Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering
InIT Institut für angewandte
Informationstechnologie

Bachelor of Science (BSc) in Informatik

Modul Software-Entwicklung 1 (SWEN1)

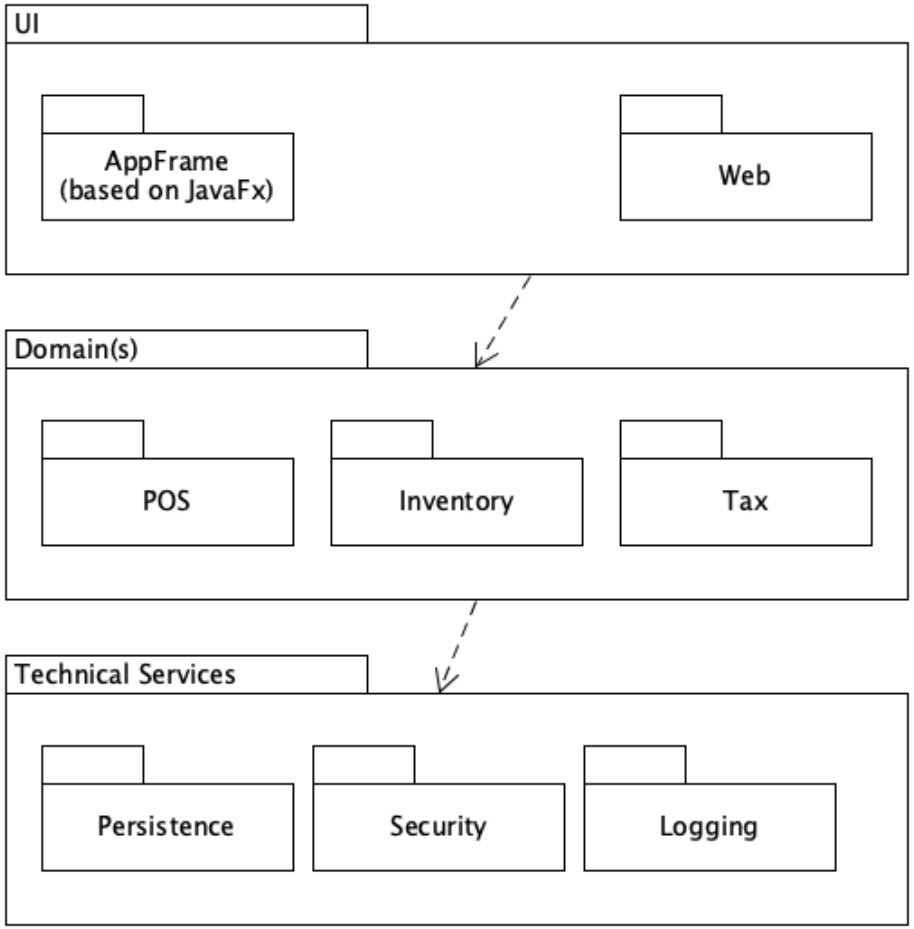
LE 05 – Softwarearchitektur und Design I

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

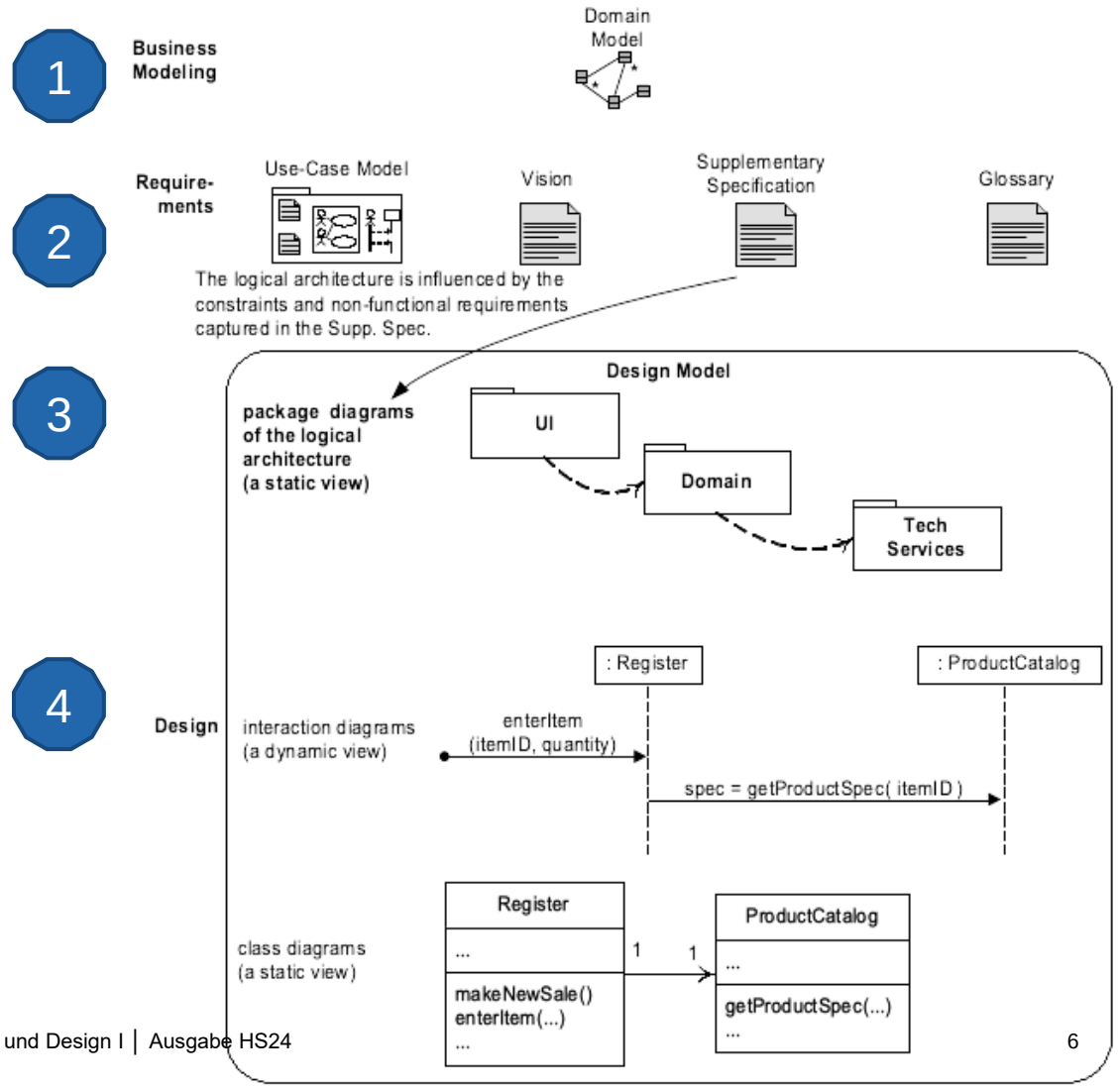
Was ist Softwarearchitektur?

