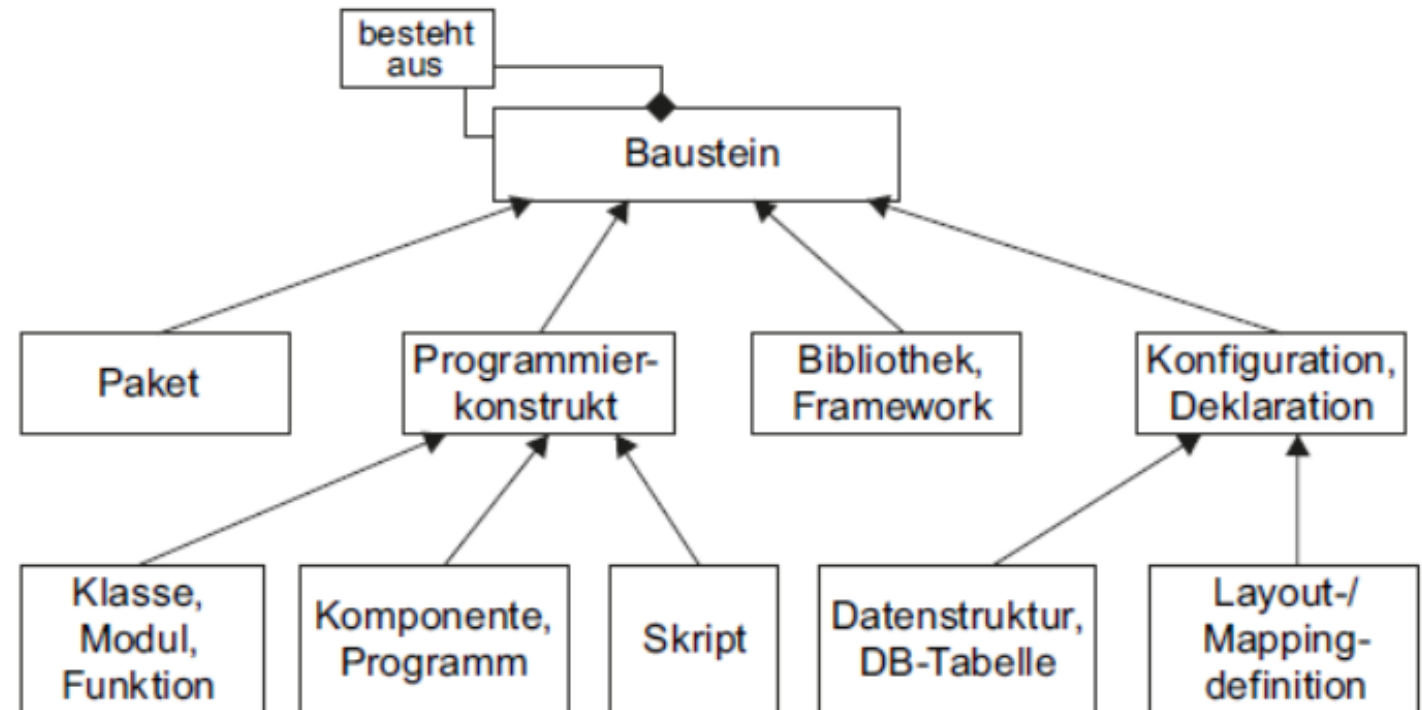
- Muss Erfüllung der Anforderungen unter den gegebenen Randbedingungen ermöglichen
 - Heutige und zukünftige Anforderungen
 - Heutige Randbedingungen und deren zukünftige Veränderung
- Grundprinzip
 - Aufteilung des Gesamtsystems in **möglichst unabhängige Teilsysteme**
 - Können unabhängig entwickelt, weiterentwickelt, angepasst, ersetzt werden

Agenda

1. Was ist eine Softwarearchitektur
2. Architektur aus den Anforderungen ableiten
3. **Modulkonzept**
4. Architekturen beschreiben
5. UML-Paketdiagramme
6. Ausgewählte Architekturpatterns und Beispielarchitekturen
7. Aufgaben eines Software-Architekten
8. Wrap-up und Ausblick

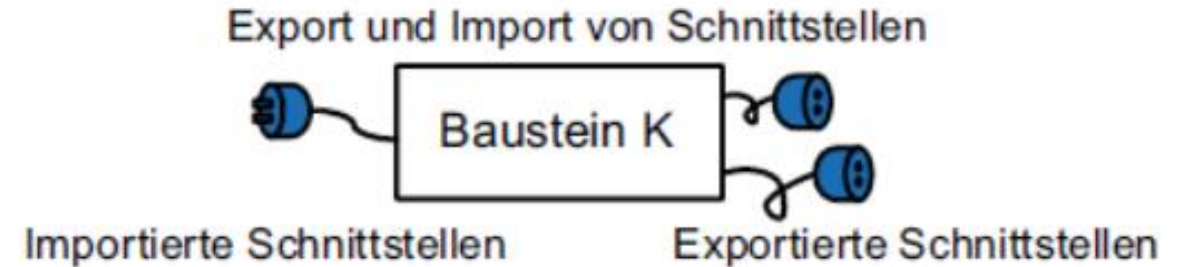
Bausteine und Schnittstellen

- Was ist ein Baustein?
 - Paket
 - Komponente
 - Library
 - Kann aus weiteren Bausteinen aufgebaut sein
- Hat mind. eine Schnittstelle
 - Systemschnittstelle (externe Schnittstelle)
 - Systeminterne Schnittstelle
 - Z.B. Schicht
 - Benutzerschnittstelle



