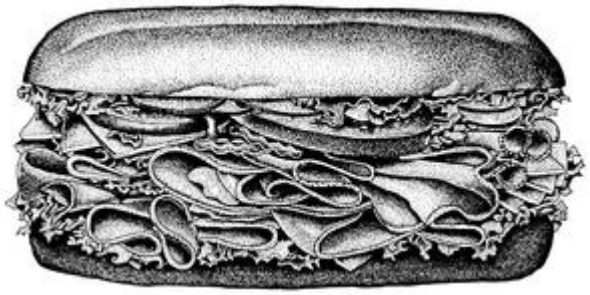
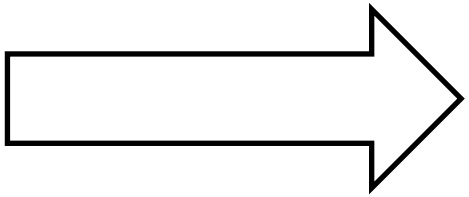
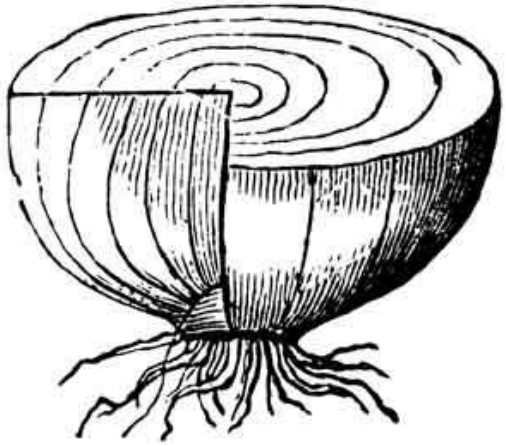
Onion Architecture

Alternative zur klassischen Schichtenarchitektur

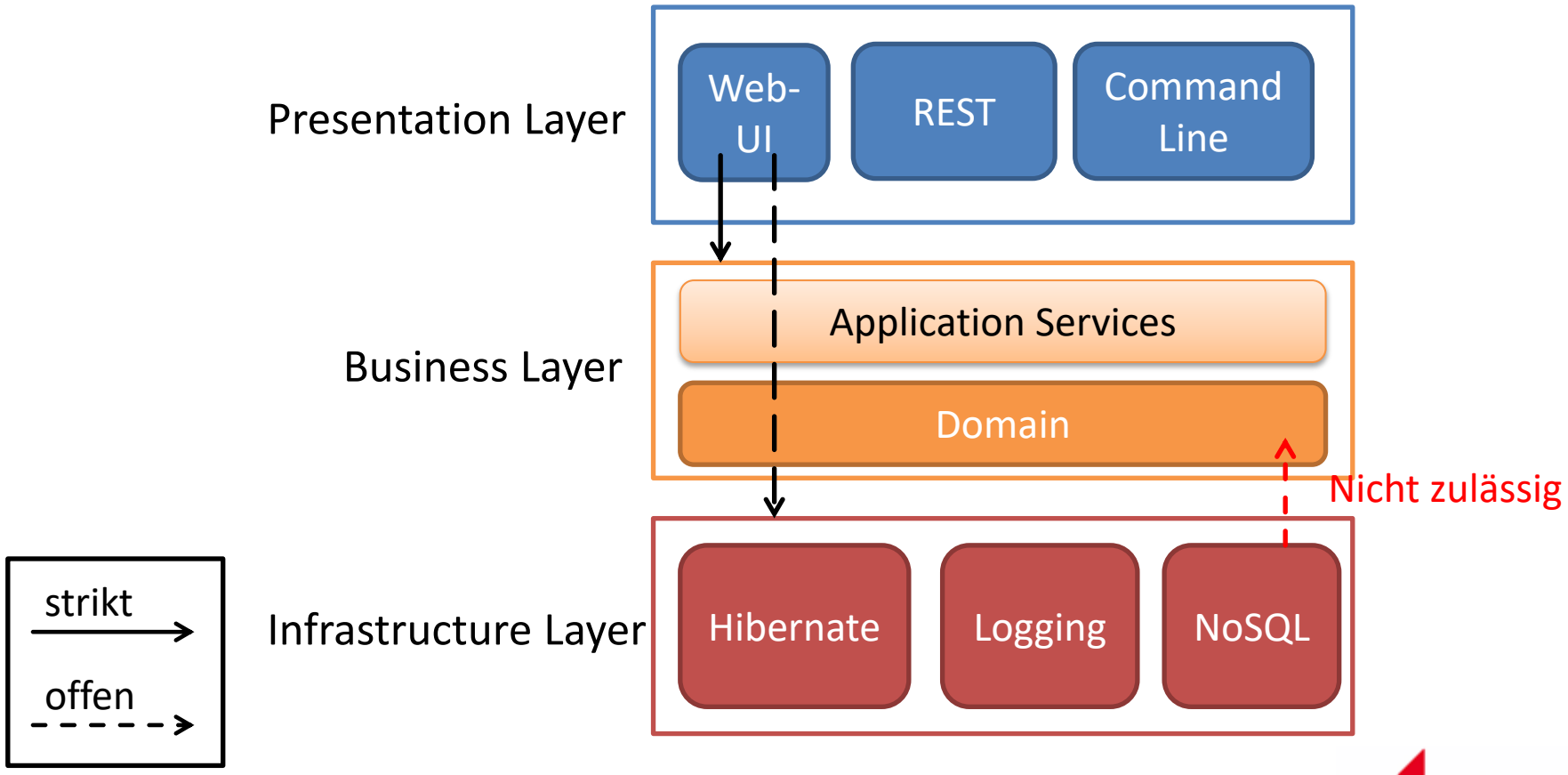






DHBW

Klassische Schichten-Architektur



Klassische Schichten-Architektur

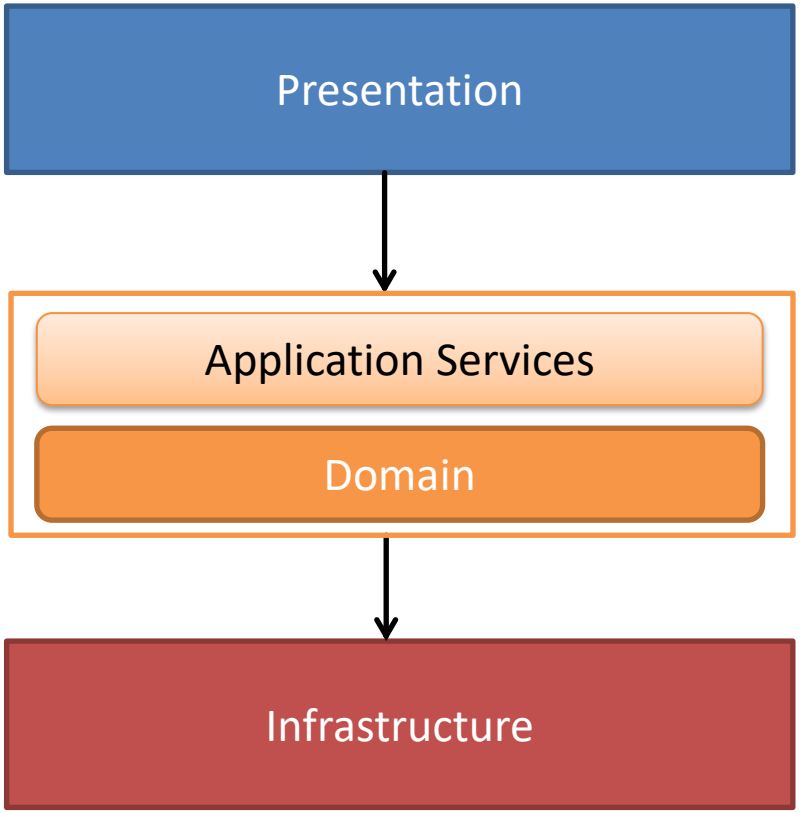
- Die Schichten-Architektur ist **das** „klassische“ Architekturmuster
- Eine Schicht darf nur mit den **unter ihr liegenden** Schichten kommunizieren
- Bei **strikter** Schichten-A. darf nur die **nächstniedrigere** Schicht aufgerufen werden
- Bei einer **offenen** Schichten-Architektur darf **jede beliebige niedrigere** Schicht aufgerufen werden