- 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

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

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

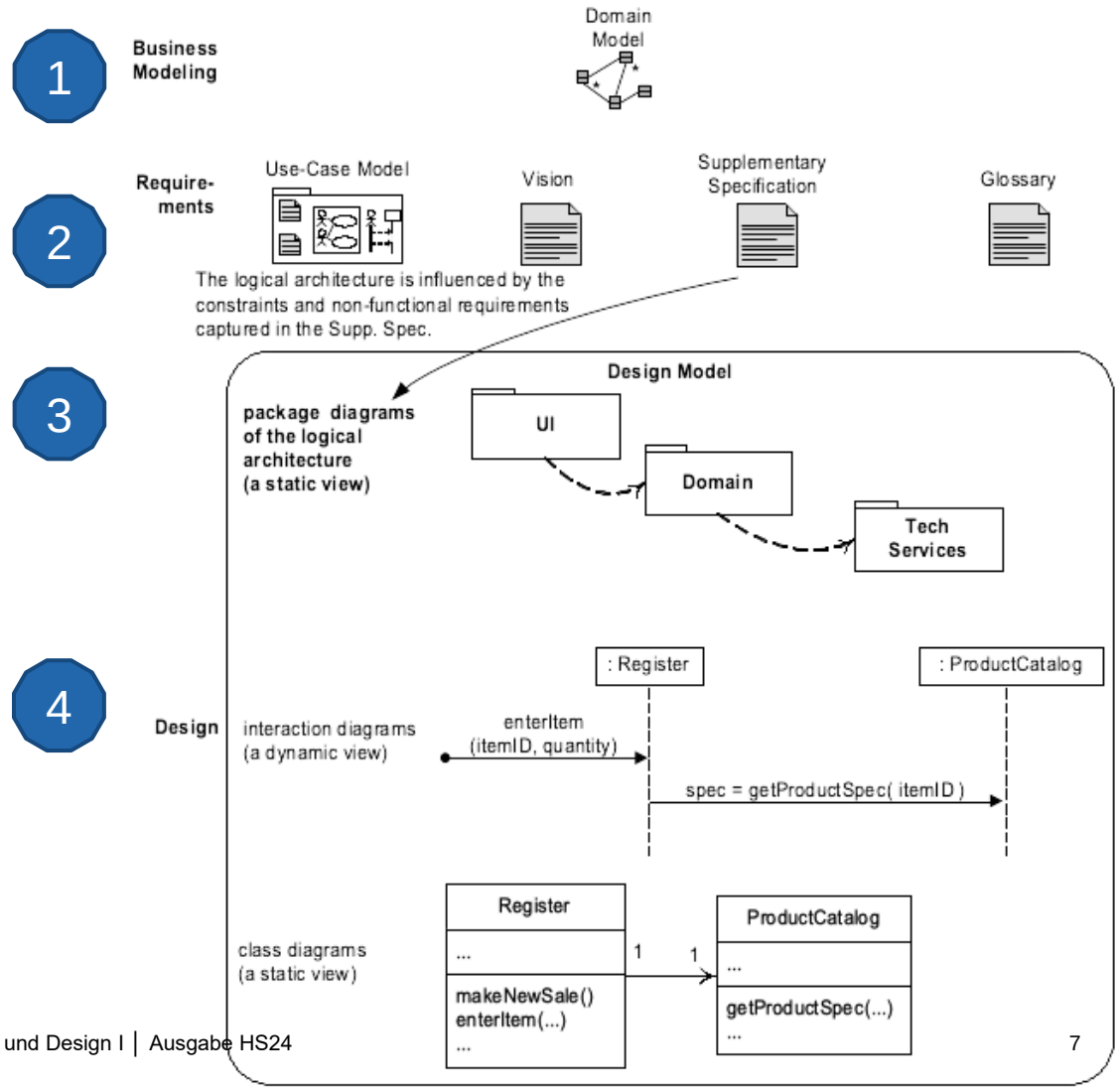


Übersicht Business Analyse vs. Architektur vs. Entwicklung

3 Logische Architektur (Software Architekt)

4 Umsetzung (Entwicklung)

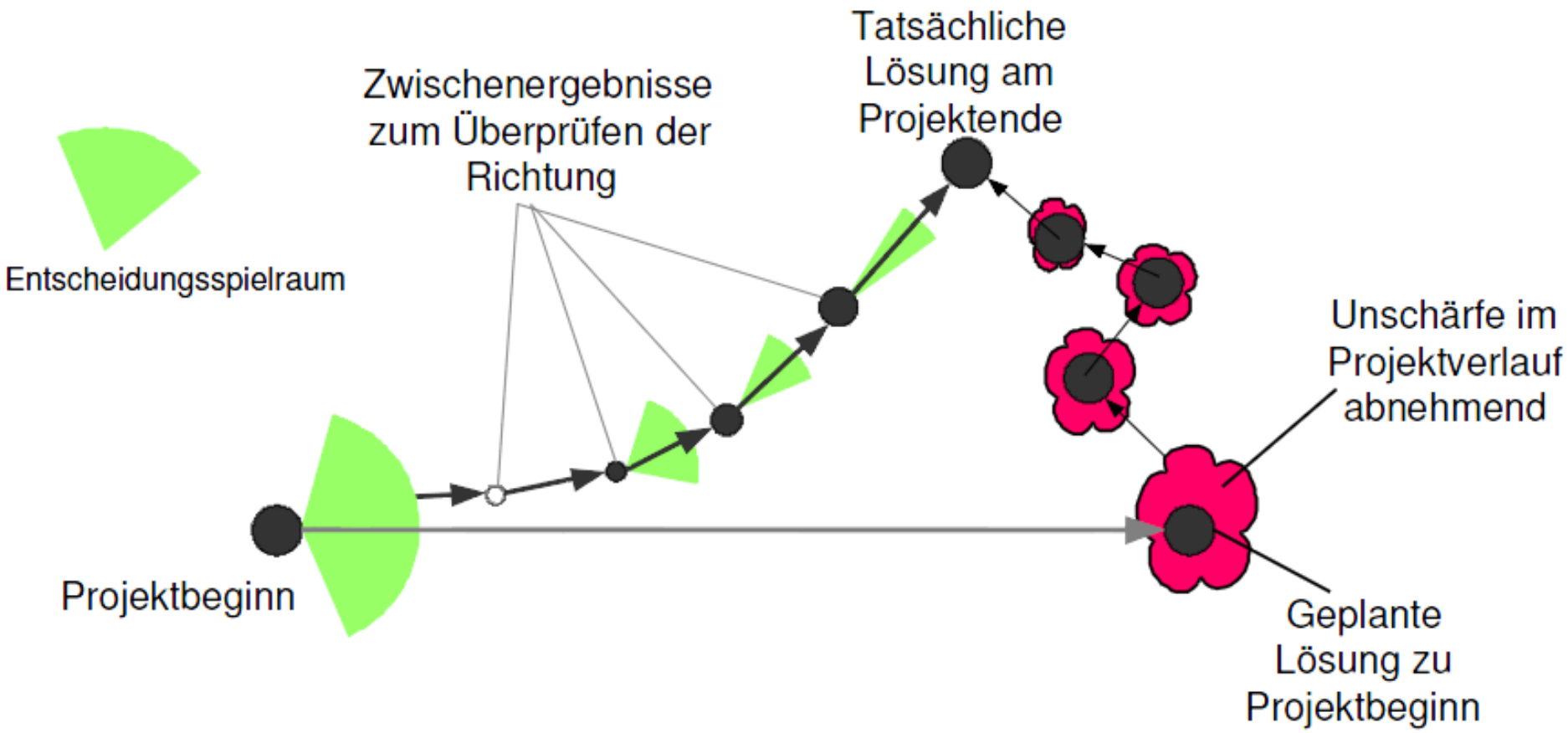
- Use Case / User Story Realisierung
- Anwendung von GRASP
- DCD – Design-Klassen-Diagramm
- Interaktionsdiagramme
- Programmierung
- Erstellen der Unit- / Integrations-Tests



Was ist eine Softwarearchitektur?

- Jede Software hat eine Architektur (bewusst oder unbewusst gewählt)
- Die Architektur definiert die tragenden Elemente der Software
- Die Softwarearchitektur beeinflusst den gesamten Systemlebenszyklus
  - Analyse, Entwurf und Implementierung
  - Betrieb und Weiterentwicklung
  - Entwicklungs- und Betriebsorganisation
- Eine gute Architektur unterstützt das Refactoring
- Eine gute Architektur ist Voraussetzung für eine erfolgreiche Softwareentwicklung

Wie entstehen Architekturen



Agenda

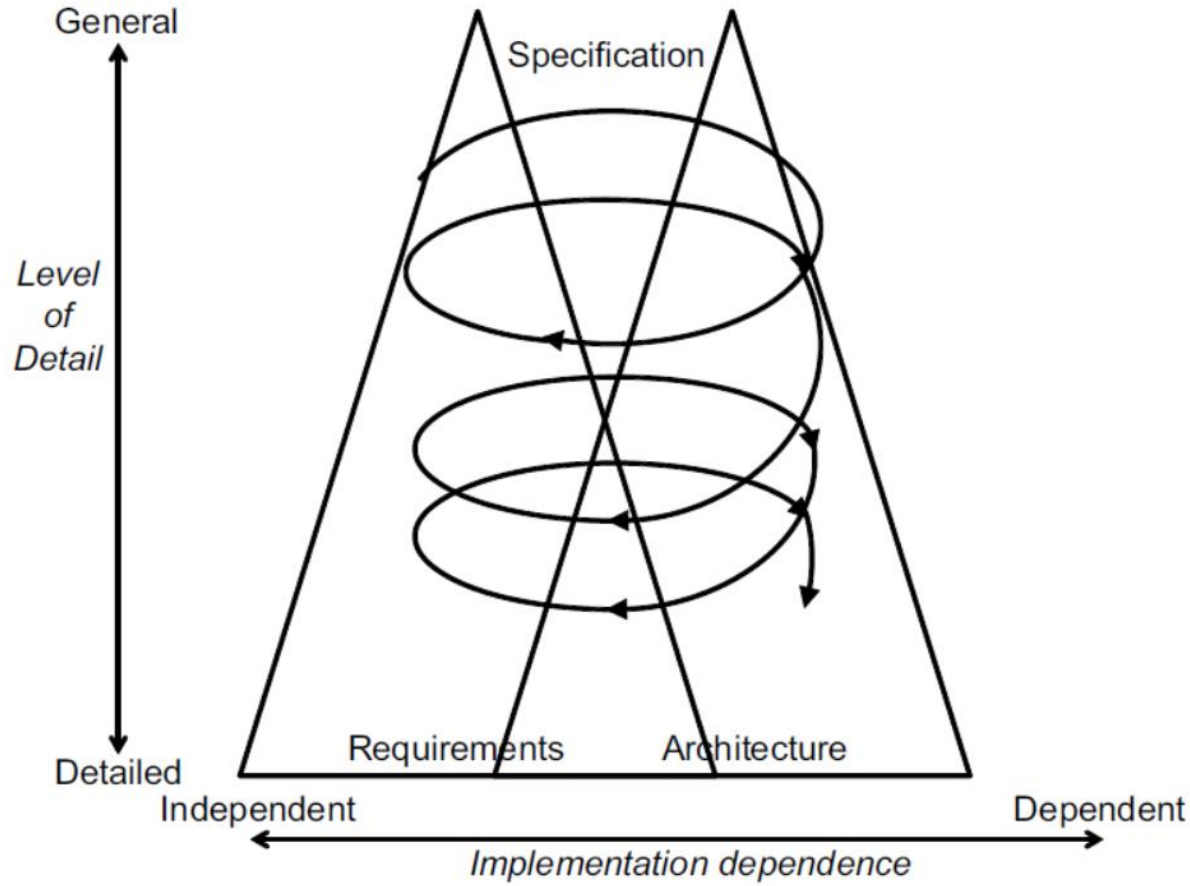
1. Was ist eine Softwarearchitektur
2. **Architektur aus den Anforderungen ableiten**
3. Modulkonzept
4. Architekturen beschreiben
5. UML-Paketdiagramme
6. Ausgewählte Architekturpatterns und Beispielarchitekturen
7. Aufgaben eines Software-Architekten
8. Wrap-up und Ausblick

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.

Anforderungen und Architektur: Twin Peak Model

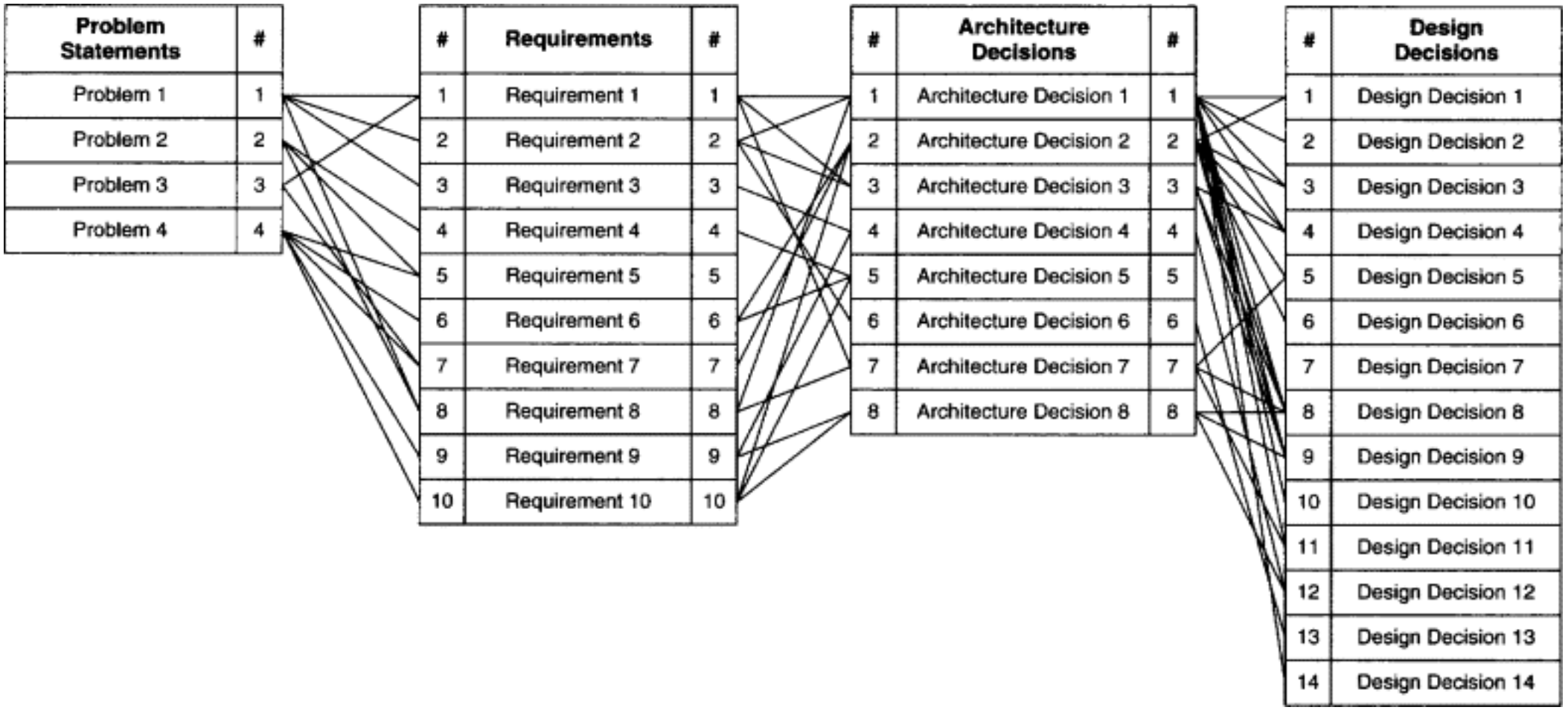
- Anforderungen (Requirements-Engineering)
  - Beeinflussen die Wahl und Ausgestaltung der Architektur
  - V.a. nichtfunktionale Anforderungen beeinflussen Architektur
- Gewählte Architektur
  - Muss Anforderungen erfüllen können
  - Hat Einfluss auf die Ausdetaillierung der Spezifikation der Anforderungen
    - Was bedeuten die Anforderungen bei einer bestimmten Wahl der Architektur



Anforderungen und Architektur

- Entwurfsentscheidungen sollten in erster Linie aus den Anforderungen abgeleitet werden
  - Mit welchen Architekturvarianten können die jetzigen und zukünftigen Anforderungen am besten erfüllt werden?
  - Was sind die Vor- und Nachteile der einzelnen Varianten
  - Wo gehen wir Kompromisse ein?
    - Es können ggf. nicht alle Anforderungen perfekt umgesetzt werden.
- Architekturentscheidungen und die Konsequenzen daraus müssen mit den Stakeholdern abgestimmt werden.