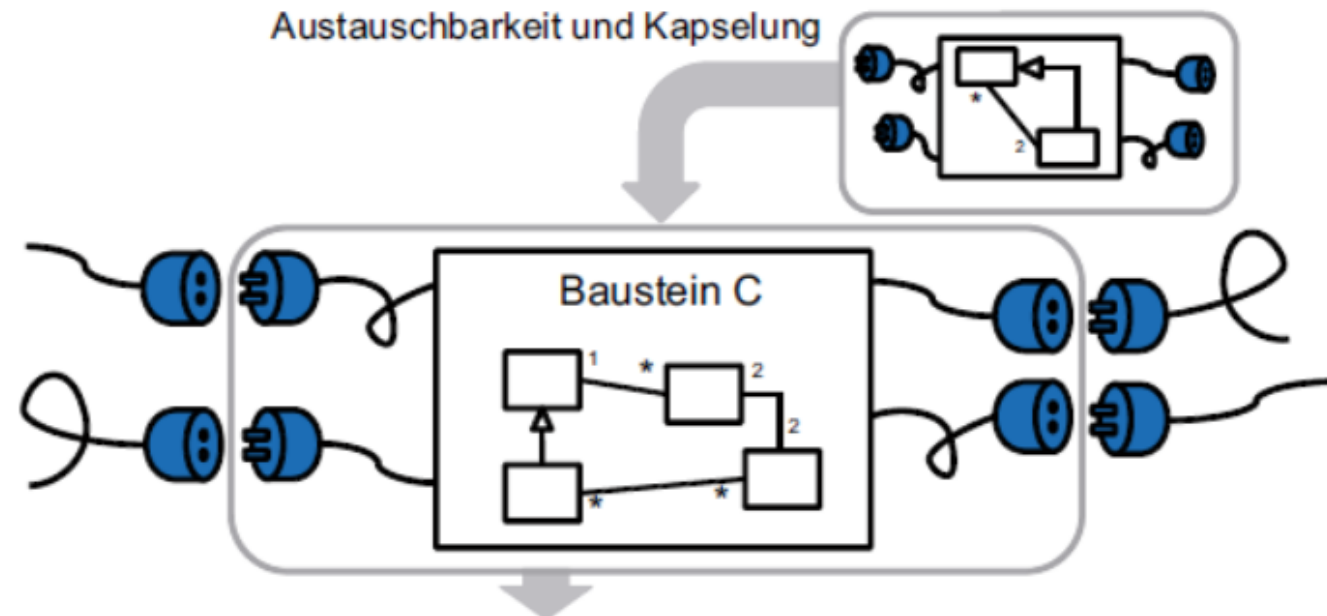
Schnittstellen (Interfaces)

- Ein Modul bietet Schnittstellen an
 - Sogenannte **exportierte** Schnittstellen
 - Definieren angebotene Funktionalität
 - Sind im Sinne eines Vertrags garantiert
 - Einzige Information, die von aussen bekannt sein muss, um Modul zu verwenden
 - Modul kann intern beliebig verändert werden, solange Schnittstellen gleich bleiben
- Importierte Schnittstellen
 - Verwendet ein Modul andere Module, so importiert sie deren Schnittstellen
 - Einzige Kopplung zwischen den Modulen



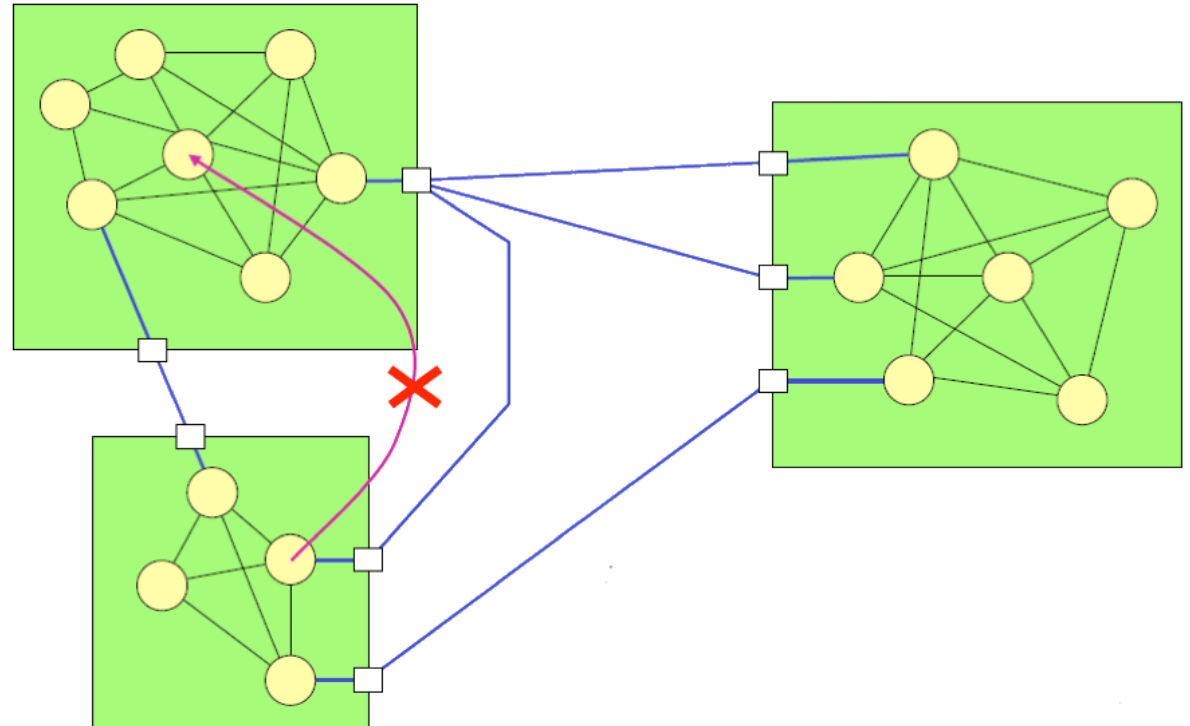
Bausteine und Schnittstellen

- Kapselung und Austauschbarkeit
 - Über die angebotenen und benötigten Schnittstellen **kapselt** der Baustein die **Implementierung** dieser Schnittstellen.
 - Implementation ist unsichtbar für Aussenwelt
 - Daher kann er durch andere Bausteine **problemlos ersetzt** werden, solange dieselben Schnittstellen exportiert werden



Das Prinzip einer modularen Struktur

- Zwischen den Modulen
 - Möglichst schwache Kopplung
 - Kommunikation nur über Schnittstellen
- Innerhalb eines Moduls
 - Alle Funktionalitäten und Daten, die benötigt werden
 - von aussen nicht sichtbar
 - meist starker Zusammenhang