Gründe für Schichten-Architektur

- Beherrschung der Komplexität durch technische Trennung der Anwendung in mehrere Schichten
- Geringe Kopplung zwischen den Schichten, hohe Kohäsion innerhalb einer Schicht
- Dadurch sollen einzelne Schichten leichter und unabhängig von anderen Schichten geändert werden können

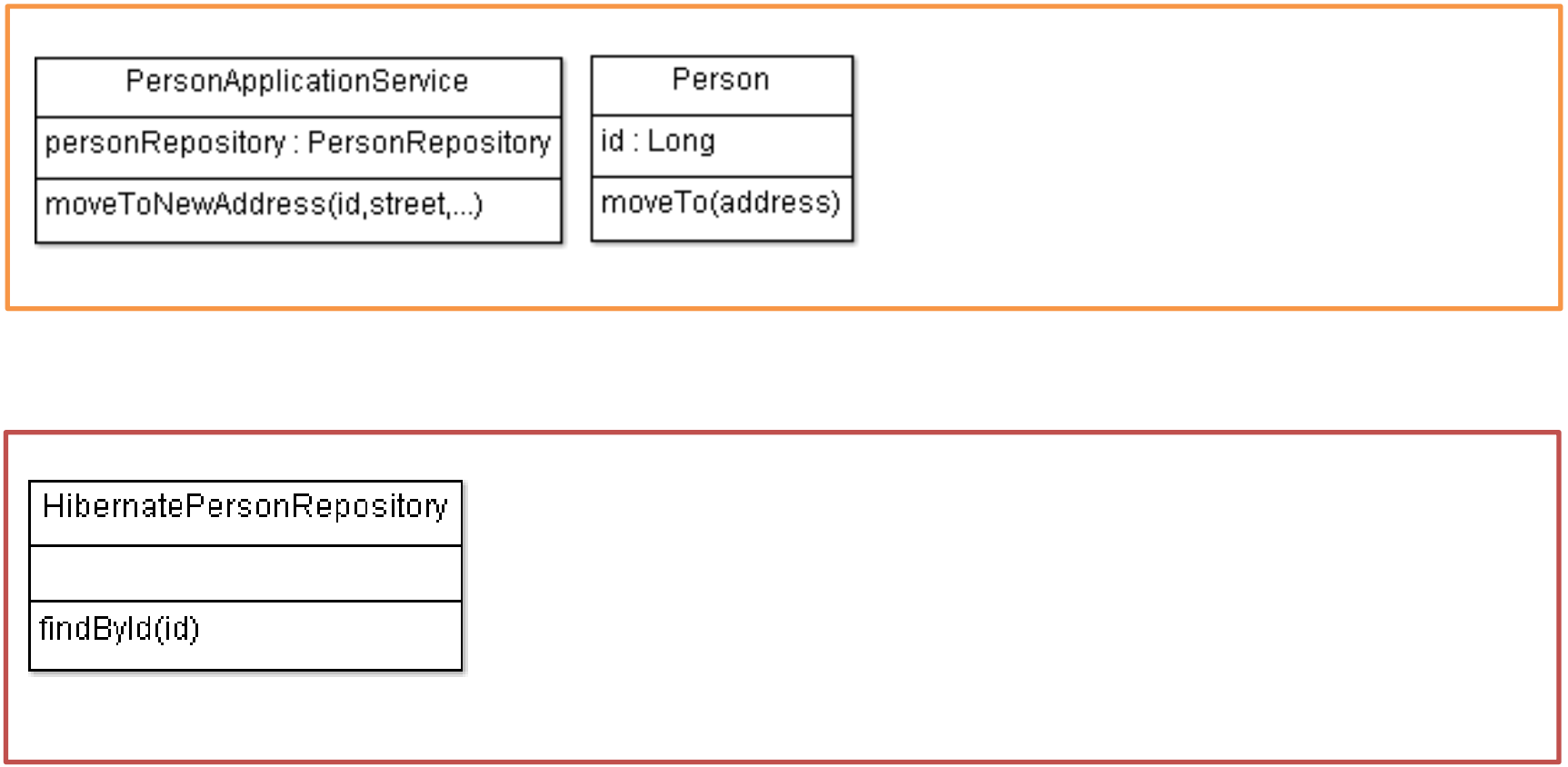
Klassische Schichten-Architektur

Frage: zu welchen Schichten gehören folgende Klassen?



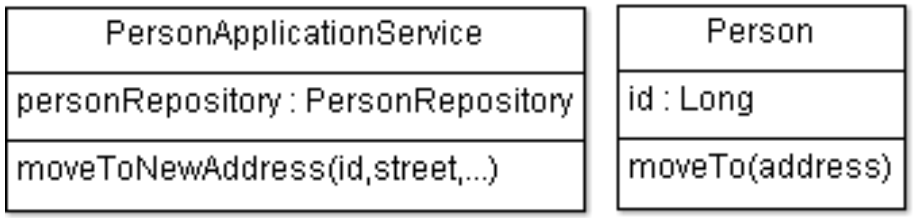
DHBW

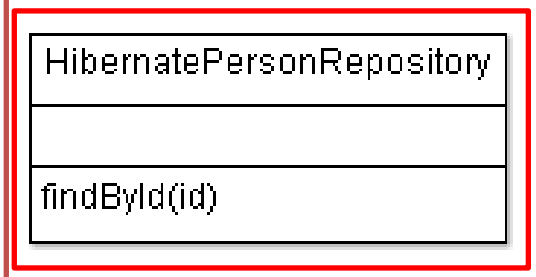
Klassische Schichten-Architektur



DHBW

Problem I





Problem:

- Das Repository ist normalerweise Teil der Domänenschicht bzw. Business-Schicht
- Der konkrete DB-Zugriff (zum Beispiel per Hibernate) gehört aber in die Infrastruktur - Schicht