Anforderungen und Entwurfsentscheidungen



Problem Statements	#
Problem 1	1
Problem 2	2
Problem 3	3
Problem 4	4

#	Requirements	#
1	Requirement 1	1
2	Requirement 2	2
3	Requirement 3	3
4	Requirement 4	4
5	Requirement 5	5
6	Requirement 6	6
7	Requirement 7	7
8	Requirement 8	8
9	Requirement 9	9
10	Requirement 10	10

#	Architecture Decisions	#
1	Architecture Decision 1	1
2	Architecture Decision 2	2
3	Architecture Decision 3	3
4	Architecture Decision 4	4
5	Architecture Decision 5	5
6	Architecture Decision 6	6
7	Architecture Decision 7	7
8	Architecture Decision 8	8

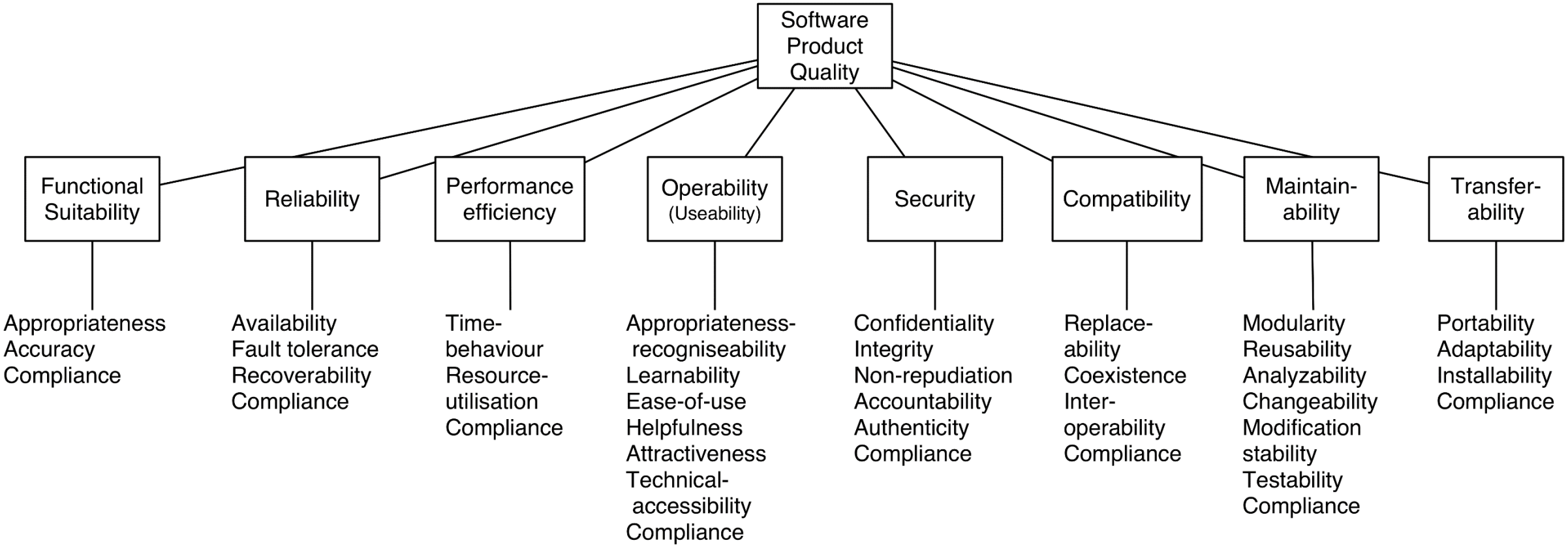
#	Design Decisions
1	Design Decision 1
2	Design Decision 2
3	Design Decision 3
4	Design Decision 4
5	Design Decision 5
6	Design Decision 6
7	Design Decision 7
8	Design Decision 8
9	Design Decision 9
10	Design Decision 10
11	Design Decision 11
12	Design Decision 12
13	Design Decision 13
14	Design Decision 14

Denkpause

Aufgabe 5.1 (5')

- Nennen Sie einige Entwurfsentscheidungen, die Sie in Ihrem PM3 Projekt gefällt haben. Was waren Ihre Gründe für diese Entscheidung?

Nichtfunktionale Anforderungen gemäss ISO 25010



Nichtfunktionale Anforderungen gemäss ISO 25010

- Hauptziel: Jede Anforderung muss so formuliert sein, dass sie gemessen werden kann (Akzeptanzkriterium).
- ISO 25010
  - nichtfunktionale Anforderungen in ISO 25010 sind hierarchisch strukturiert.
    - Hauptmerkmale, Untermerkmale und Metriken
  - Für jede nichtfunktionale Anforderung definiert ISO 25010 Metriken.
  - Jede Metrik beinhaltet eine Beschreibung der Anforderung, ein Messverfahren, um die Erfüllung der Anforderung zu prüfen und eine Hilfestellung für die Interpretation der Ergebnisse.
  - Damit können Anforderungen genauer und messbarer formuliert und später überprüft werden.
- Unterschied zu FURPS+:
  - ist ein Akronym (beschrieben in SWEN1 Anforderungsanalyse II) und keine Norm
  - Beinhaltet Functionality, Usability, Reliability, Performance, Supportability, + (und weitere Begriffe)

Denkpause

Aufgabe 5.2 (5’)

- Sie sind König eines grossen Landes und müssen die Verwaltung organisieren:

  - Steuern eintreiben
  - Infrastruktur bauen und unterhalten

- Wie gehen Sie vor?

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

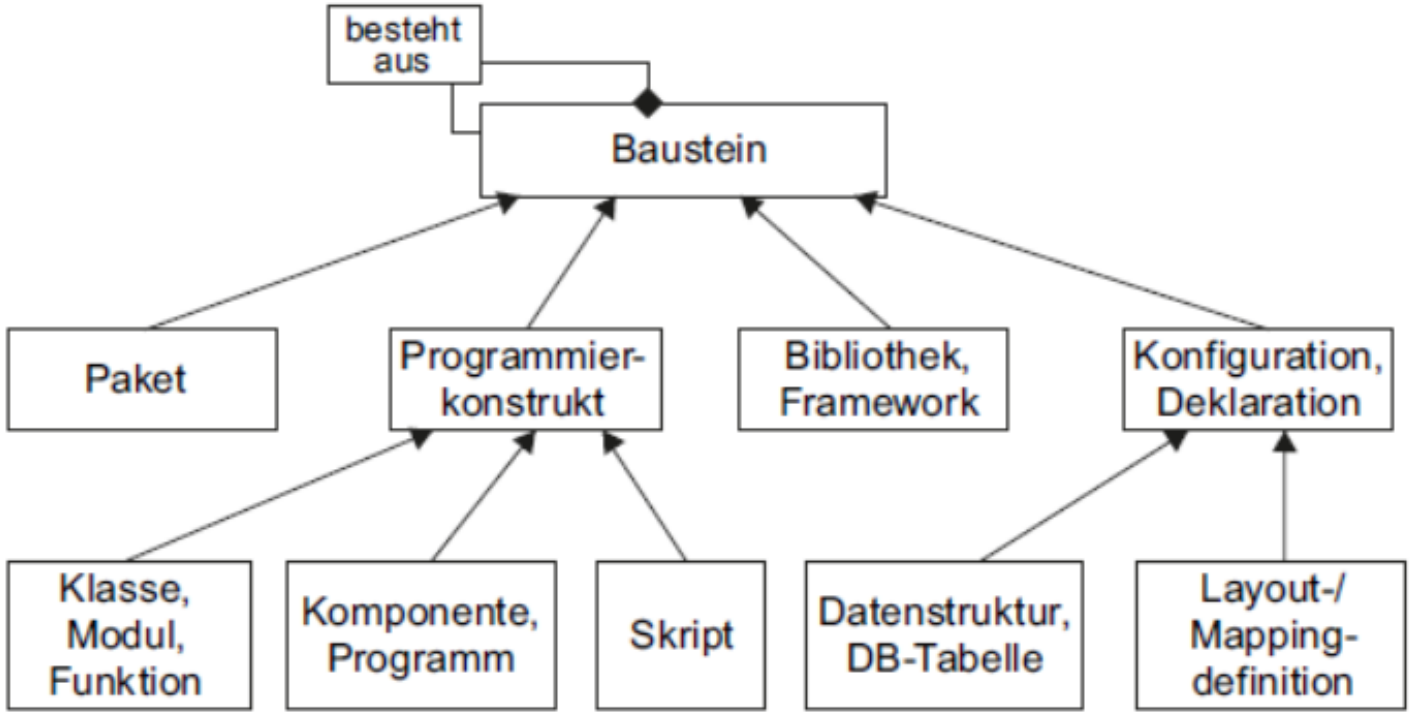
1. Was ist eine Softwarearchitektur
2. Architektur aus den Anforderungen ableiten
3. **Modulkonzept**
4. Architekturen beschreiben
5. UML-Paketdiagramme
6. Ausgewählte Architekturpatterns und Beispielarchitekturen
7. Aufgaben eines Software-Architekten
8. Wrap-up und Ausblick

Modulkonzept

- Erstmals vorgeschlagen in den 70er Jahren (D. Parnas)
- Modul (Baustein, Komponente):
  - Möglichst autarkes Teilsystem (wenig Kopplung nach aussen)
  - Hat eine klare minimale Schnittstelle gegen aussen
  - Software-Modul enthält alle Funktionen und Datenstrukturen, die es benötigt
  - Modul kann sein: Paket, Programmierkonstrukt, Library, Komponente, Service
- Modulkonzept wird in allen Ingenieurdisziplinen angewendet

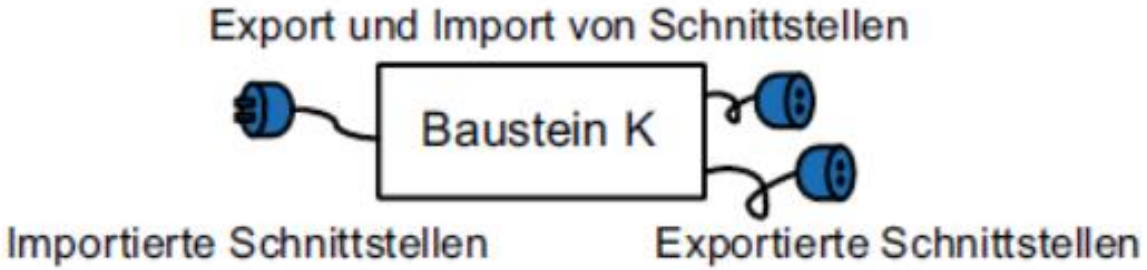
Bausteine und Schnittstellen

- 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



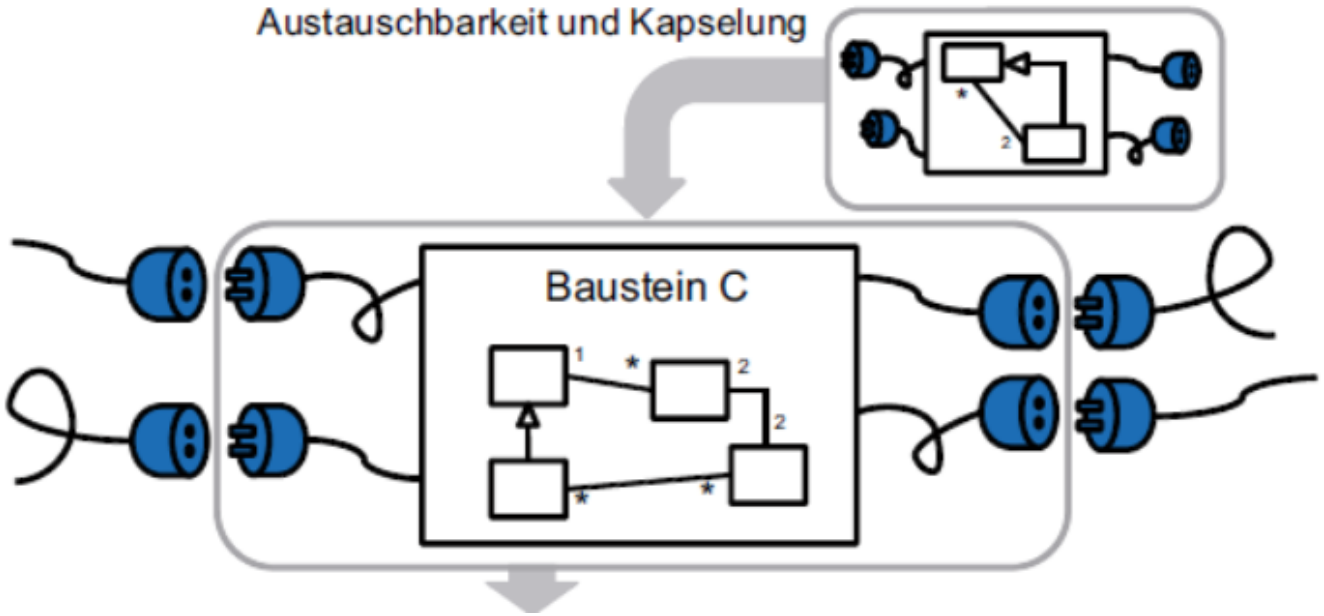
Schnittstellen (Interfaces)