Messung der Güte einer Modularisierung

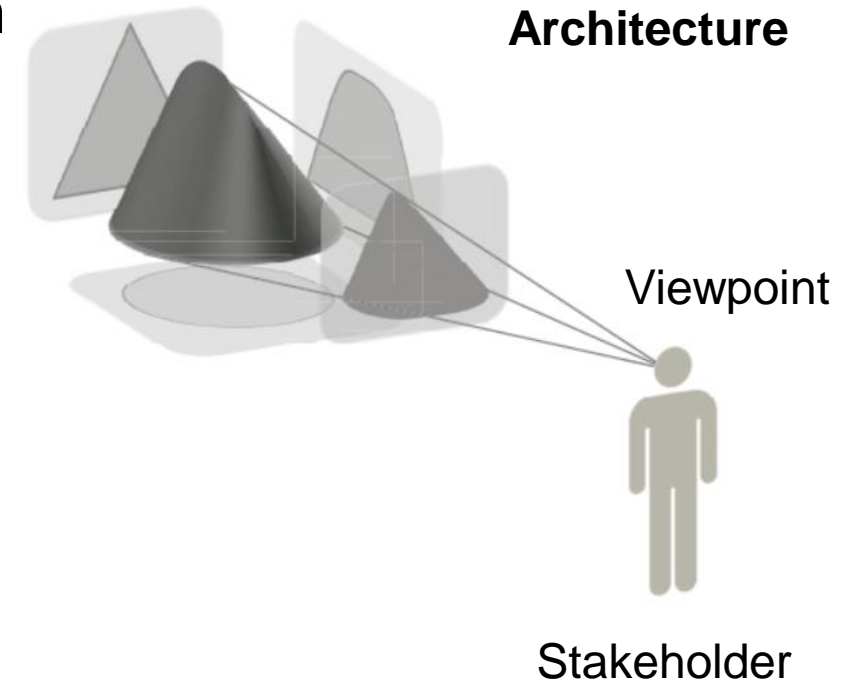
- Zwei charakteristische Masse: **Kohäsion** und **Kopplung** (-> GRASP LE 06)
- **Kohäsion**
 - Ein Mass für die Stärke des inneren Zusammenhangs
 - Je **höher** die **Kohäsion innerhalb eines Moduls**, desto besser die Modularisierung
 - schlecht: zufällig, zeitlich
 - gut: funktional, objektbezogen
- **Kopplung** – Ein Mass für die Abhängigkeit zwischen zwei Modulen.
 - Je **geringer** die wechselseitige **Kopplung zwischen den Modulen**, desto besser die Modularisierung
 - schlecht: Globale Kopplung (Globale Daten)
 - akzeptabel: Datenbereichskopplung (Referenzen auf gemeinsame Daten)
 - gut: Datenkopplung (alle Daten werden beim Aufruf der Schnittstelle übergeben)

Agenda

1. Was ist eine Softwarearchitektur
2. Architektur aus den Anforderungen ableiten
3. Modulkonzept
4. **Architekturen beschreiben**
5. UML-Paketdiagramme
6. Ausgewählte Architekturpatterns und Beispielarchitekturen
7. Aufgaben eines Software-Architekten
8. Wrap-up und Ausblick

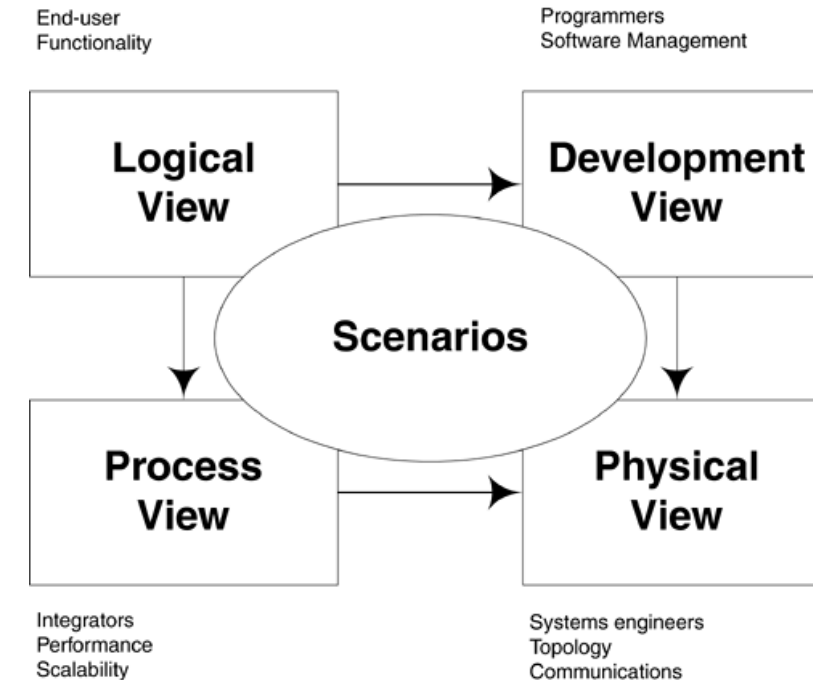
Architekturen beschreiben

- Architektur umfasst verschiedene Aspekte, die je nach Sichtweise wichtig sind
- **Architekturbeschreibungen** sind deshalb in verschiedene **Sichten (Views)** aufgeteilt
 - Sichten sind **Projektionen** der **Softwarearchitektur**
 - Beschreiben die Architektur aus einer bestimmten Sicht
 - Die anderen Aspekte werden ausgeblendet
- Je nach Aufgabe benötigen Stakeholder (Interessenvertreter) **eine unterschiedliche Sicht auf die Architektur**