DHBW

Problem I

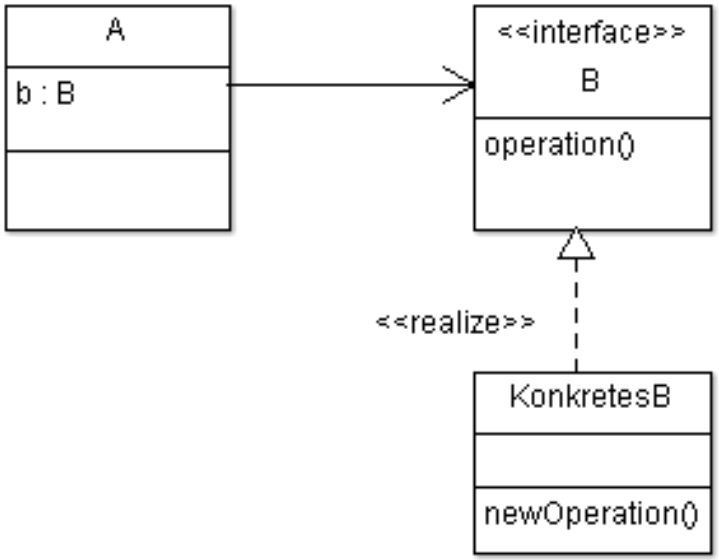
- Die Domänenschicht soll frei von technischen Details bleiben
- Die konkrete Implementierung „HibernatePersonRepository“ darf also nicht in die Domänenschicht
- Gleichzeitig gehört aber das Repository konzeptionell zu Domänenschicht, darf also eigentlich auch nicht in die Infrastruktur-Schicht
- Die Lösung: Dependency Inversion

Dependency Inversion Principle (DIP)

Definition:

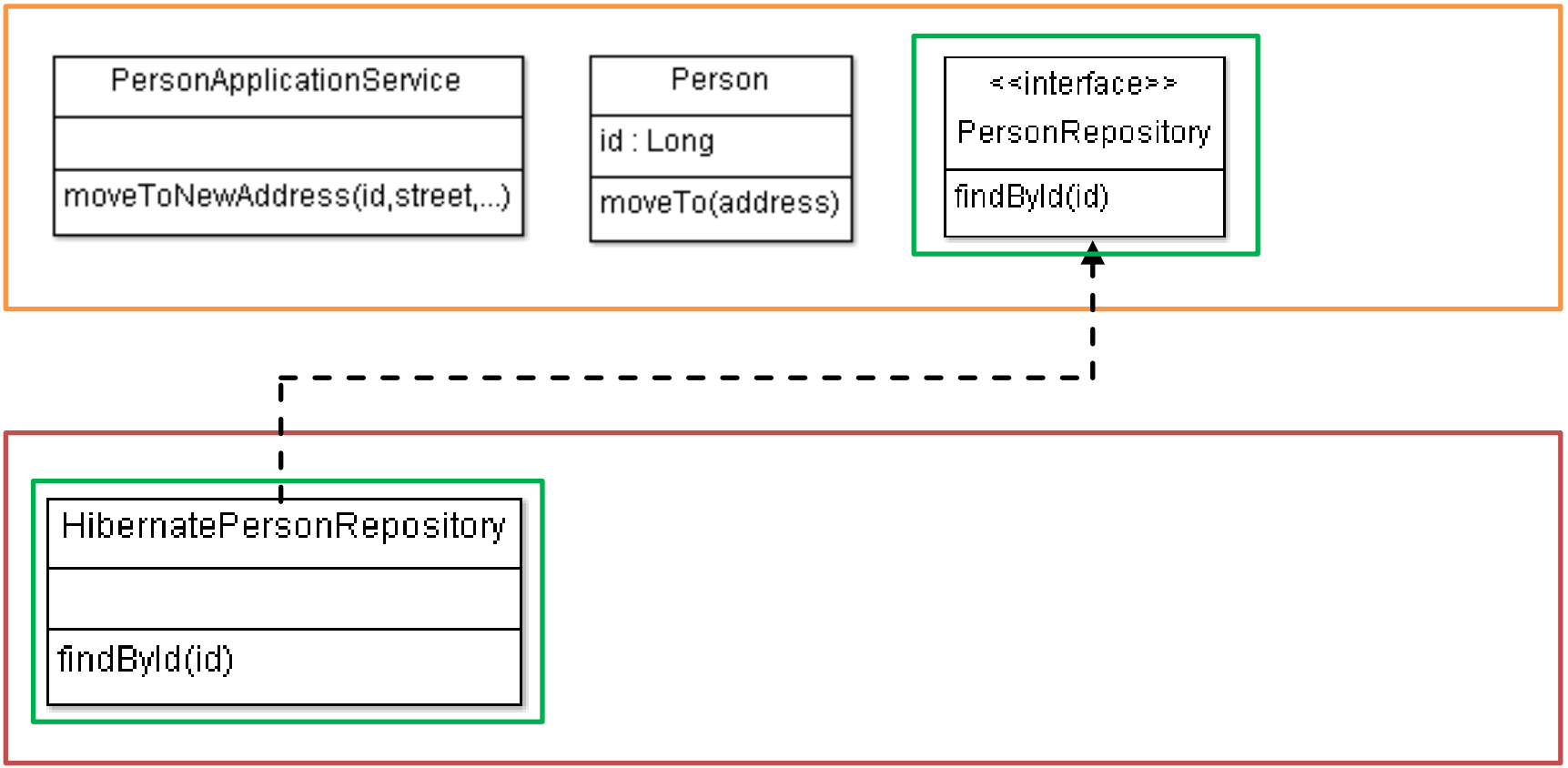
„A. *Module höherer Ebenen sollten nicht von Modulen niedrigerer Ebenen abhängen. Beide sollten von Abstraktionen abhängen.*

B. *Abstraktionen sollten nicht von Details abhängen. Details sollten von Abstraktionen abhängen.“* [Martin, 1996]



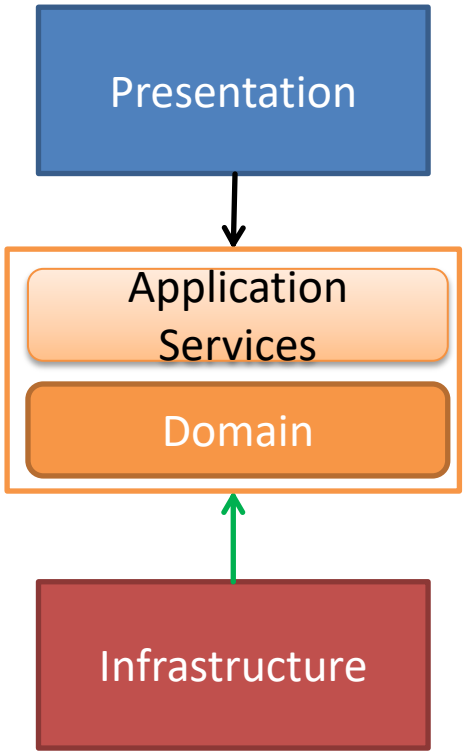
Dependency Inversion Principle (DIP)