- 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



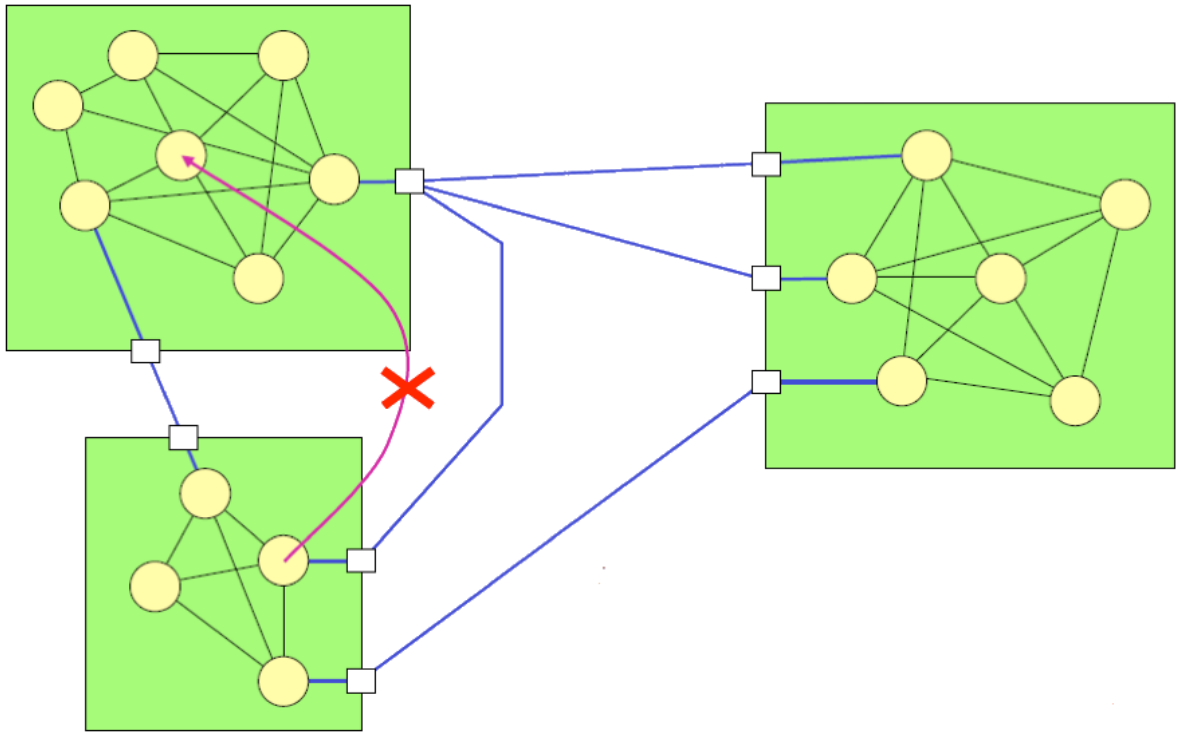
Bausteine und Schnittstellen

- 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

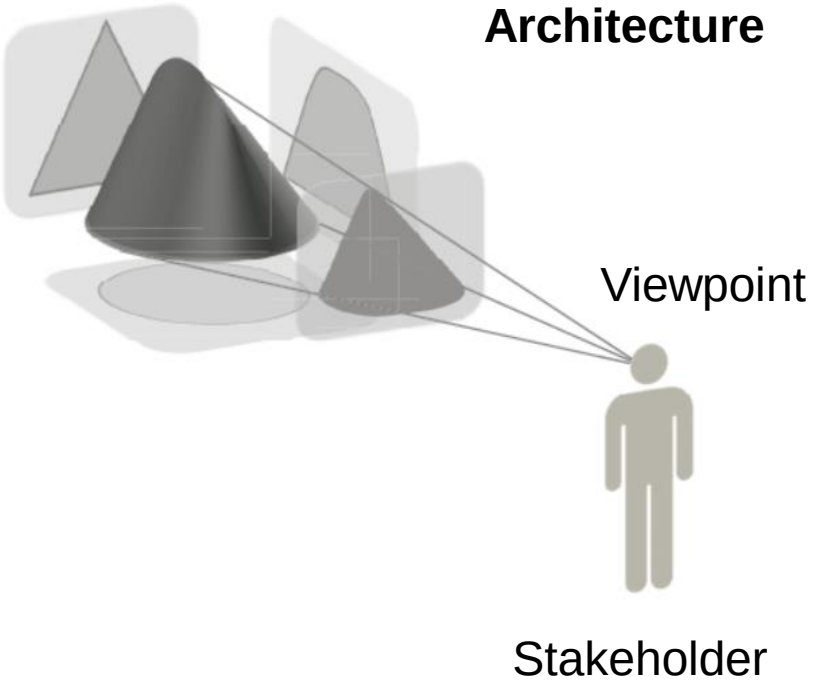
- Zwei charakteristische Masse: Kohäsion und Kopplung (-> GRASP LE 06)
- Kohäsion
  - Ein Mass für die Stärke des inneren Zusammenhangs
  - Je höher die Kohäsion innerhalb eines Moduls, desto besser die Modularisierung
    - schlecht: zufällig, zeitlich
    - gut: funktional, objektbezogen
- Kopplung – 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

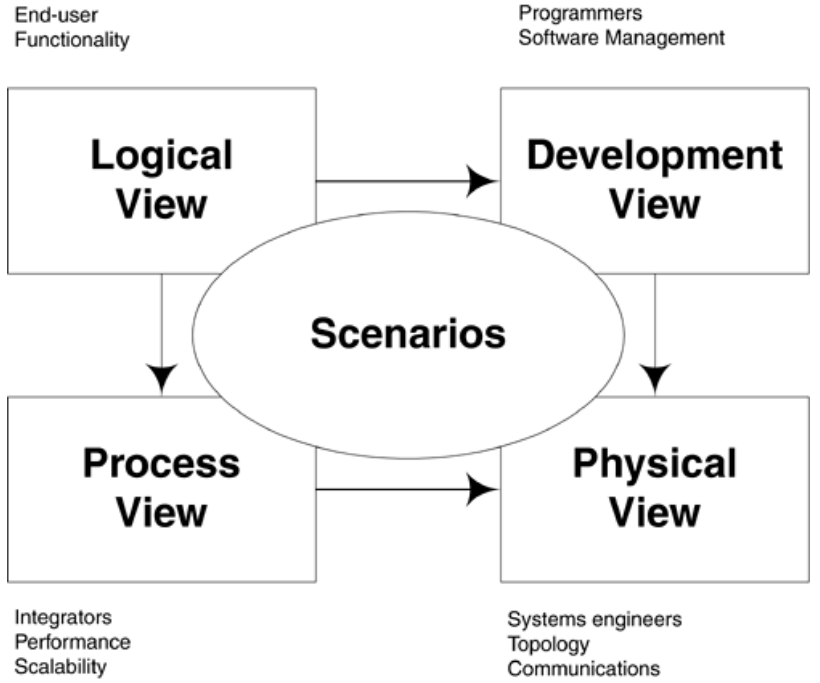
Architekturen beschreiben

- 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

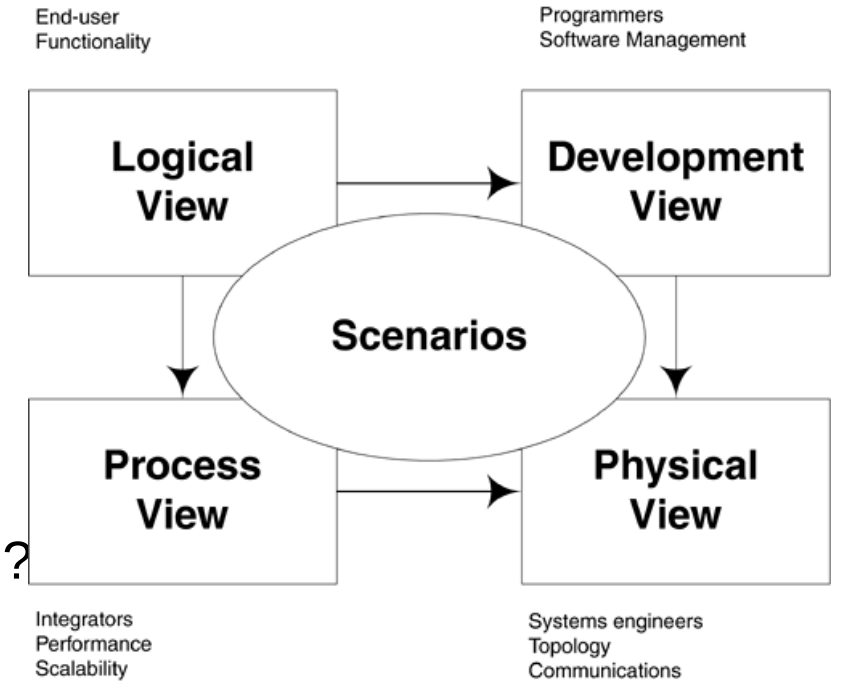
- Philippe Kruchten, 1995: 4 + 1 View Model
- **Logical View:**
  - Welche Funktionalität bietet das System gegen aussen an?
  - Wichtige Aspekte: Schichten, Subsysteme, Pakete, Frameworks, Klassen, Interfaces
  - UML: Systemsequenzdiagramme, Interaktionsdiagramme, Klassendiagramm, Zustandsdiagramme
- **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

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

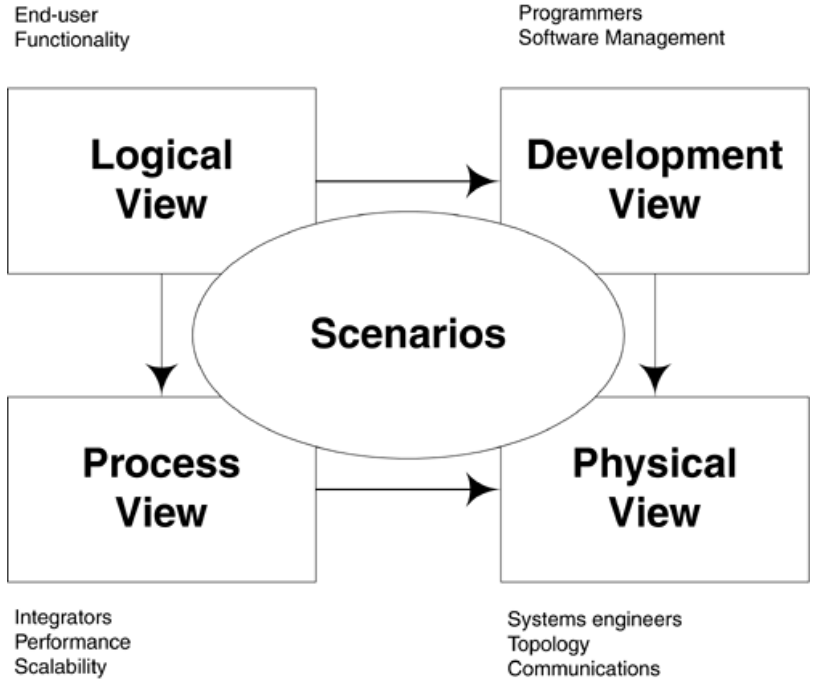


Philippe Kruchten, 1995:
http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

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

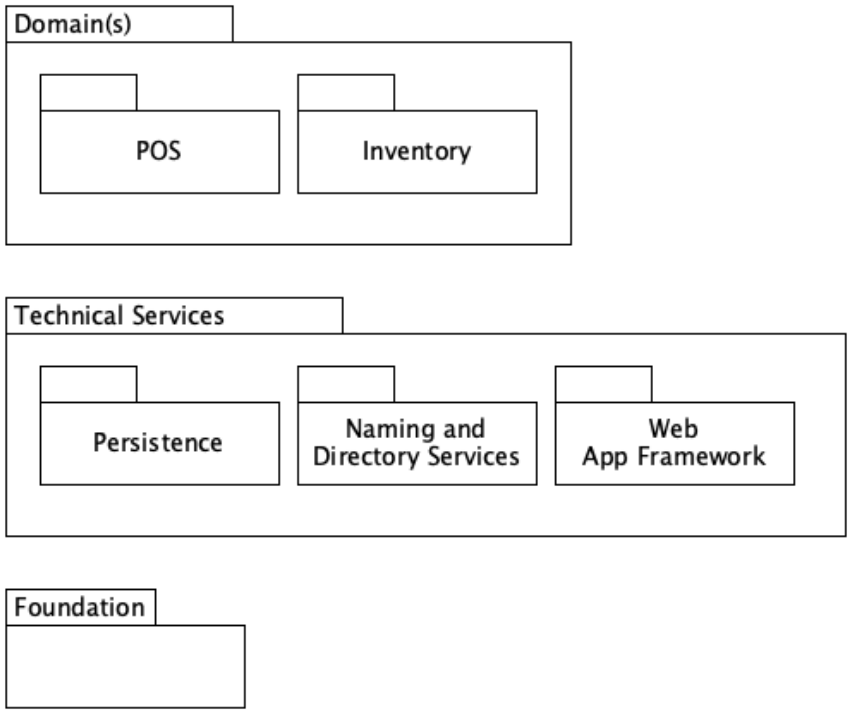
- **Weitere mögliche Views**
  - Daten-Sicht
  - Sicherheit