Das N+1 View Model

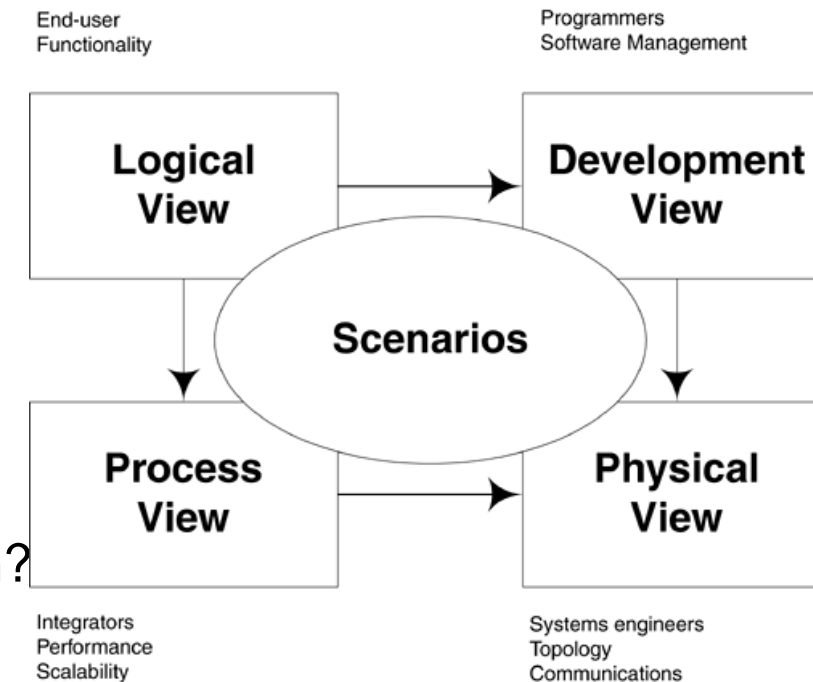
- Philippe Kruchten, 1995: 4 + 1 View Model
- **Logical View:**
 - Welche Funktionalität bietet das System gegen aussen an?
 - Wichtige Aspekte: Schichten, Subsysteme, Pakete, Frameworks, Klassen, Interfaces
 - UML: Systemsequenzdiagramme, Interaktionsdiagramme, Klassendiagramm, Zustandsdiagramme
- **Process View:**
 - Welche Prozesse laufen wo und wie ab im System?
 - Wichtige Aspekte: Prozesse, Threads, Wie werden Anforderungen wie Performance und Stabilität erreicht?
 - UML: Klassendiagramme, Interaktionsdiagramme, Aktivitätsdiagramme.



[Philippe Kruchten, 1995:
http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf](http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf)

Das N+1 View Model

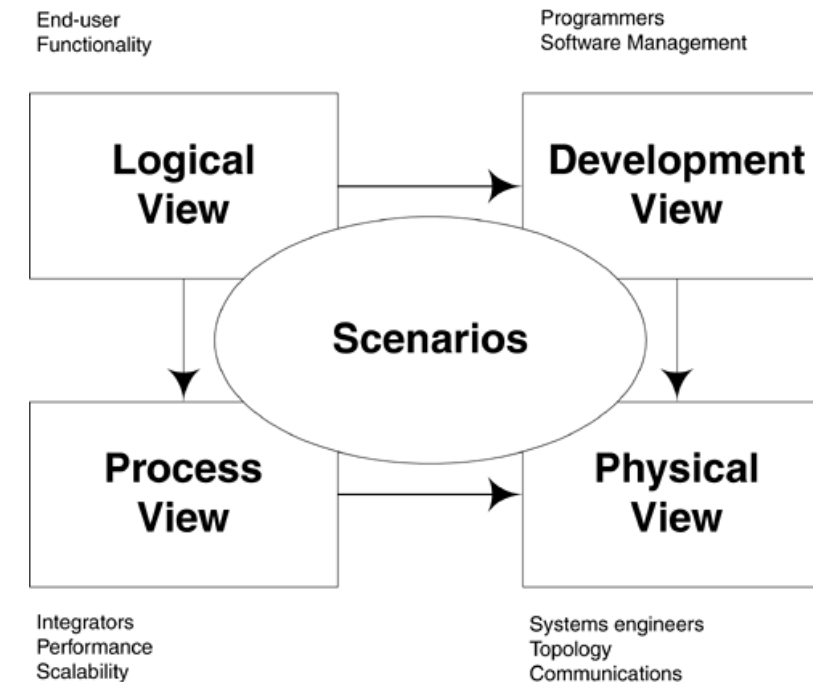
- **Development View (Implementation View):**
 - Wie wurde die logische Struktur (Layer, Schichten, Komponenten) umgesetzt?
 - Wichtige Aspekte: Source Code, Executables, Artefakte
 - UML: Paketdiagramme, Komponentendiagramme
- **Physical View (Deployment View):**
 - Auf welcher Infrastruktur wird ein System ausgeliefert/betrieben?
 - Wichtige Aspekte: Prozessknoten, Netzwerke, Protokolle
 - UML: Deployment Diagram



[Philippe Kruchten, 1995:](http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf)
<http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>

Das N+1 View Model

- **«+1» View: Scenarios (Use Cases)**
 - Welches sind die wichtigsten Use-Cases und ihre nichtfunktionalen Anforderungen? Wie wurden sie umgesetzt?
 - Wichtige Aspekte: Architektonisch wichtige UCs, deren nichtfunktionale Anforderungen und deren Implementation
 - UML: UC-Diagramm, Systemsequenzdiagramme, UC-Realisierungen
- **Weitere mögliche Views**
 - Daten-Sicht
 - Sicherheit



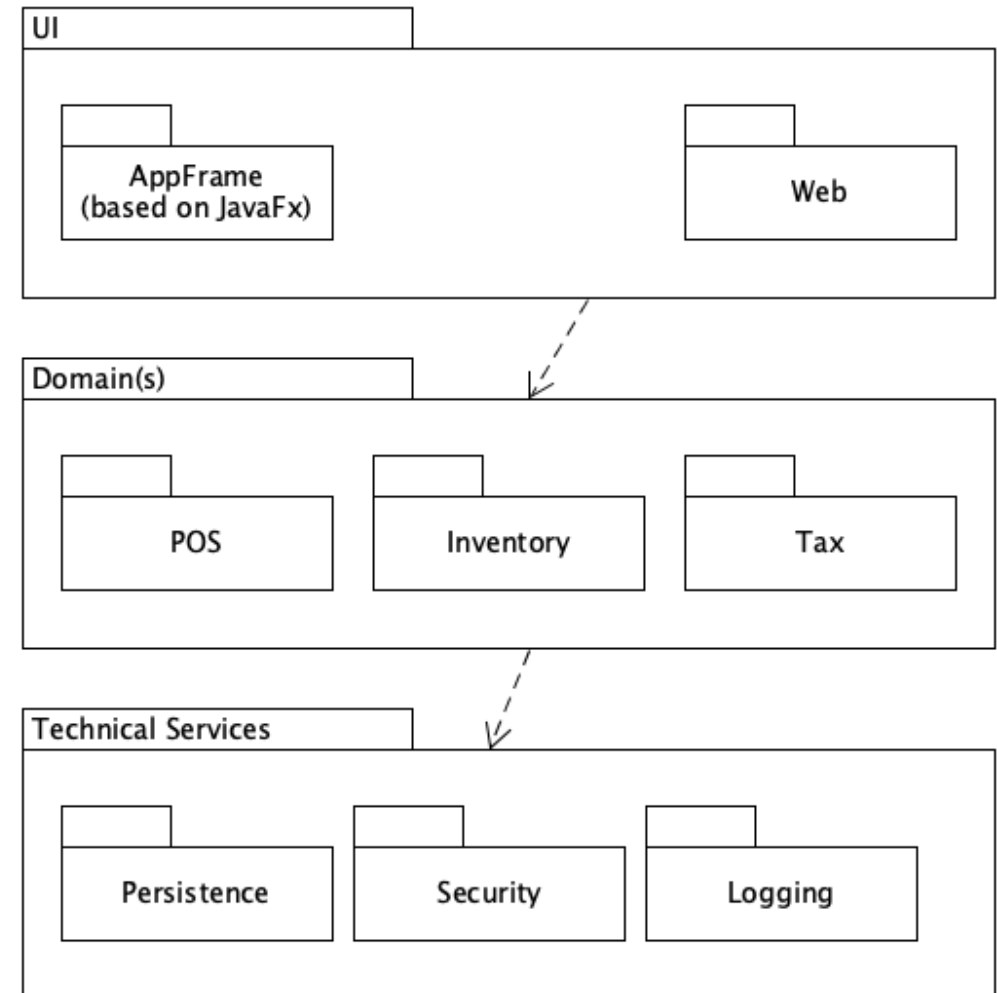
[Philippe Kruchten, 1995:](http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf)
<http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>