Lösung: Definition eines Vertrages durch Dependency Inversion



Dependency Inversion Principle

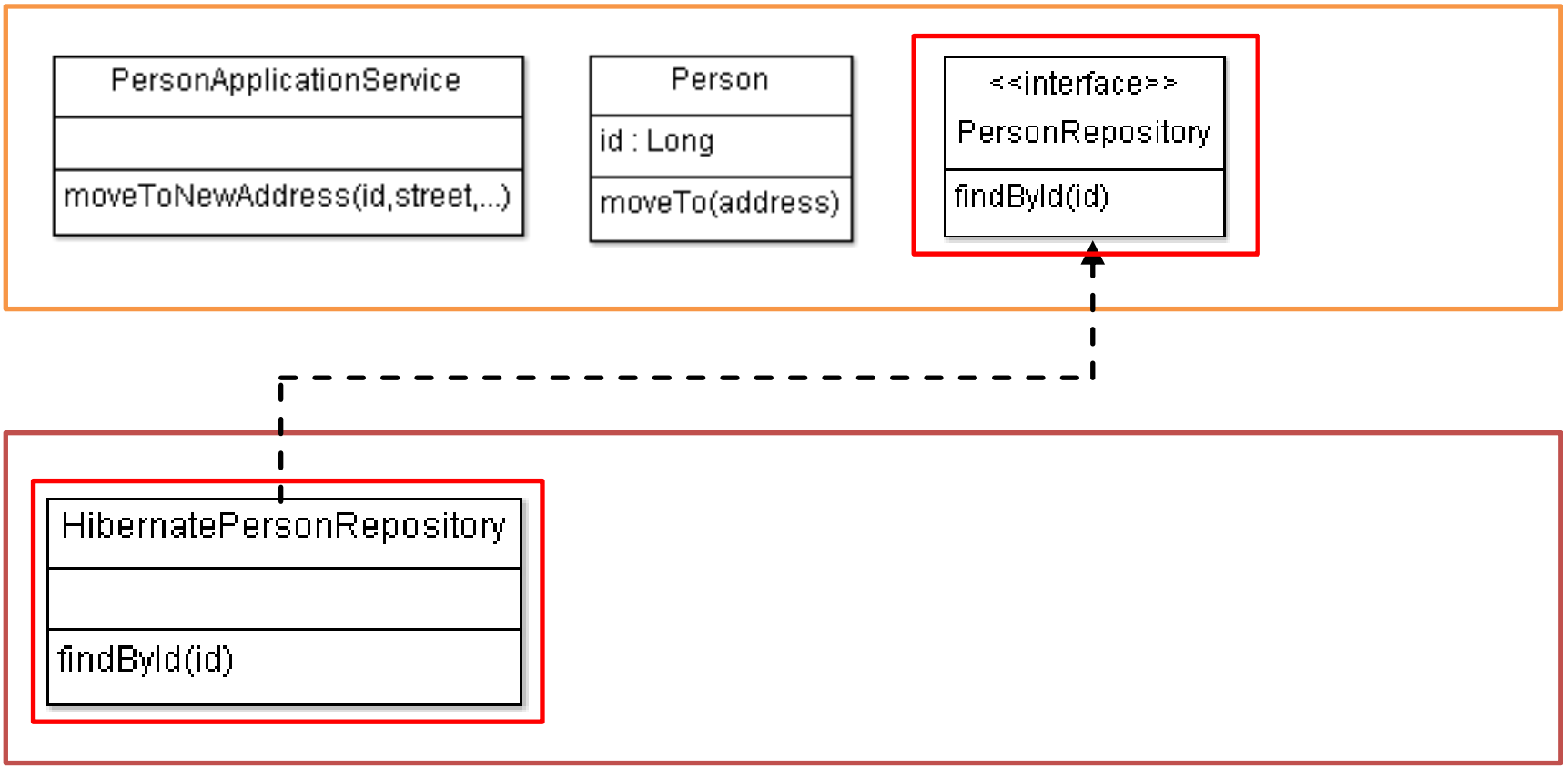
- Die Domänenschicht gibt durch ein Interface einen **Vertrag** vor, der beschreibt, welches Verhalten sie erwartet
- Die **konkrete Implementierung des Vertrages** erfolgt in der Infrastruktur-Schicht
- Die Domäne ist **damit nicht mehr abhängig von Details** (HibernatePersonRepository), sondern von Abstraktionen (PersonRepository)
- die konkrete Implementierung kann dann je nach Anwendungsfall **gewählt** werden, beispielsweise durch **Dependency Injection**



DHBW

Problem II

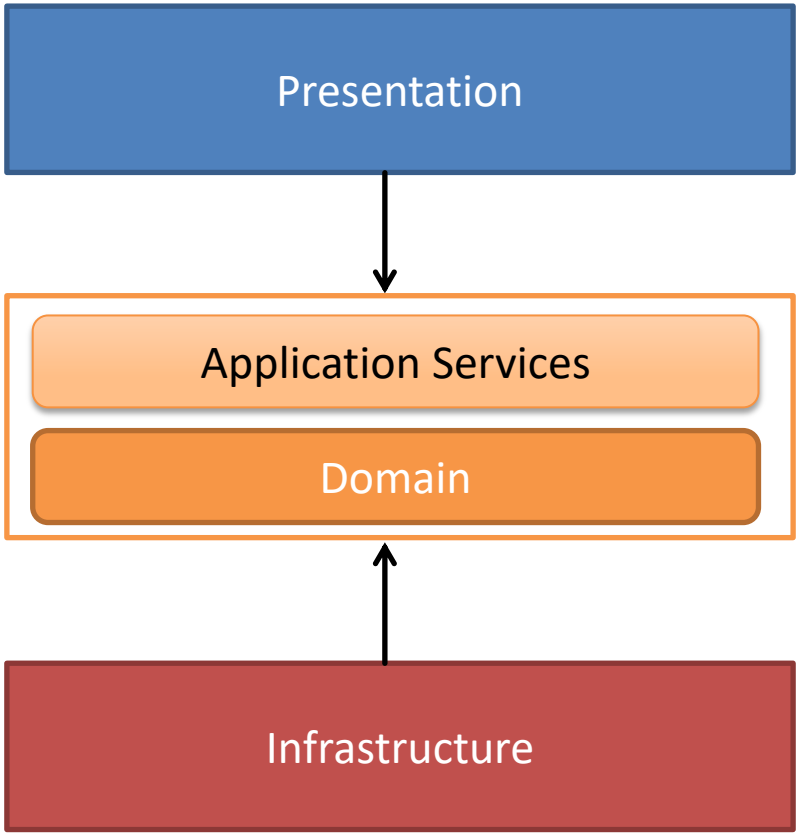
Infrastrukturschicht ist jetzt abhängig von Domänenschicht