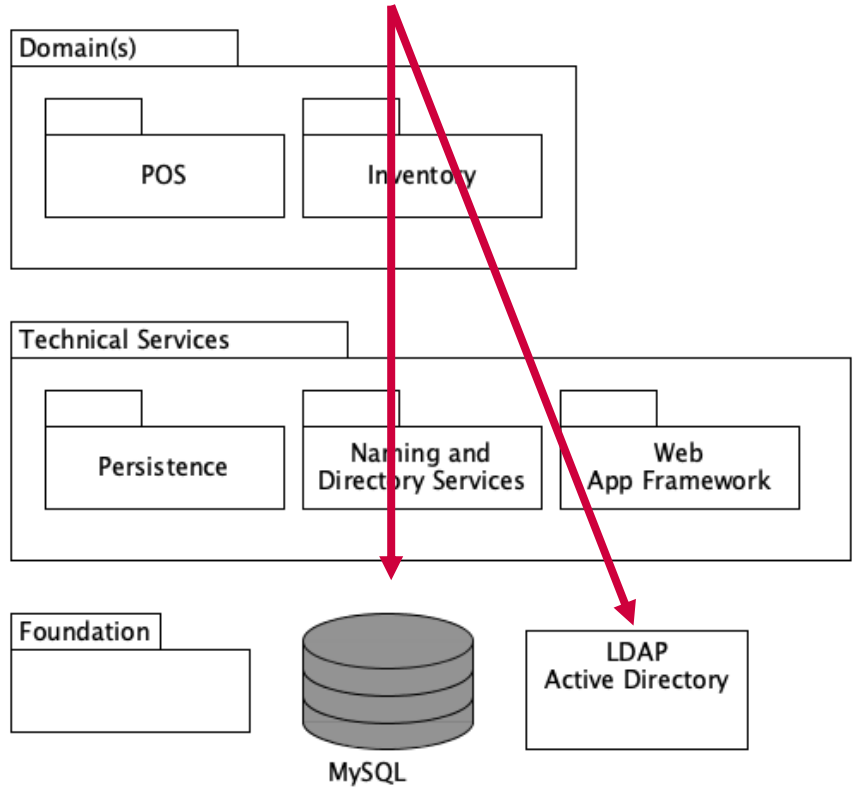
Philippe Kruchten, 1995: http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

Logische Architektur vs. Physikalische Architektur

- Logische Architektur
  - Zeigt die *logische* Strukturierung



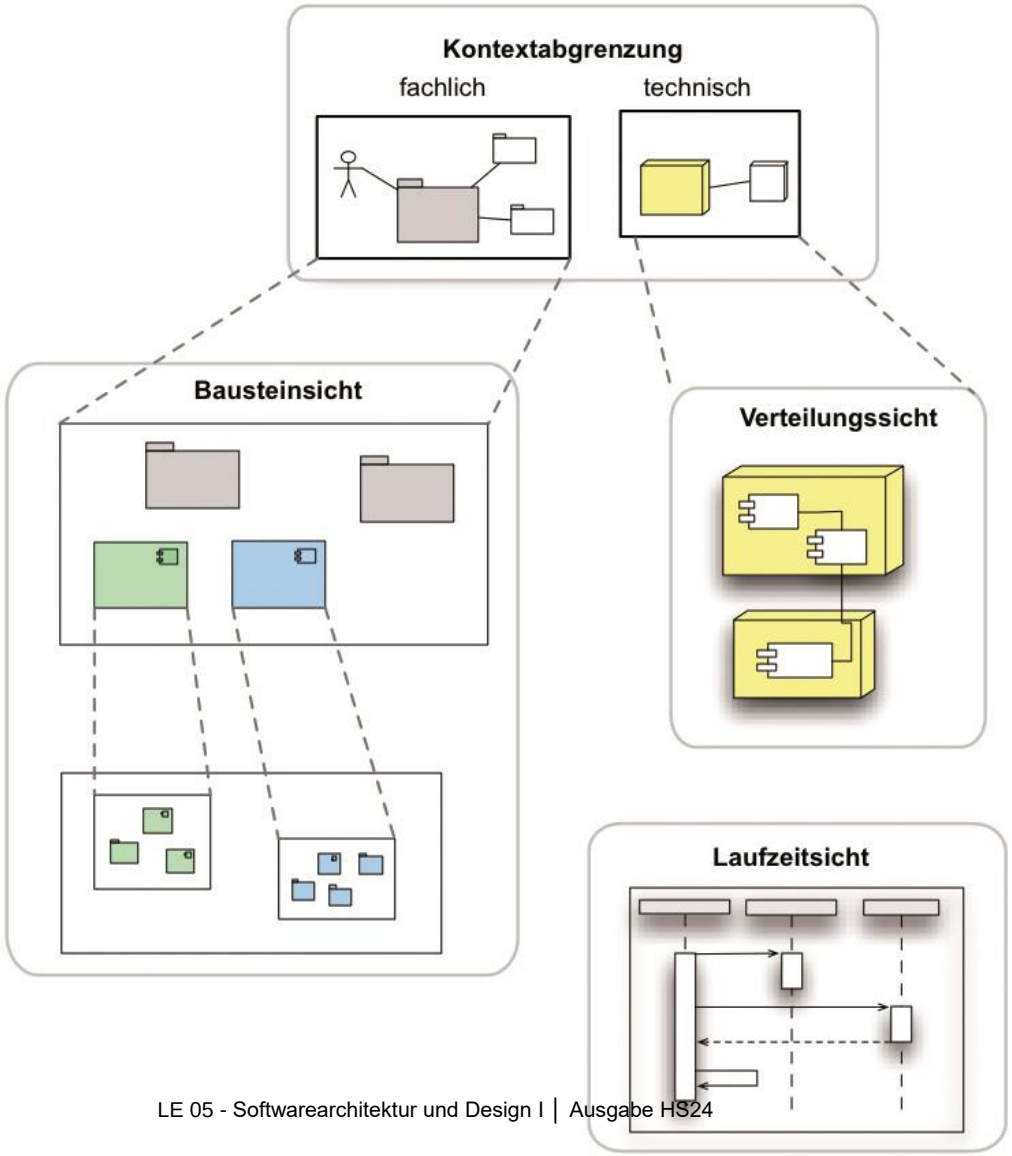
Mischung mit Deployment View vermeiden



Inhalt einer Architekturbeschreibung gemäss Arc42

Arc42 Architektur-Dokumentation
1. Einführung und Ziele
2. Randbedingungen
3. Kontextabgrenzung
4. Lösungsstrategie
5. Bausteinsicht (Modulsicht)
6. Laufzeitsicht
7. Verteilungssicht
8. Konzepte
9. Entwurfsentscheidungen
10. Qualitätsszenarien
11. Risiken und technische Schulden
12. Glossar

https://arc42.org/overview/



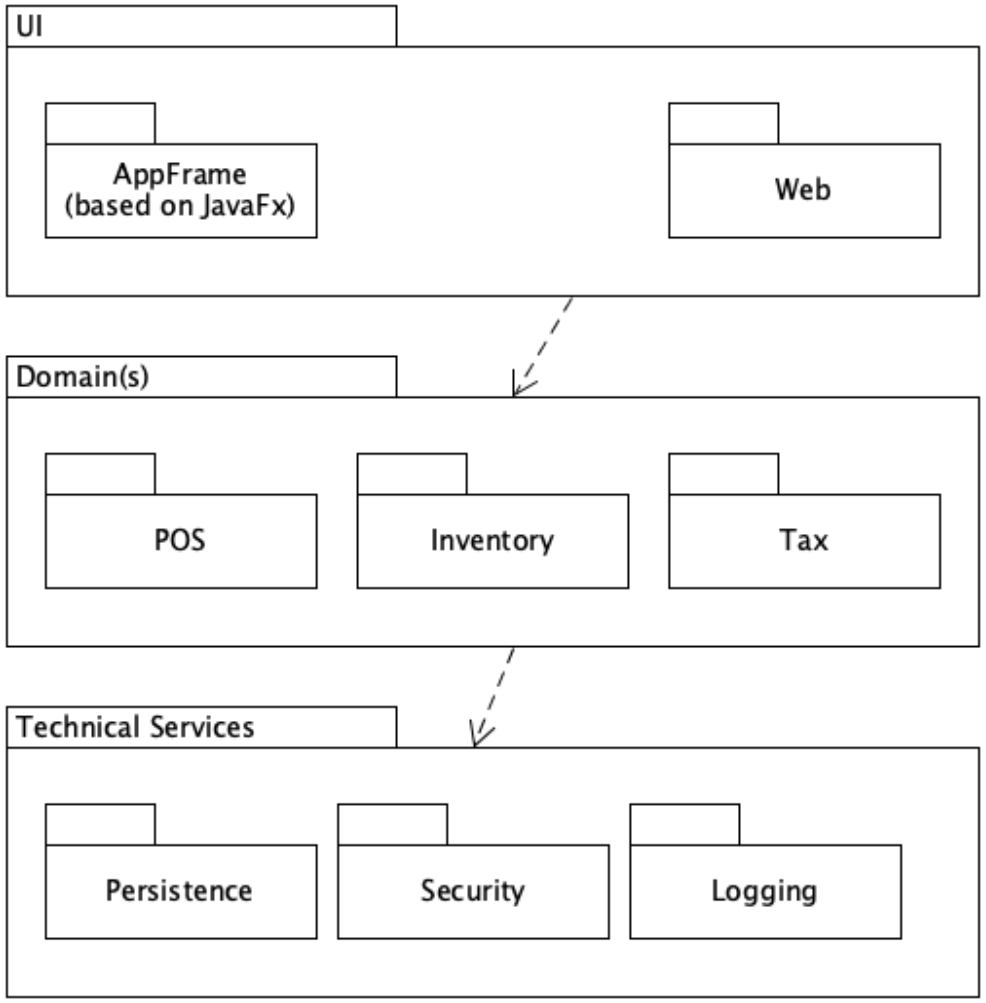
Weitere Informationen dazu in der Vorlesung ASE-Advanced Software Engineering (Wahlpflichtmodul)

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

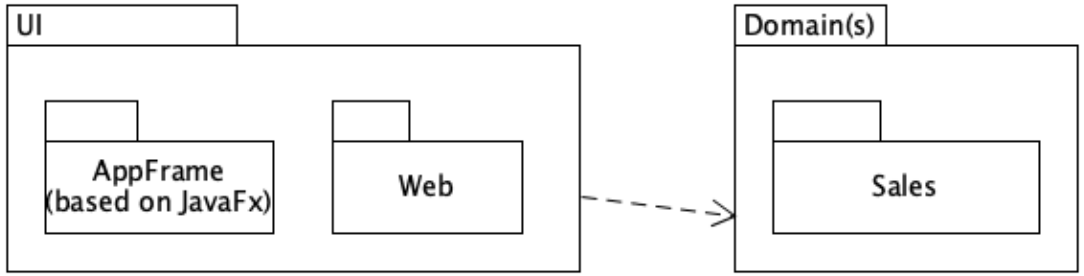
UML-Paketdiagramme

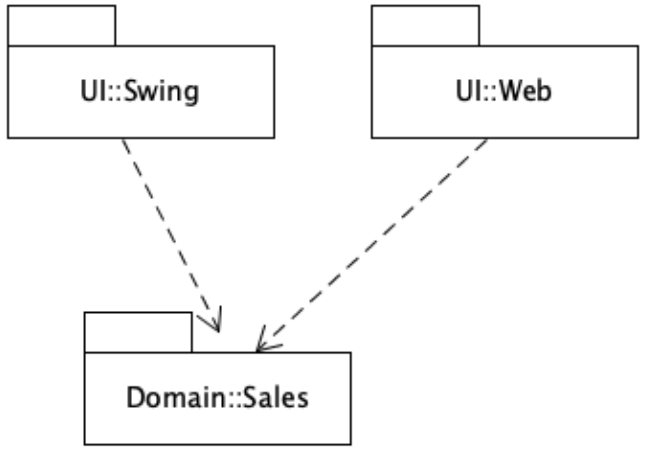
- UML-Paketdiagramme werden häufig zur Dokumentation der Architektur verwendet
  - Mittel, um Teilsysteme zu definieren
  - Mittel zur Gruppierung von Elementen
- Paket enthält Klassen und andere Pakete
  - Ähnlich, aber allgemeiner als Java Packages
- Abhängigkeiten zwischen Paketen