Agenda

1. Was ist eine Software Architektur
2. Grundlagen für die Architektur aus den Anforderungen ableiten
3. Modulkonzept
4. Architekturen beschreiben
- 5. UML-Paketdiagramme und Verteilungsdiagramm**
6. Ausgewählte Architekturpatterns und Beispielarchitekturen
7. Wrap-up und Ausblick

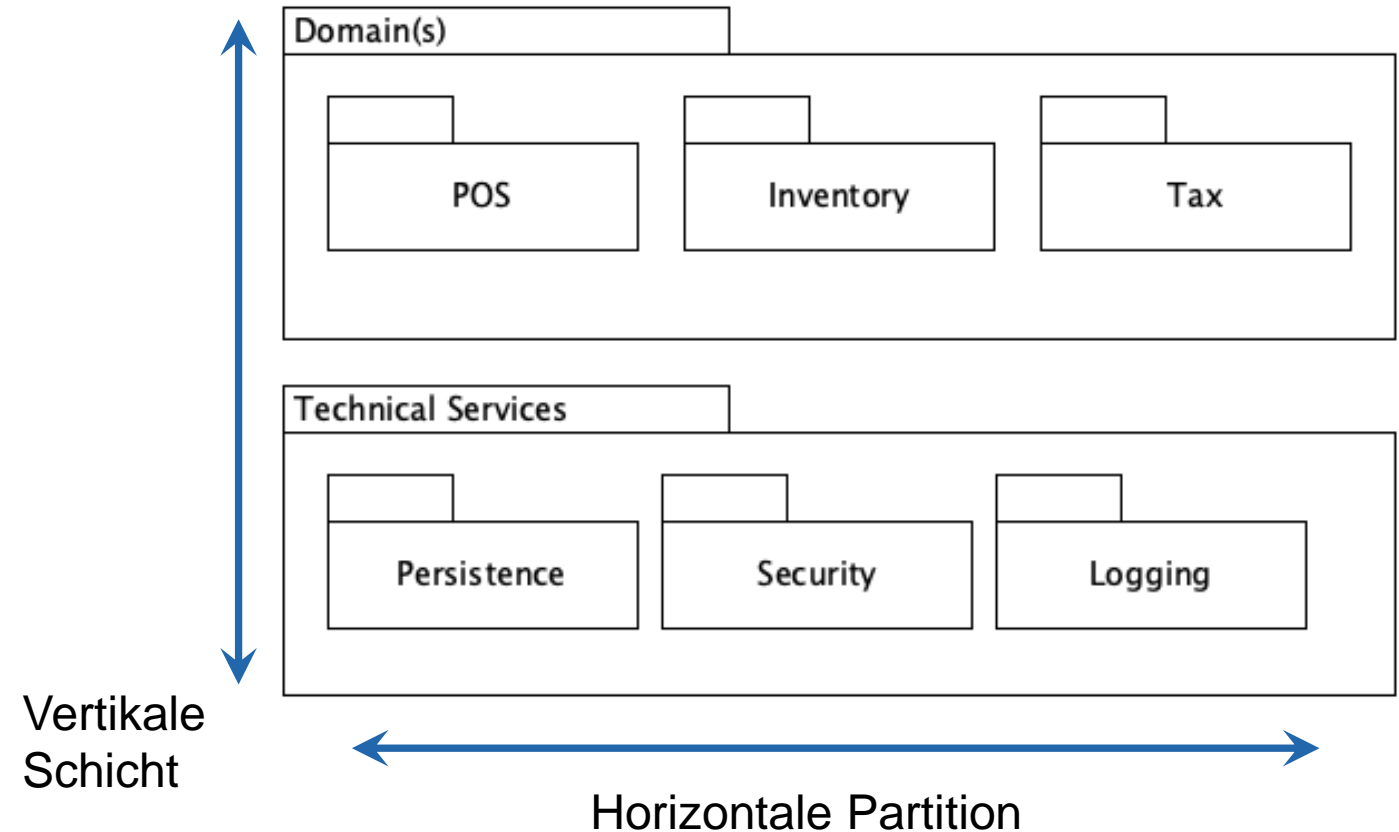
UML-Paketdiagramme

- UML-Paketdiagramme werden häufig zur Dokumentation der Architektur verwendet
 - Mittel, um Teilsysteme zu definieren
 - Mittel zur Gruppierung von Elementen
- Paket enthält Klassen und andere Pakete
 - Ähnlich, aber allgemeiner als Java Packages
- Abhängigkeiten zwischen Paketen



UML-Paketdiagramme: Tier, Layer (Schicht) und Partition

- Layer: logische Struktur
 - Unabhängig betreffend Ausführung
- Physical Tier
 - auf welchem Rechnerknoten
- Partition
 - Unterteilung in einzelne Themen