Problem II

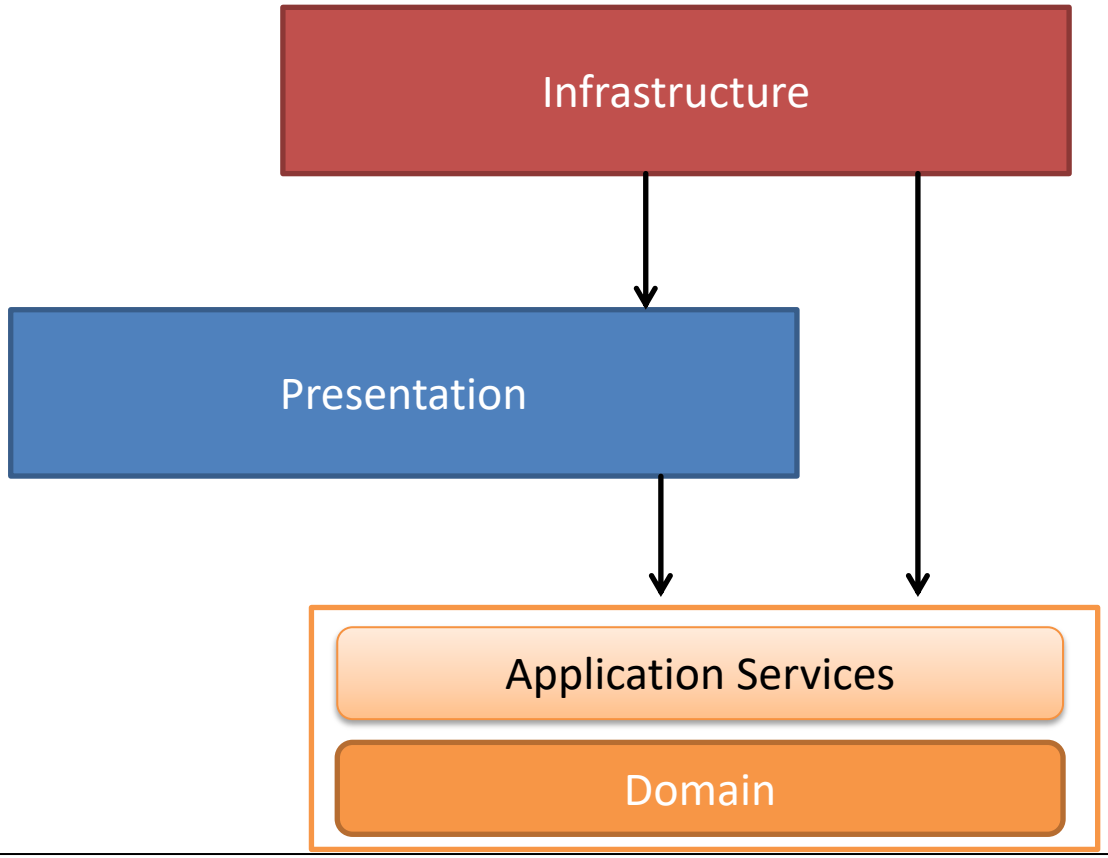
- Die Infrastrukturschicht ist abhängig vom in der Domänenschicht definierten PersonRepository
- Dies verletzt die Regeln der Schichtenarchitektur (Schicht darf nur von darunterliegenden Schichten abhängig sein)
- Die Lösung: das „Verschieben“ der Infrastruktur-Schicht