UML-Paketdiagramme: Abhängigkeiten modellieren

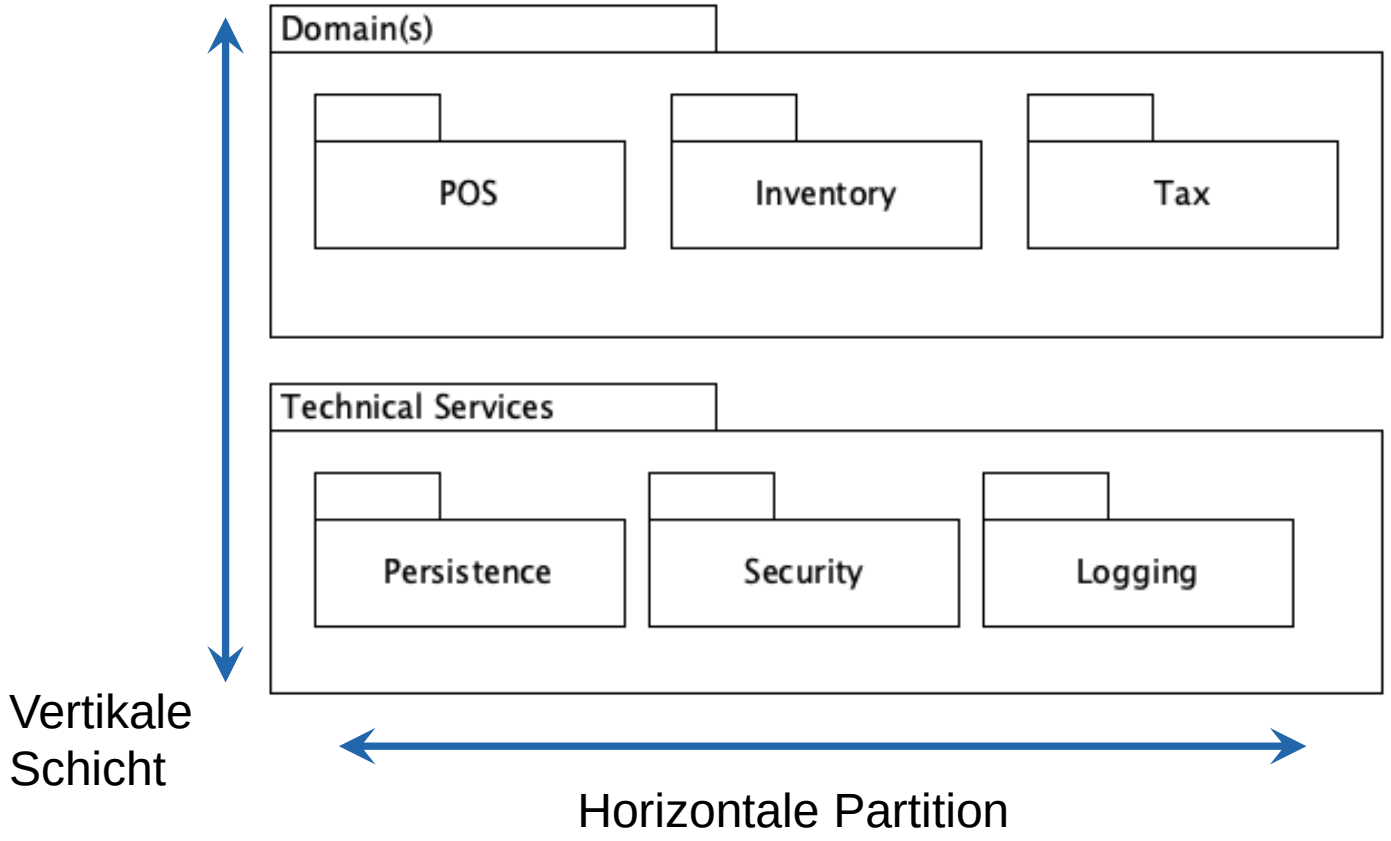
- Welche Abhängigkeiten existieren?
- Wer konsumiert welche Schnittstellen?





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

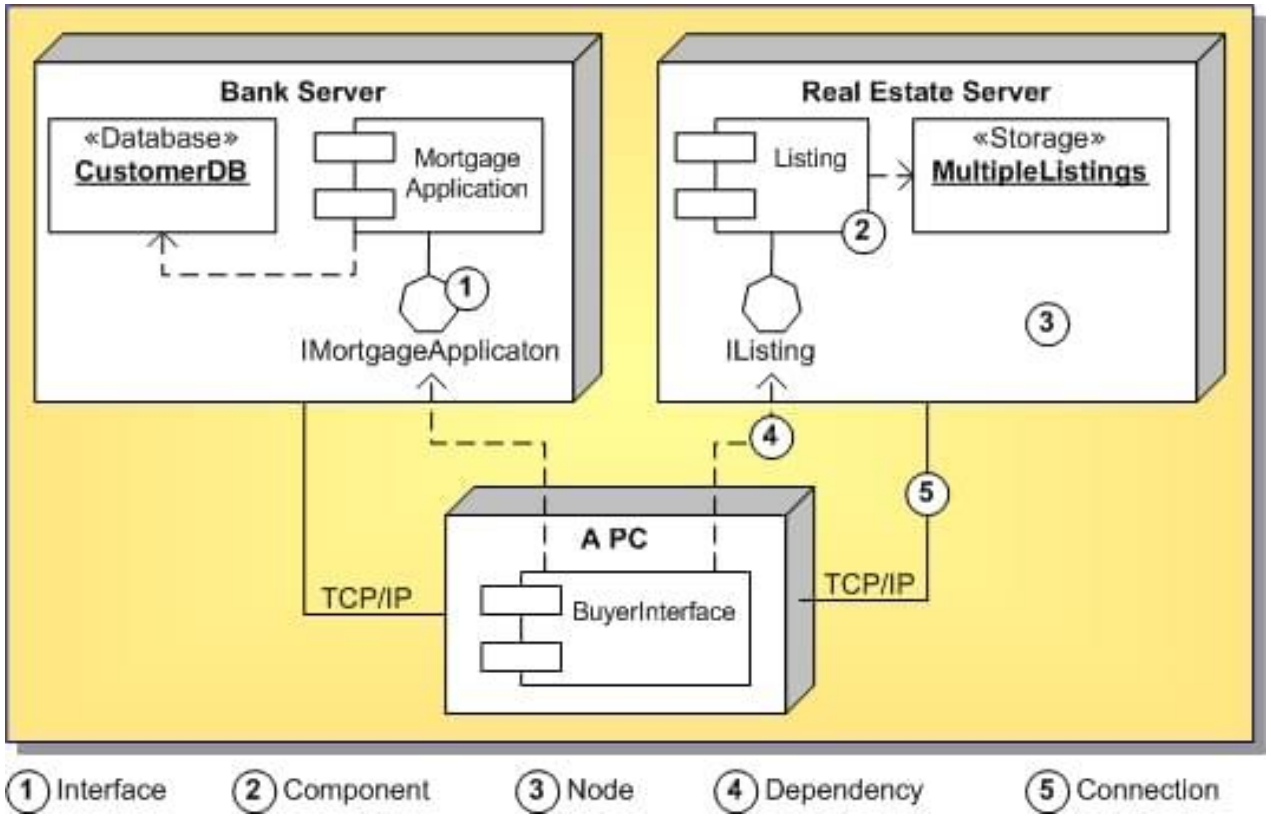
Denkpause

Aufgabe 5.3 (5')

1. Was ist der Vorteil einer guten und stabilen Schnittstelle zwischen UI und Anwendungslogik?
2. Bis zu welchen Schichten hinunter soll eine Firma selber Code schreiben?
3. Wie können untere Schichten ungefragt obere Schichten benachrichtigen?

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

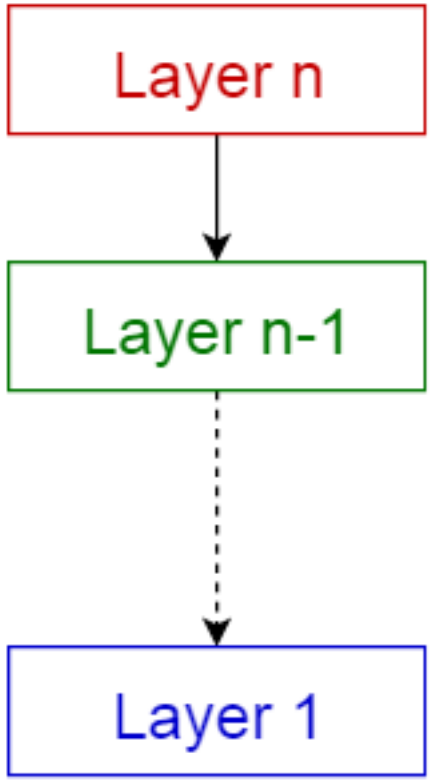
1. Was ist eine Software Architektur
2. Grundlagen für die Architektur aus den Anforderungen ableiten
3. Modulkonzept
4. Architekturen beschreiben
5. UML-Paketdiagramme
6. **Ausgewählte Architekturpatterns und Beispielarchitekturen**
7. Wrap-up und Ausblick

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

Schichtenkonzept (Layered Pattern) (1/3)

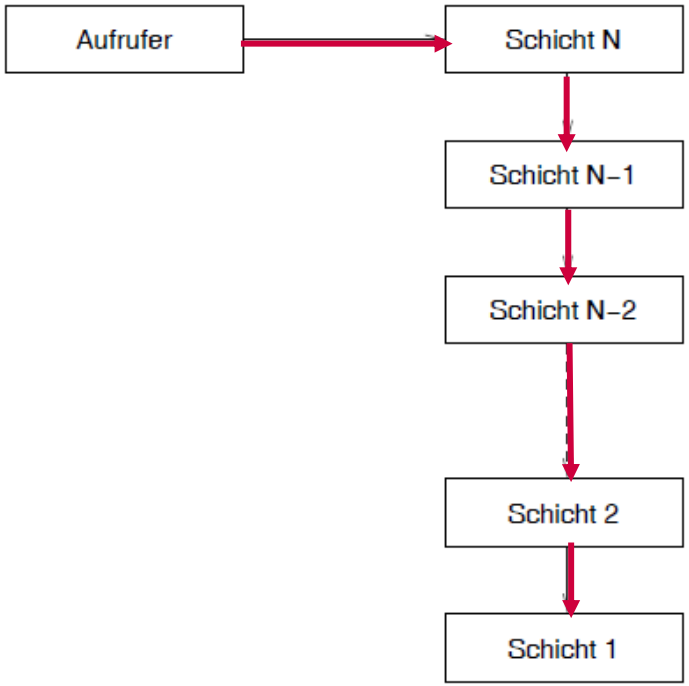
- Zerlegung des Gesamtsystems in Schichten
- Je weiter unten, desto allgemeiner
- Je höher, desto anwendungs-spezifischer
- Zuoberst ist das Benutzerinterface
- Kopplung nur von oben nach unten, NIE von unten nach oben



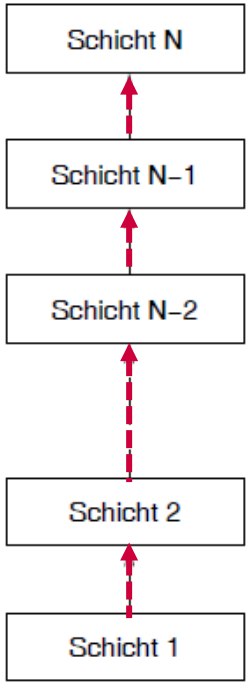
Schichtenkonzept (Layered Pattern) (2/3)

- Aufrufszenarien

höherer Schichten rufen Funktionalität in unteren Schichten direkt auf



untere Schicht benachrichtigt obere Schicht über Ereignis (Observer)

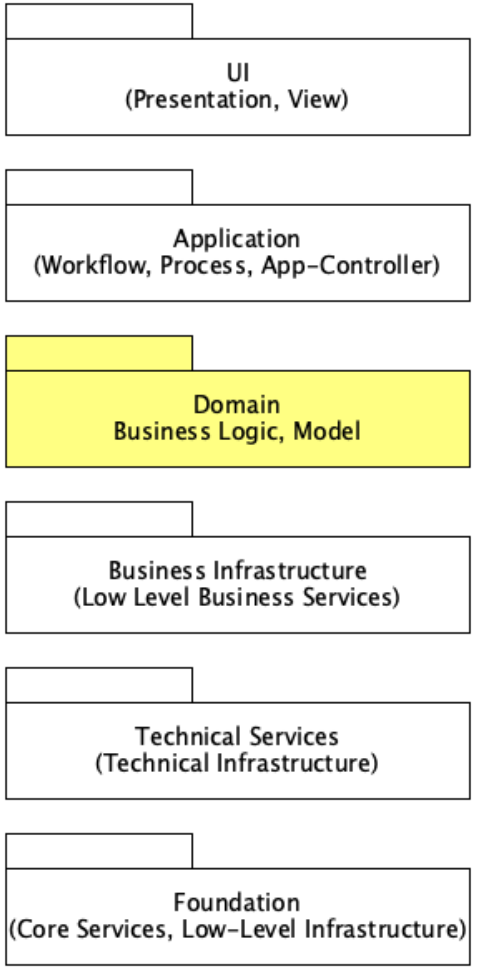


Kein direkter Aufruf!