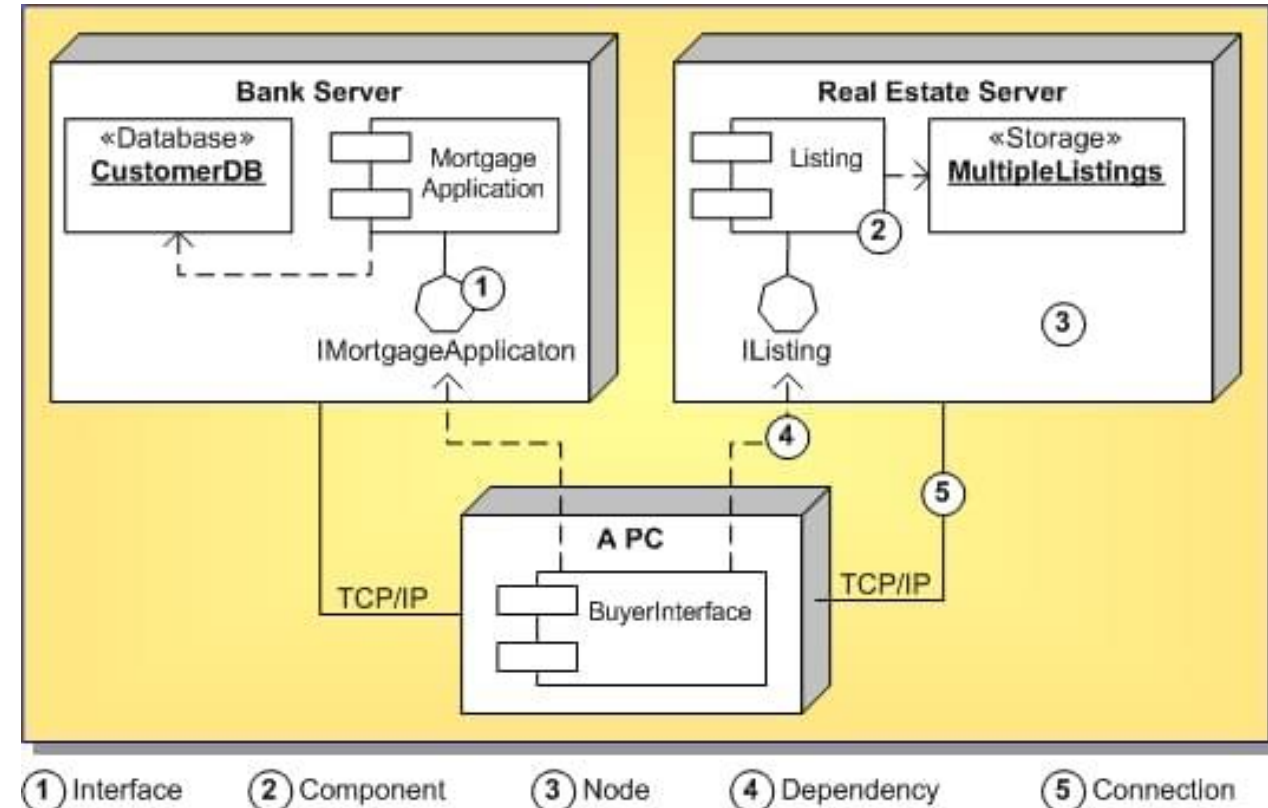


Pakete umsetzen

- Java : Packages
 - `com.mycompany.nextgen.ui.swing`
 - `com.mycompany.nextgen.domain.sales`
 - `com.mycompany.service.persistence`
 - `org.apache.log4j`
- C# : Namensräume und Assemblies
- Tipp für wiederverwendbare Pakete: Keine projektspezifischen Namen

Verteilungsdiagramm

- Das Verteilungsdiagramm
 - dient der Darstellung der Verteilung von Komponenten auf Rechenknoten mit Abhängigkeiten, Schnittstellen und Verbindungen
 - gehört zu den Diagrammen der statischen Modellierung



Agenda

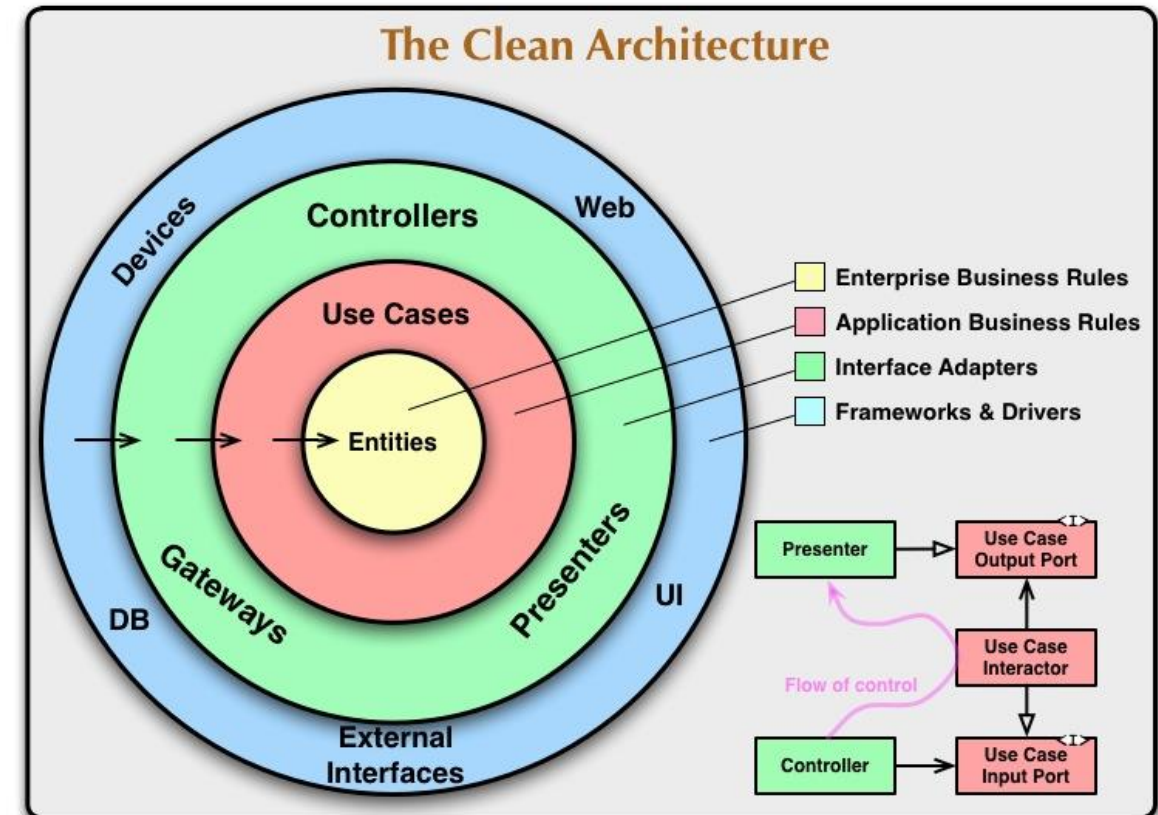
1. Was ist eine Software Architektur
2. Grundlagen für die Architektur aus den Anforderungen ableiten
3. Modulkonzept
4. Architekturen beschreiben
5. UML-Paketdiagramme
6. **Ausgewählte Architekturpatterns und Beispielarchitekturen**
7. Wrap-up und Ausblick