Umschichtung



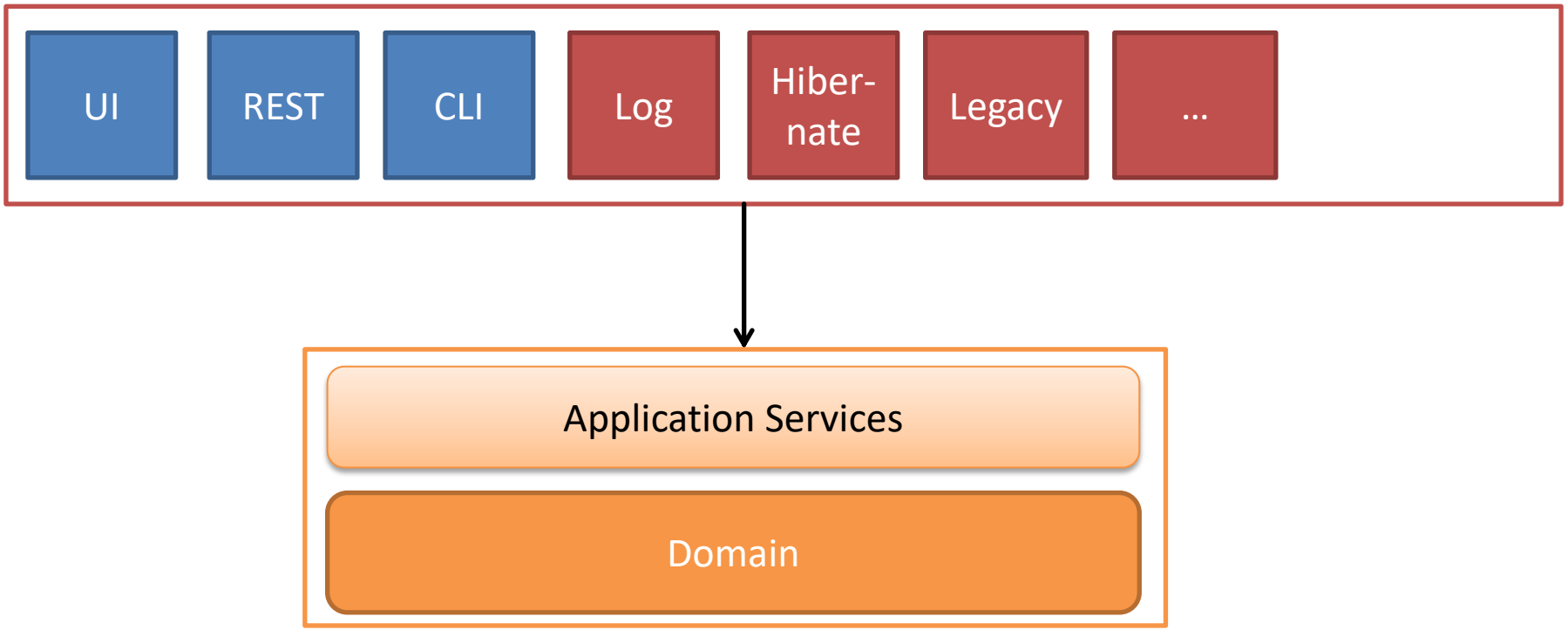
DHBW

Umschichtung



DHBW

Umschichtung

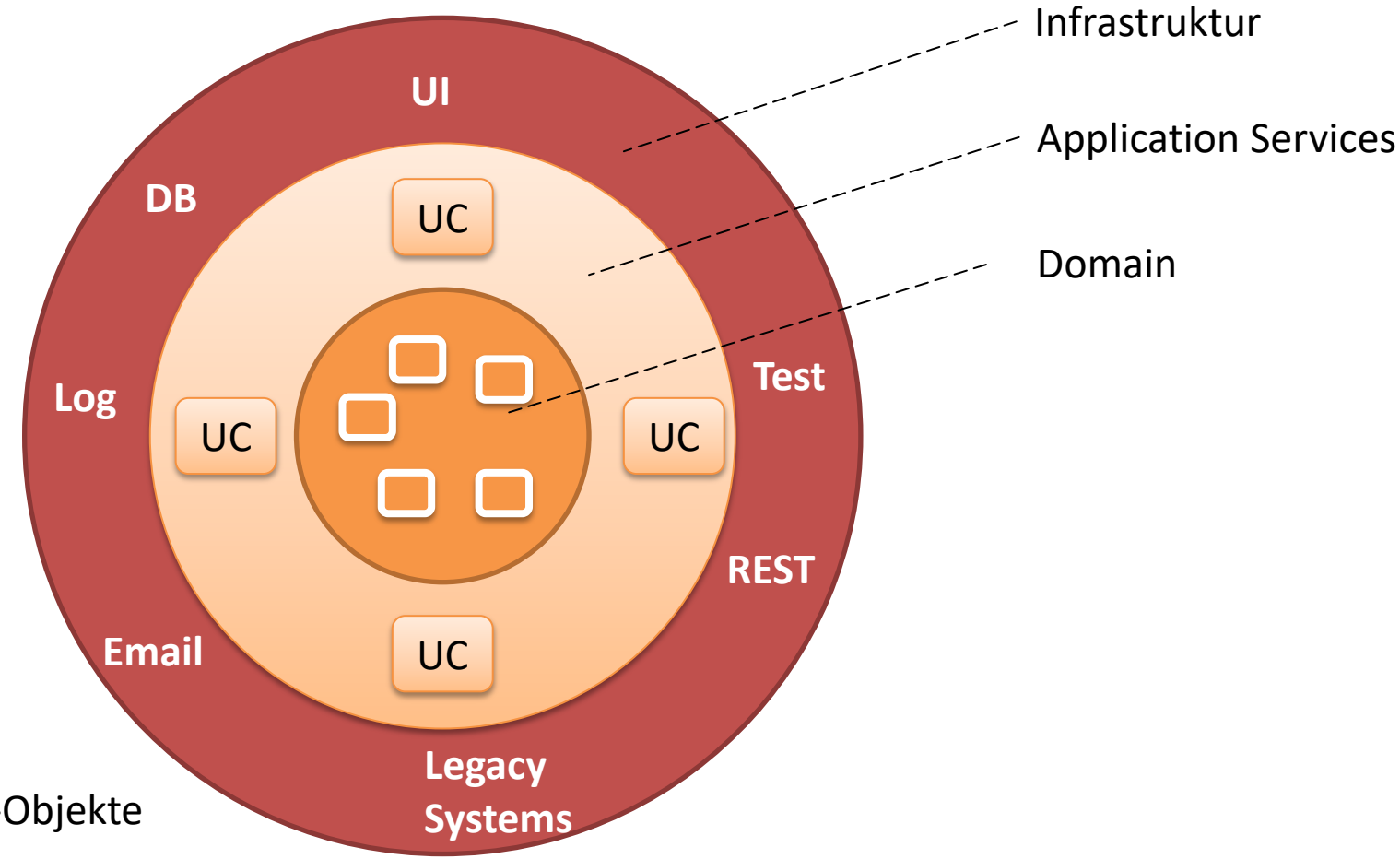


DHBW

Umschichtung

- Durch das Verschieben der Infrastruktur-Schicht darf diese konkrete Verträge für alle darunter liegenden Schichten implementieren, ohne die Regeln der Schichten-Architektur zu verletzen
- Die Präsentationsschicht kann dann auch als Teil der Infrastrukturschicht betrachtet werden
- Die Domänenschicht liegt ganz unten und ist von keiner anderen Schicht abhängig -> sie bildet den **Kern der Architektur**

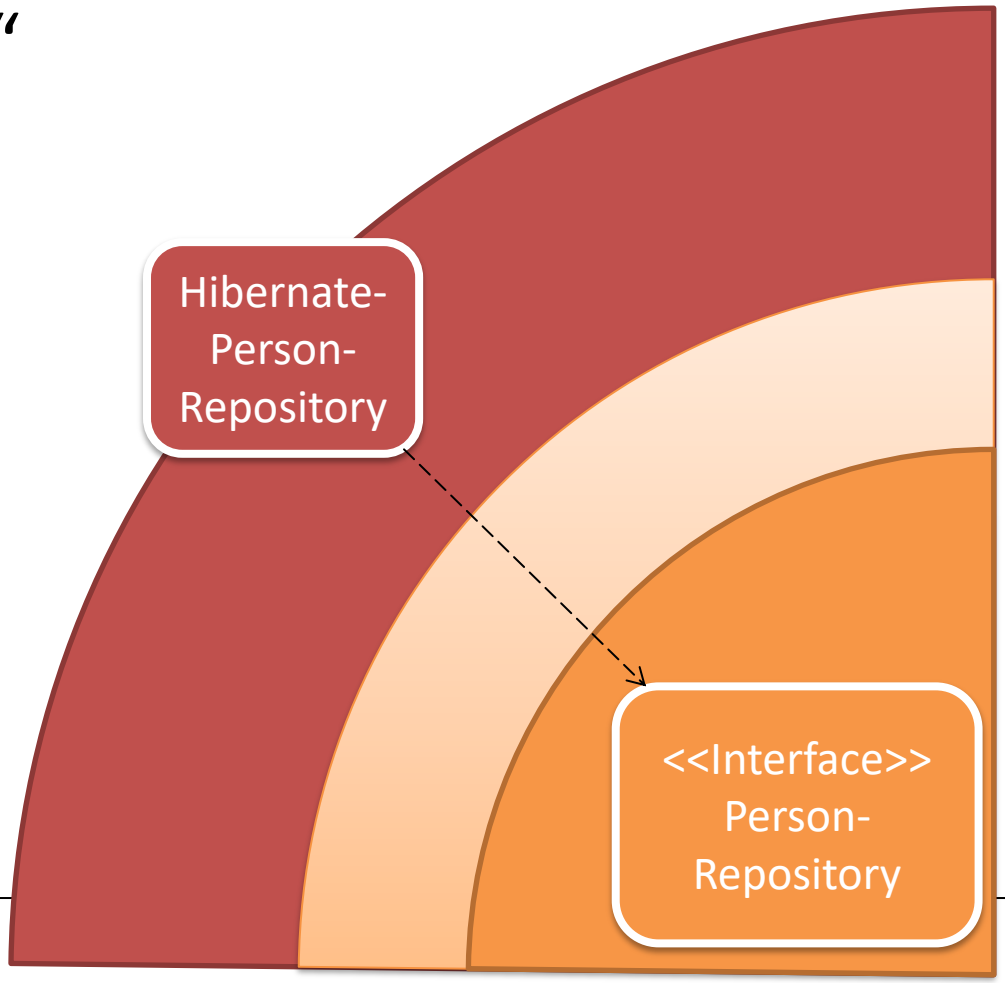
Domäne als isolierter Kern: Zwiebel statt Sandwich



DHBW

Hauptmerkmale der „Onion Architecture“

- Alle Abhängigkeiten zeigen von „Aussen“ nach „Innen“



Hauptmerkmale der „Onion Architecture“

- Alle Abhängigkeiten zeigen von „Aussen“ nach „Innen“ ->
  - Schichten im Inneren sind dann auch nie von weiter außen liegenden Schichten abhängig
- Technische Details werden innerhalb der Infrastruktur-Schicht definiert; UI, DB usw. sind **Klienten** der inneren Schichten
- Die inneren Schichten sind **frei von technischen Details** und daher weniger anfällig für Änderungen der UI usw.