Schichtenkonzept (Layered Pattern) (3/3)
Beispiel Enterprise Architektur

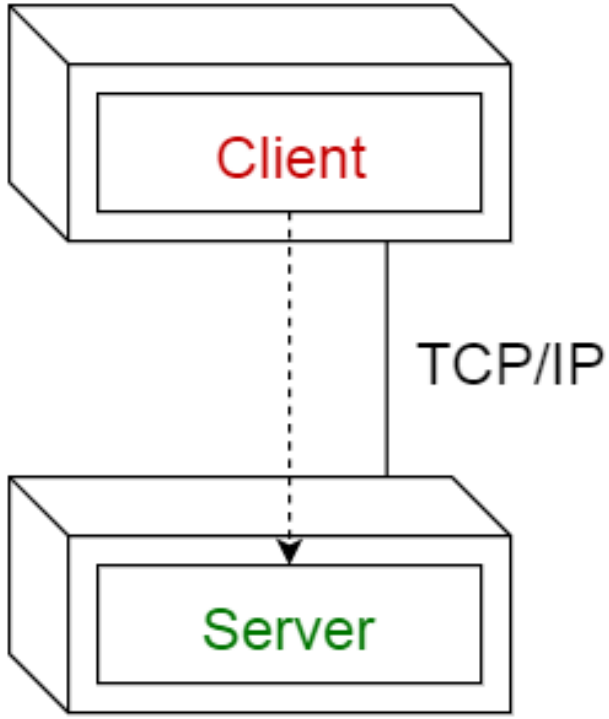
Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- UI
  - Presentation, Windows, Dialoge, Reports, WEB, Mobile
- Application
  - behandelt Requests von UI Layer, Workflow, Sessions
- Domain
  - behandelt Requests von Application Layer, Domain Rules und Services
- Business Infrastructure
  - Low Level Business Services, wie z.B. CurrencyConverter
- Technical Services
  - Persistence, Security, Logging
- Foundation
  - Datenstrukturen, Threads, Dateien, Network IO



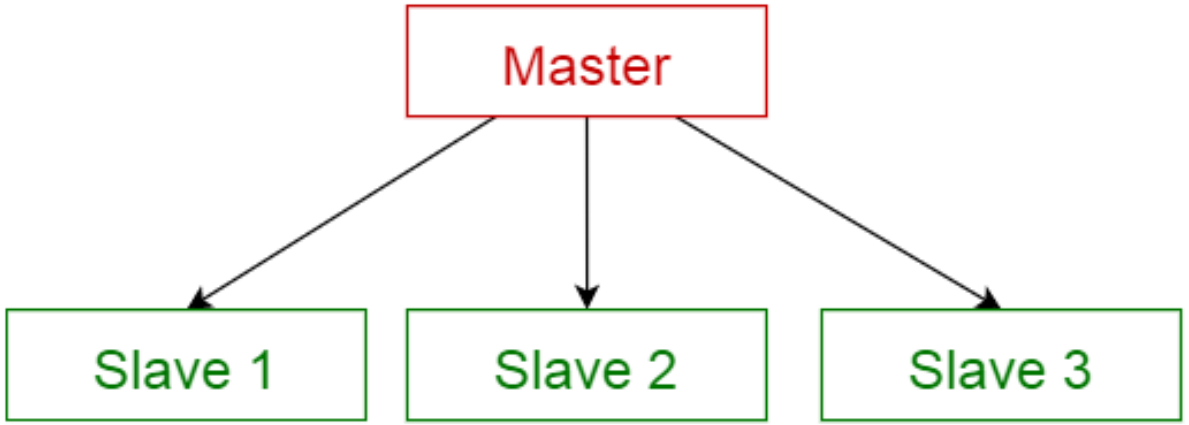
Client-Server

- Ein Server und mehrere Clients
- Ein Server stellt einen oder mehrere Services zur Verfügung
- Der Client macht eine Anfrage (Request) zum Server
- Der Server sendet eine Antwort (Response) zurück



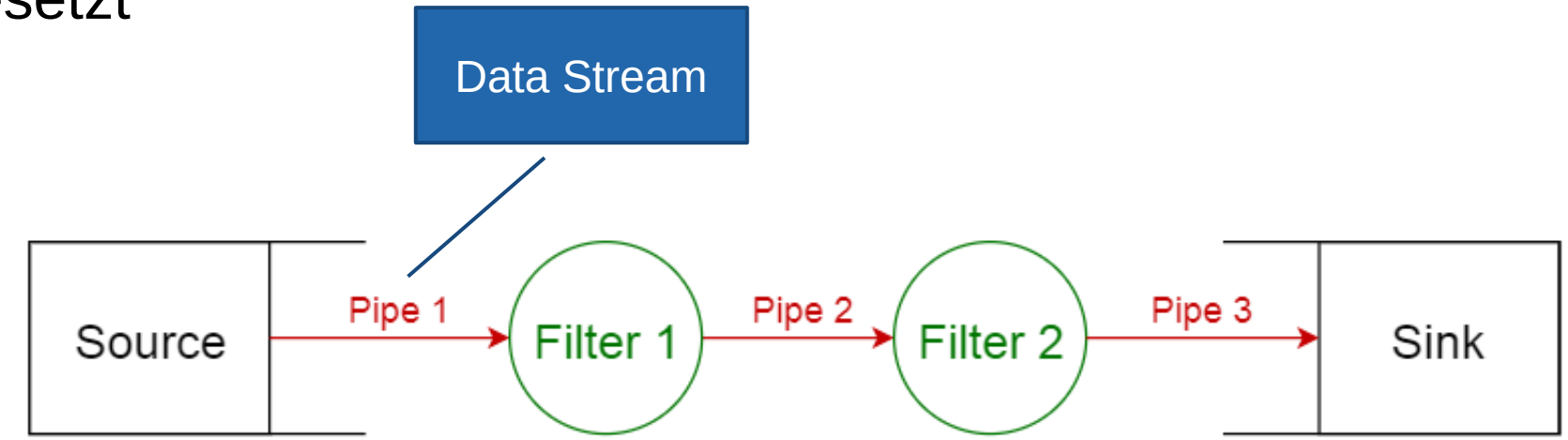
Master-Slave Pattern

- Der Master verteilt die Aufgaben auf mehrere Slaves
- Die Slaves führen die Berechnung aus und senden das Ergebnis zum Master
- Der Master berechnet das Endergebnis



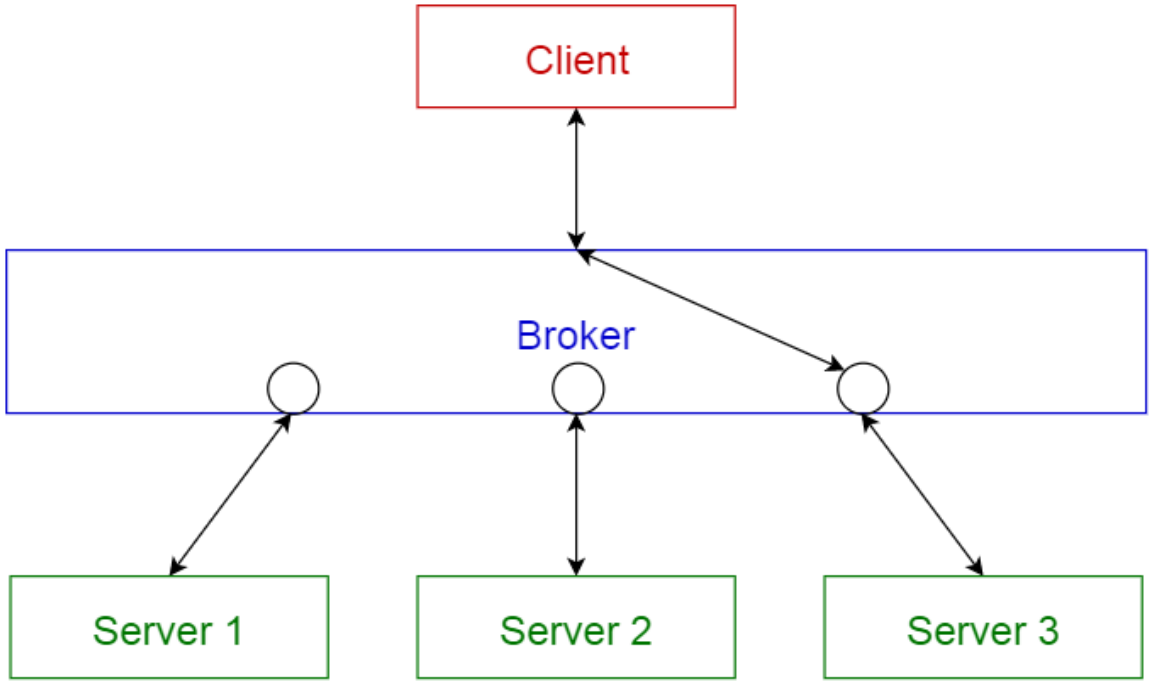
Pipe-Filter-Pattern

- Das Pattern kommt bei der Verarbeitung von Datenströmen zum Einsatz (Linux Pipe, RxJS Observable Streams, Java Streams, …)
- Jeder Verarbeitungsschritt wird durch einen Operator wie Filter, Mapper, etc. umgesetzt



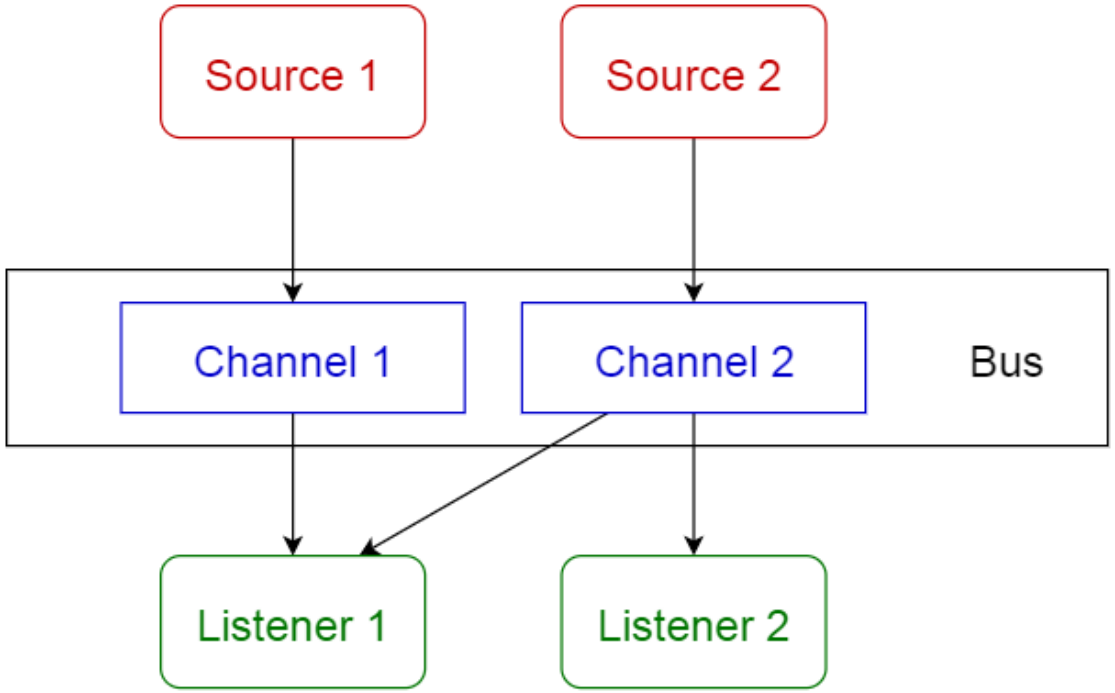
Broker Pattern

- Das Pattern wird eingesetzt, um verteilte Systeme mit entkoppelten Subsystemen zu koordinieren.
- Der Broker (Vermittler) vermittelt die Kommunikation zwischen einem Client und dem entsprechenden Subsystem
- Bsp.: Message Broker