Ausgewählte Architekturpatterns

Pattern	Beschreibung
Layered Pattern	Strukturierung eines Programms in Schichten
Client-Server Pattern	Ein Server stellt Services für mehrere Clients zur Verfügung
Master-Slave Pattern	Ein Master verteilt die Arbeit auf mehrere Slaves
Pipe-Filter Pattern	Verarbeitung eines Datenstroms (filtern, zuordnen, speichern)
Broker Pattern	Meldungsvermittler zwischen verschiedenen Endpunkten
Event-Bus Pattern	Datenquellen publizieren Meldungen an einen Kanal auf dem Event-Bus. Datensenken abonnieren einen bestimmten Kanal
MVC Pattern	Eine interaktive Anwendung wird in 3 Komponenten aufgeteilt: Model, View – Informationsanzeige, Controller – Verarbeitung der Benutzereingabe

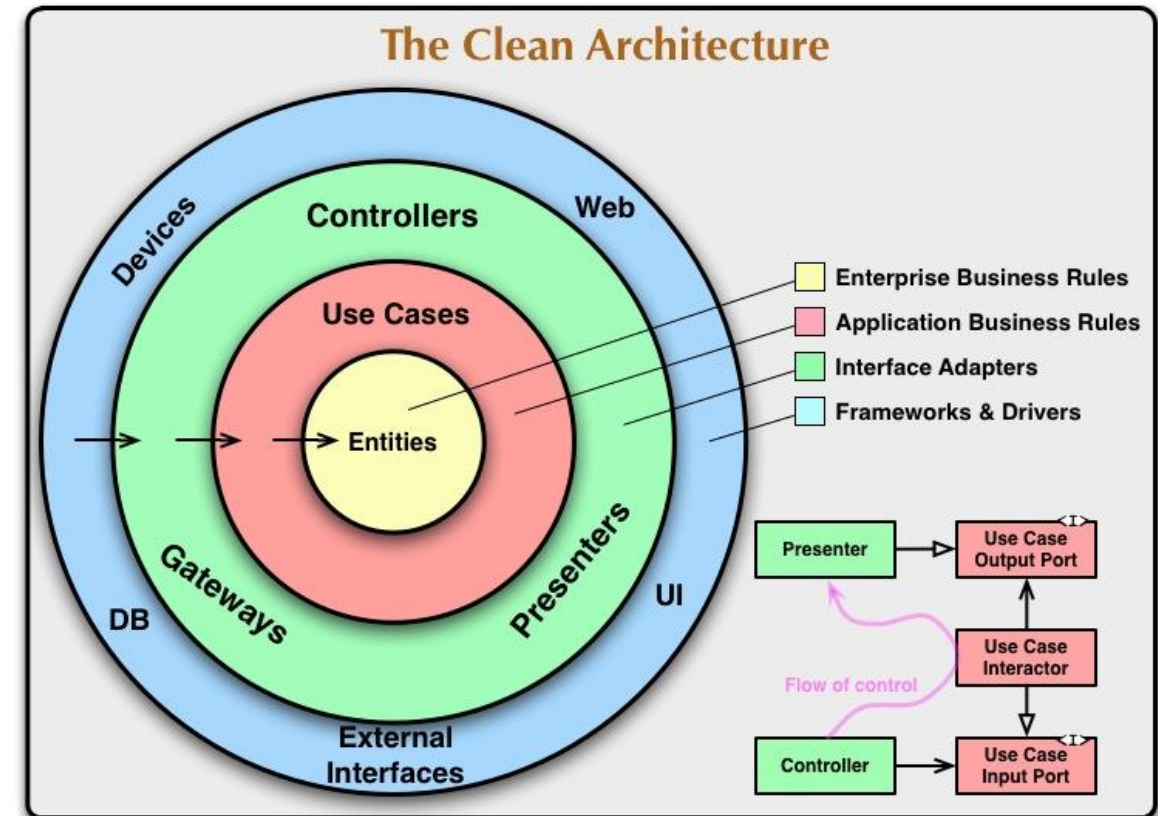
Clean Architecture (1/2)

- Unter dem von Uncle Bob (Robert C. Martin) geprägten Begriff versteht man
 - Unabhängigkeit von einem bestimmten Framework
 - Business Rules können unabhängig von UI, DB, Web Server getestet werden
 - Unabhängig von einem bestimmten UI
 - Unabhängig von einer bestimmten DB



Clean Architecture (2/2)