Domain Layer

- Enthält Kernobjekte und Regeln der Fachlogik, im Falle von DDD also das Domänenmodell (Aggregates,Entities, Value Objects, Domain Services, …)
- Implementiert organisationsweit gültige Geschäftslogik (Enterprise Business Rules)
- Sollte sich am seltensten ändern
  - Immun gegen Änderungen an Details wie Anzeige, Transport oder Speicherung
  - Unabhängig vom konkreten Betrieb der Anwendung

Application Service Layer (ASL)

- Normalerweise ist bei der Abbildung eines bestimmten **Anwendungsfalles (Use Case)** mehr als ein Domänenobjekt aus der Domänenschicht involviert
- Beispiel: Anwendungsfall „Ändern der Kundenadresse eines Auftrags“
  - benötigt Zugriff auf **CustomerRepository** und **OrderRepository**

Application Service Layer (ASL)

- Die ASL implementiert diese Anwendungsfälle als sog. **Application Services**
- Änderungen an dieser Schicht beeinflussen die Domain Layer nicht
- Isoliert von Änderungen an der Datenbank, der graphischen Benutzeroberfläche, etc.
- Wenn sich Anforderungen ändern, hat das wahrscheinlich Auswirkungen auf diese Schicht
- Wenn sich der konkrete Betrieb der Anwendung ändert, kann das hier Auswirkungen haben

Aufgaben eines Application Service

- Implementierung eines Anwendungsfalls
- Validierung, Übersetzung und Aufbereitung von Eingaben und Ausgaben
- Reporting
- Security

Aufgaben eines Application Service:
Implementierung eines Anwendungsfalls

- Ein Application Service bildet einen oder mehrere Anwendungsfälle ab
- Er nutzt dazu die Komponenten des Domänenmodells und orchestriert und koordiniert diese, um den gewünschten Anwendungsfall umzusetzen
- Ein Application Service enthält selbst keine Regeln; er weiß lediglich, welche Domänenobjekte er in welche Reihenfolge aufrufen muss

Aufgaben eines Application Service:
Implementierung eines Anwendungsfalls

- Ein Application Service folgt eher einem prozeduralen Programmierstil (siehe auch Entwurfsmuster „Transaction Script“)
- Bietet normalerweise keine Create/Read/Update/Delete-Methoden, sondern konkrete Methoden für den jeweiligen Anwendungsfall (Erinnerung: Ubiquitous Language)

Aufgaben eines Application Service:
Validierung, Übersetzung, Aufbereitung von Eingaben und Ausgaben

Validierung:

- Application Service stellt sicher, dass alle benötigten **Eingaben** zur Realisierung eines Anwendungsfalls **vorhanden** und **korrekt** sind
- Prüft **keine Regeln der Domäne**, sondern **technische Details** (korrekter Datentyp, korrektes Format, not null usw.)

Aufgaben eines Application Service:
Validierung, Übersetzung, Aufbereitung von Eingaben und Ausgaben

Übersetzung:

- **Übersetzt** die **Eingaben** der Infrastrukturschicht in **Domänenobjekte**
  - Bspw. Mapping von Request-Parametern auf ein Domänenobjekt
- **Übersetzt** bei Bedarf die **Ausgaben** der Domänenschicht für die Außenwelt (beispielsweise Übersetzung eines Kunden in ein allgemeineres **Data Transfer Objects** für eine REST-API)

Aufgaben eines Application Service:
Reporting

- Oft muss eine Anwendung verschiedene Berichte (Reports) liefern, um Auswertungen zu ermöglichen, beispielsweise
  - Umsätze
  - Lagerbestände je Artikel
- Zur Generierung solcher Berichte müssen normalerweise verschiedene Daten abgefragt und zusammengefasst werde – beispielsweise alle Artikel und alle Lagerbewegungen in einem bestimmten Zeitraum
- Ein Application Service kann diese Daten aggregieren und in einem speziellen Report-Objekt (Data Transfer Object) zurückliefern
- Darf ggf. auch nativ (und damit am Domänenkern vorbei) auf die Persistenzschicht zugreifen, wenn Performance dies erfordert

Aufgaben eines Application Service:
Security

- Meist existieren in einer Applikation Anwendungsfälle, die nur Benutzern mit bestimmten Rechten zugänglich sind
- Application Services bieten sich daher auch für Authentifizierung und Autorisierung an
- Die konkrete Umsetzung ist von der Art der Zugriffskontrolle abhängig (rollenbasiert, …)

Wie stark soll die Domäne von der Außenwelt abgeschottet sein?

Es gibt zwei Möglichkeiten:

- Entweder dürfen (bestimmte) Objekte des Domänenmodells die ASL passieren und an die Außenwelt weitergegeben werden
- oder die Außenwelt hat keine Kenntnis von den Objekten des Domänenmodells