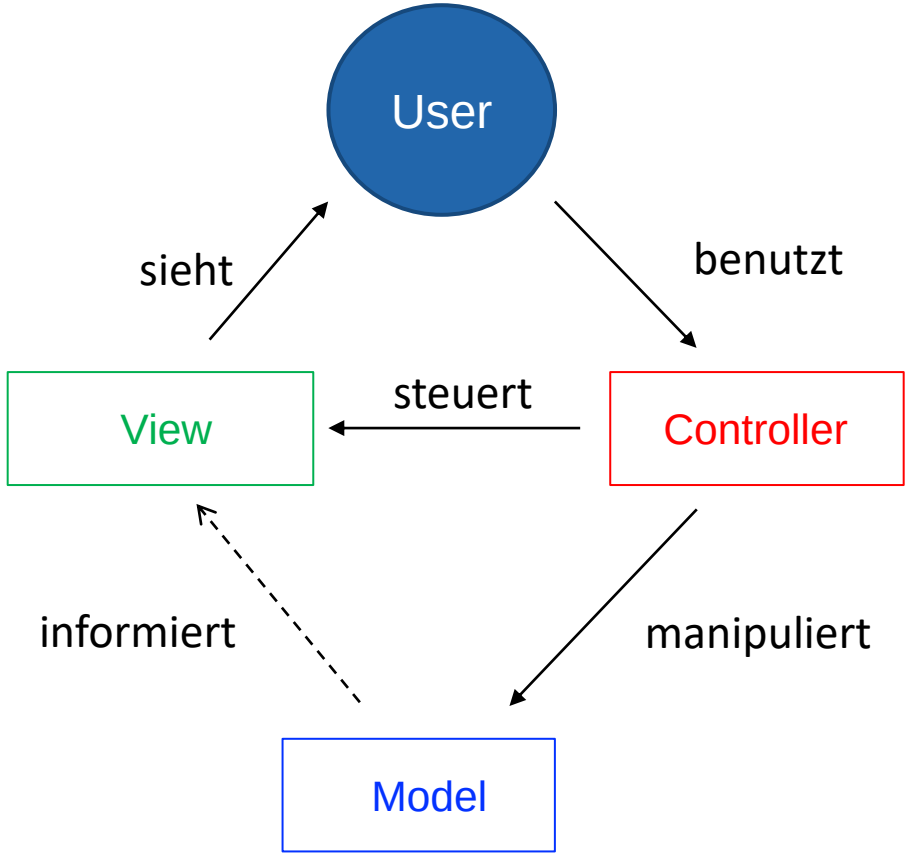
Event-Bus-Pattern

- Das Pattern umfasst vier Hauptkomponenten: EventSource, EventListener, Channel und Event Bus.
- Die Event Sources publizieren Meldungen zu einem bestimmten Kanal auf dem Event Bus
- EventListeners
  - Melden sich für bestimmte Events an
  - werden informiert, sobald sich entsprechende Meldungen auf dem Kanal befinden



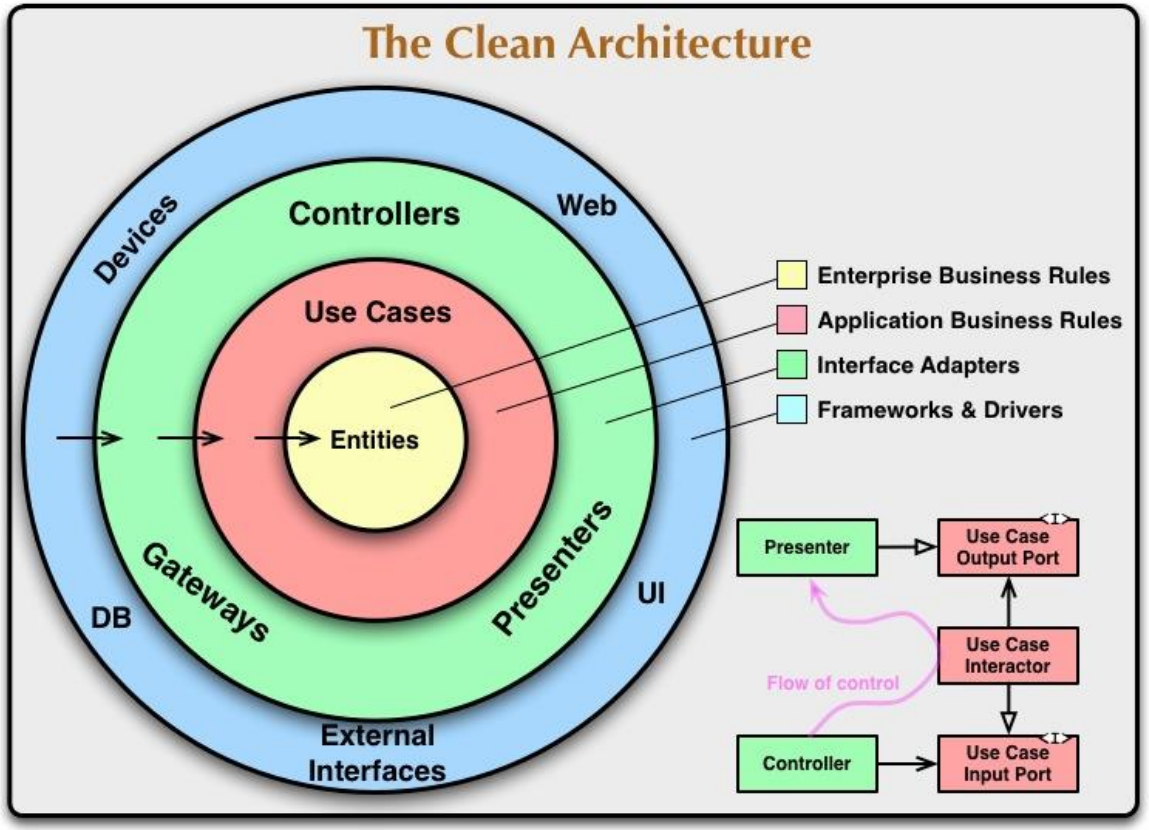
Model View Controller Pattern

- Eine interaktive Anwendung
  wird in drei Komponenten aufgeteilt:
  - **Model**: Daten und Logik,
  - **View**: Informationsanzeige
  - **Controller**: Verarbeitung der Benutzereingabe
- Bewirkt eine Entkopplung von UI und Logik
- Erlaubt Austauschbarkeit des Uis
- Alternativen
  - MVVM: Model View View Model
  - MVP: Model View Presenter
  - (mehr dazu in SWEN1-Vertiefung GUI-Architekturen)



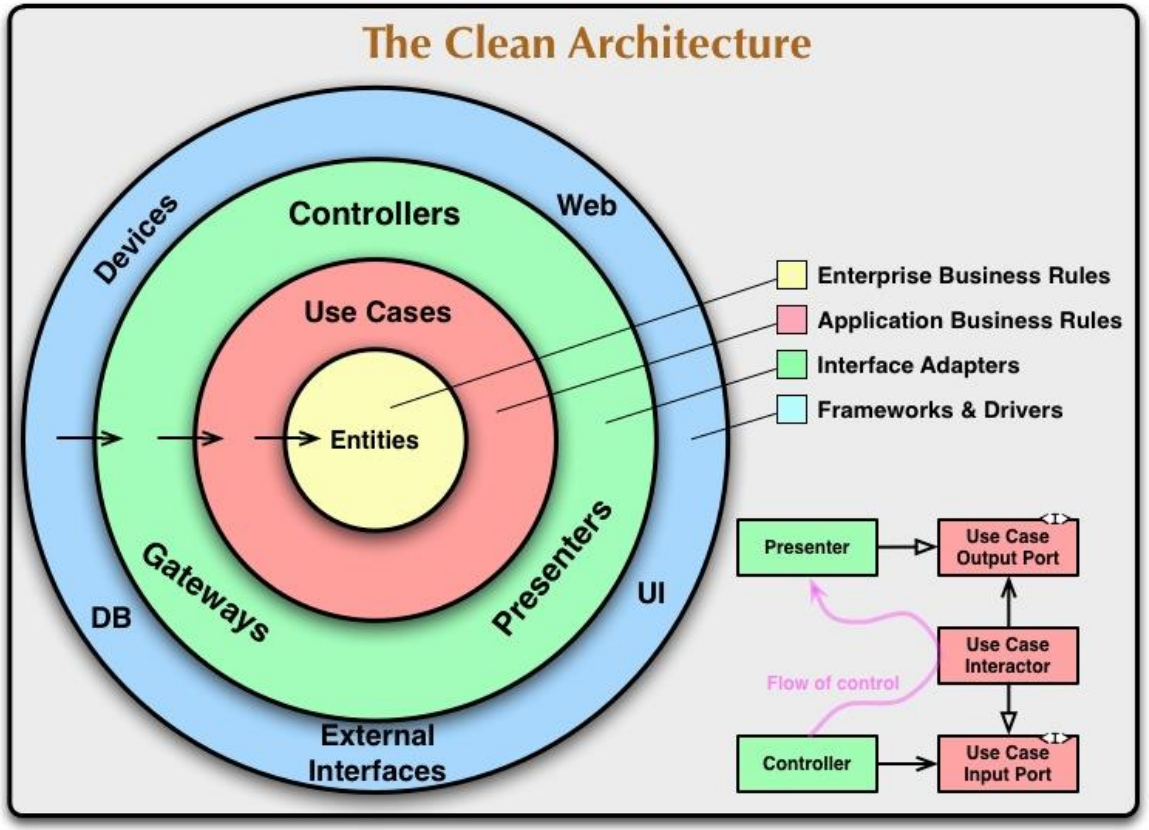
Clean Architecture (1/2)

- 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)

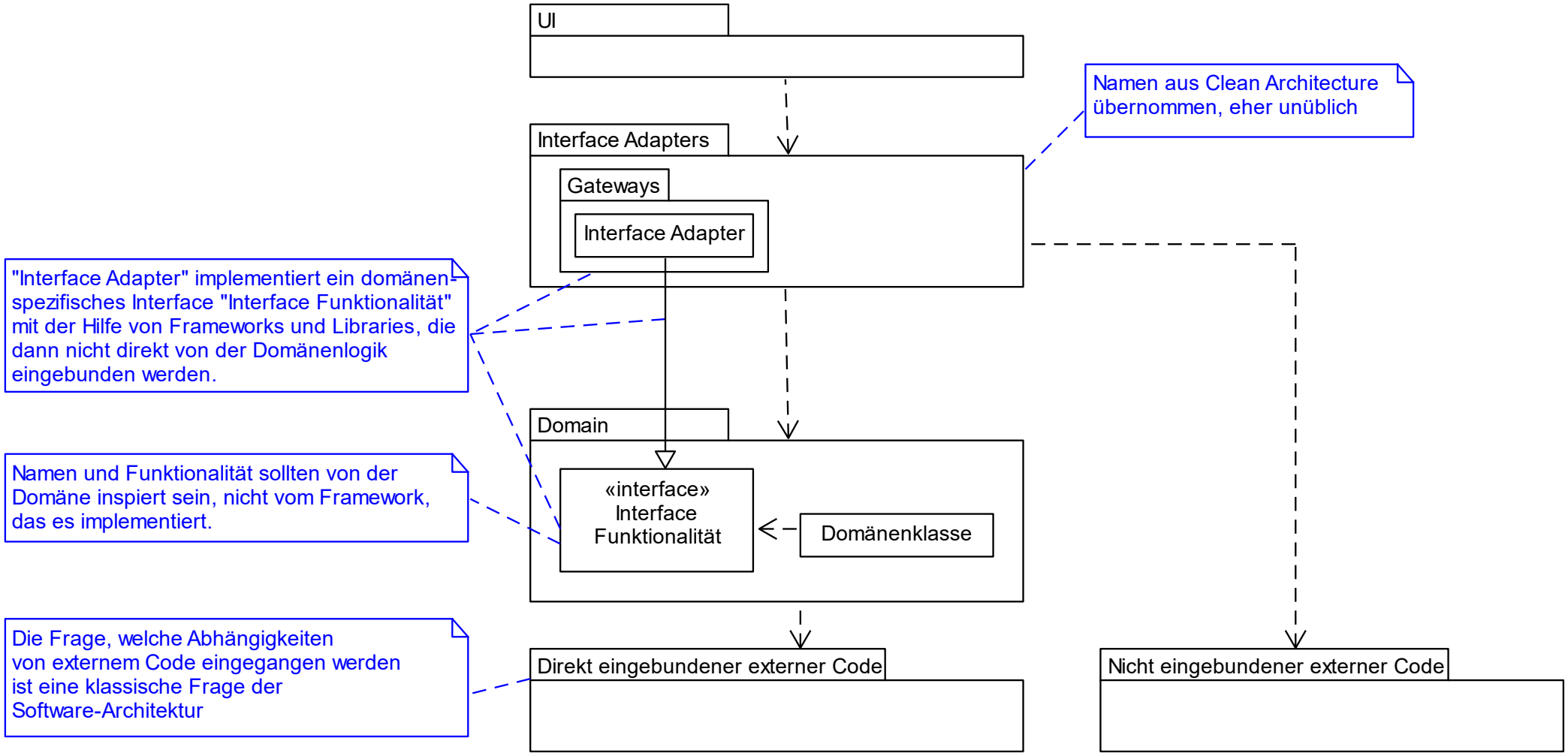
- Entities
  - Kapseln die Business Rules gültig für das gesamte Unternehmen
- Use Cases
  - Beinhalten die Business Rules einer Anwendung, orchestriert die Verwendung der Entities
- Interface Adapters
  - Adapter für die Konvertierung von Daten aus Datenbank oder Web
- Frameworks and Drivers



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