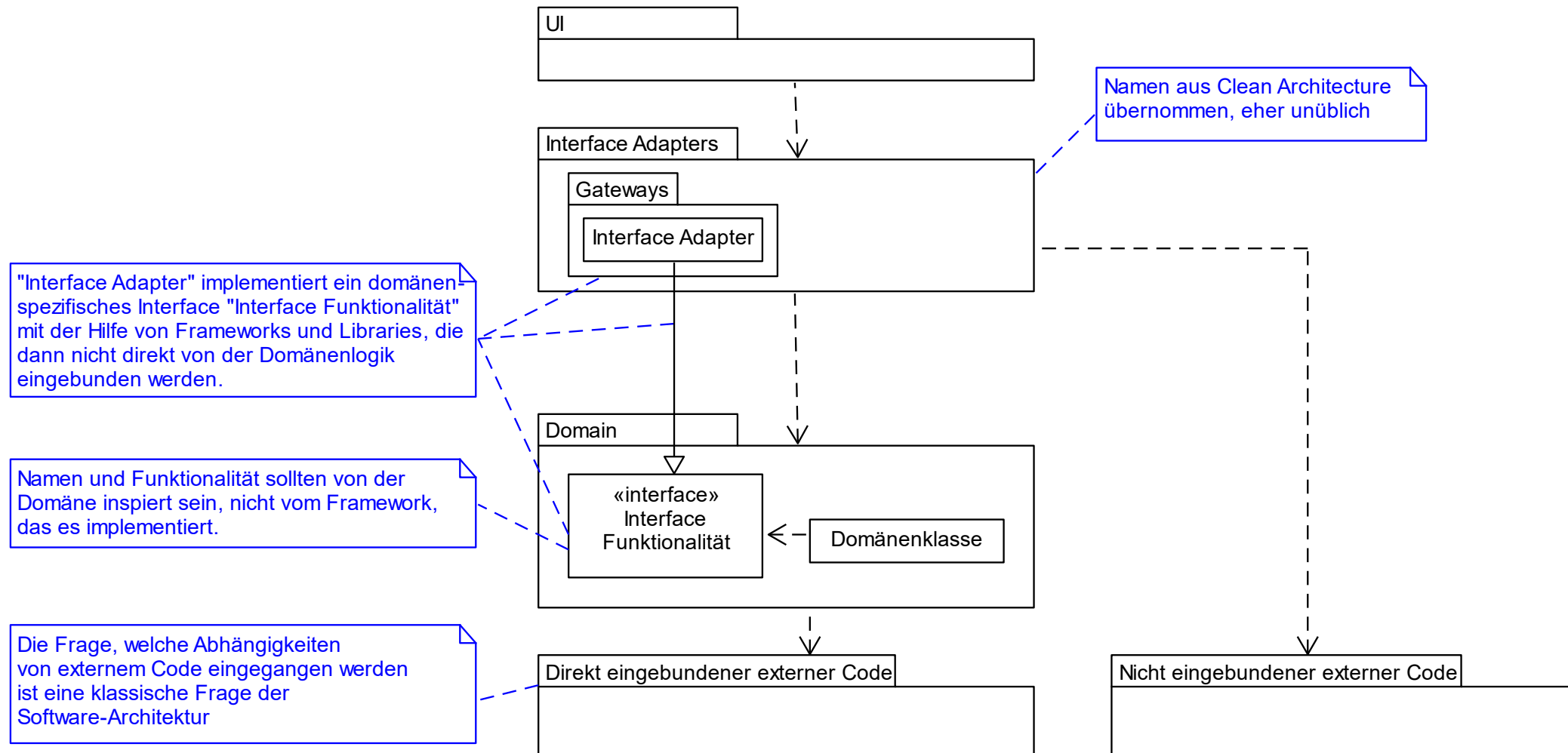
- Entities
 - Kapseln die Business Rules gültig für das gesamte Unternehmen
- Use Cases
 - Beinhalten die Business Rules einer Anwendung, orchestriert die Verwendung der Entities
- Interface Adapters
 - Adapter für die Konvertierung von Daten aus Datenbank oder Web
- Frameworks and Drivers



Clean Architecture Bemerkungen

- Die Domänenlogik hat **keine** Abhängigkeiten zu **externem** Code
 - Frameworks, technische Services, Bibliotheken
 - Externe Dienste, UI, Testcode
- Beurteilung
 - **Vieles** wird heute als «Best Practice» angeschaut und ist auch so im Schichtenkonzept umgesetzt.
 - Es gibt aber kaum ein Projekt, das «Clean Architecture» **vollständig** umsetzt. Gewisser externer Code wird immer integriert.
 - Platform Code (z.B. Java Bibliotheken) und Libraries mit engem Funktionsumfang ist eher unproblematisch.
 - Frameworks sollten zurückhaltend eingesetzt und sorgfältig ausgewählt werden.

Clean Architecture im Schichtenkonzept



Kritische Bemerkungen zu Frameworks

- Frameworks tendieren dazu, im Laufe der Zeit **immer mehr Funktionalität** zu «sammeln».
- Was auf den ersten Blick positiv scheint, kann im zweiten Blick zu **inkonsistentem Design und funktionalen Überschneidungen** führen, die den Einsatz immer mehr erschweren.
- Der **Einsatz** eines Frameworks sollte **gut überlegt werden**.
- Einerseits erfordert dies gute Kenntnisse des Frameworks, andererseits ist nach der «Verheiratung» der Anwendung mit dem Framework eine **«Scheidung» nur noch schwierig und mit hohem Aufwand möglich**.
- Allenfalls sollte das Framework **nur über eigene Schnittstellen verwendet werden** (keine direkte Abhängigkeit), was aber unter Umständen die Nützlichkeit des Einsatzes in Frage stellt.

Agenda

1. Was ist eine Software Architektur
2. Grundlagen für die Architektur aus den Anforderungen ableiten
3. Modulkonzept
4. Architekturen beschreiben
5. UML-Paketdiagramme
6. Ausgewählte Architekturpatterns und Beispielarchitekturen
7. **Wrap-up und Ausblick**

Wrap-up

- Die **Softwarearchitektur** definiert die grundlegenden Prinzipien und Regeln für die Organisation eines Systems sowie dessen Strukturierung in **Bausteinen** und **Schnittstellen** und deren **Beziehungen** zueinander wie auch zur **Umgebung**.
- Zentrale Aufgabe der **Architekturanalyse** ist es, die funktionalen und insbesondere nichtfunktionalen Anforderungen als Grundlage für den Entwurf der Softwarearchitektur zu untersuchen.
- Eine Softwarearchitektur wird aus **verschiedenen Sichten** beschrieben.
- Eine **logische Software-Architektur** wird mit einem **UML-Paketdiagramm** dargestellt.
- Es gibt verschiedene **Architekturpatterns**, die eine Standardarchitektur für eine bestimmte Problemstellungen bieten (Layered Pattern, Client-Server, Master-Slave etc.).

Ausblick

- In der nächsten Lerneinheit werden wir folgende Fragen behandeln:
 - Wie modelliere ich mein Design mit der UML, um es diskutieren und evaluieren zu können?
 - Wie realisiere ich einen Use Case mit Klassen, die klare Verantwortlichkeiten haben, wartbar und einfach erweiterbar sind?

Quellenverzeichnis

- [1] Larman, C.: UML 2 und Patterns angewendet, mitp Professional, 2005
- [2] Seidel, M. et al.: UML @ Classroom: Eine Einführung in die objektorientierte Modellierung, dpunkt.verlag, 2012
- [3] Martin, R. C.: Clean Architecture: A Craftsman's Guide to Software Structure and Design, mitp Professional, 2018