Diskussion

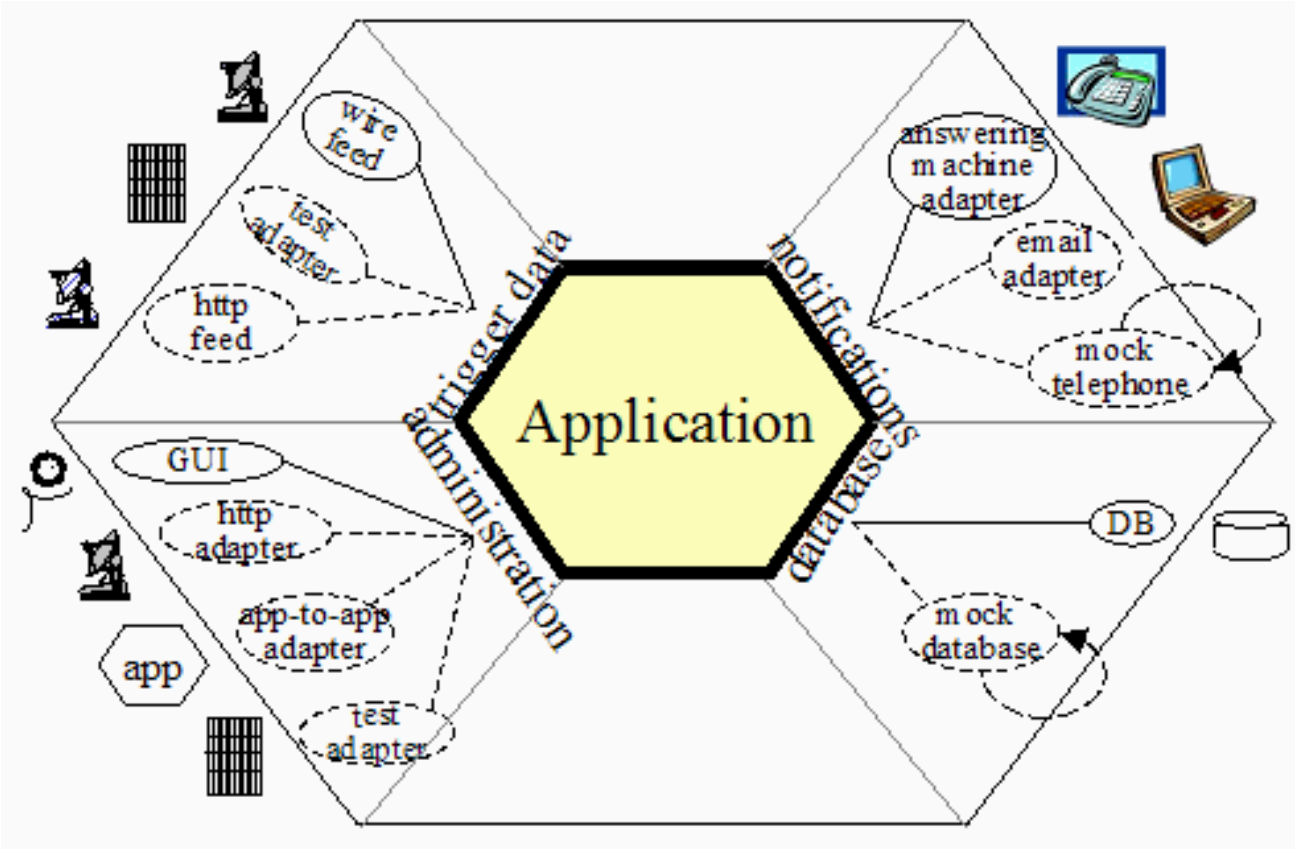
Infrastructure Layer

- Die Infrastruktur-Schicht stellt die **technischen Details** einer Anwendung bereit
- Diese Details sind in der äußersten Schicht angesiedelt – fern vom Kern
- Beispiele:
  - UI
  - Web Services
  - Datenbankzugriff
  - Anbindung von Fremdsystemen

Varianten der Onion Architecture

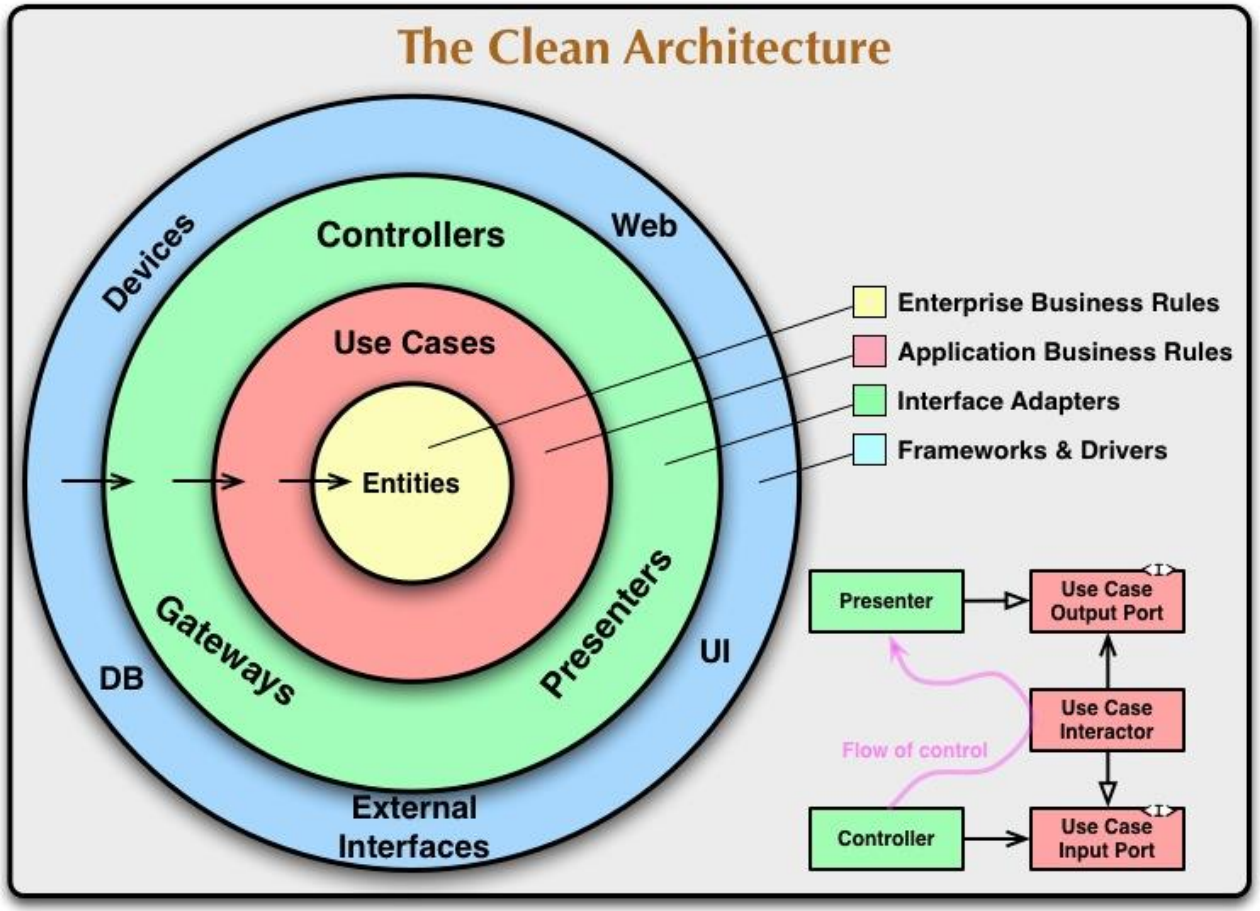
- Aktuell existieren mehrere Varianten der „Onion Architecture“, unter anderem:
  - Hexagonale Architektur / Ports-and-Adapters
  - Clean Architecture
- Zwar unterscheiden sich die genannten Architekturstile im Detail, das Grundprinzip (Abhängigkeiten zeigen von Außen nach Innen) ist aber immer dasselbe.

Hexagonale Architektur
a.k.a. Ports-and-Adapters



By Alistair Cockburn

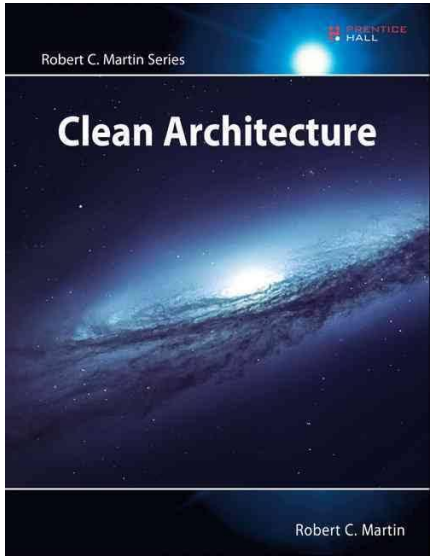
Clean Architecture



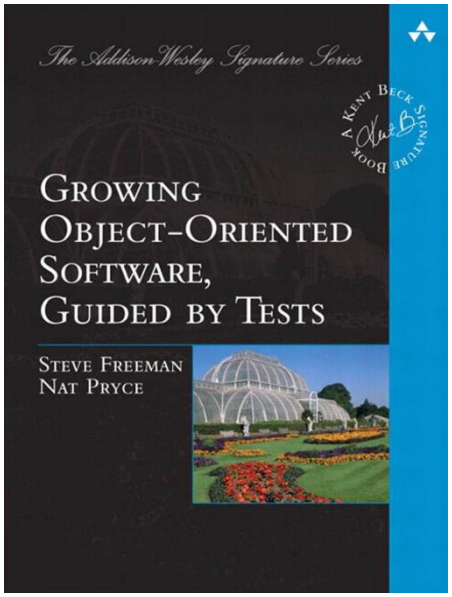
By Robert Martin

Nächstes Semester!

Literaturempfehlungen



2017



2009

https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/

DHBW