ZÜRICH / 22.11.2018

JAVA USER GROUP SWITZERLAND

DDD MIT ONION ARCHITECTURE & STEREOTYPES

DIE APPLIKATIONSARCHITEKTUR FÜR DOMAIN-DRIVEN DESIGN

Christian Stettler

INNOQ Schweiz

INNOQ

CHRISTIAN STETTLER

Senior Consultant / Architekt, INNOQ Schweiz
christian.stettler@innoq.com

Christian ist Senior Consultant und Architekt bei INNOQ in der Schweiz. Seit über 15 Jahren entwickelt er mit Leidenschaft fachlich relevante Systeme mit verteilten, skalierbaren Architekturen. Als DDD-Enthusiast gilt sein Interesse dabei vor allem den Ansätzen von Domain-Driven Design, um ein umfassendes fachliches Verständnis zu erlangen und dieses möglichst verständlich in Code zu giessen. Bei Gelegenheit gibt er seine Erfahrungen in Workshops und Trainings weiter.

Domain-Driven Design To The Code

10. – 12.09.2018 Berlin

03. – 05.12.2018 Berlin

Inhouse-Training (2 – 3 Tage)

https://www.innoq.com/de/trainings/ddd-to-the-code/

Motivation

"Make Fachlichkeit in Code great again!"

some no-name

Disclaimer

Keine Erfindungen von mir selbst

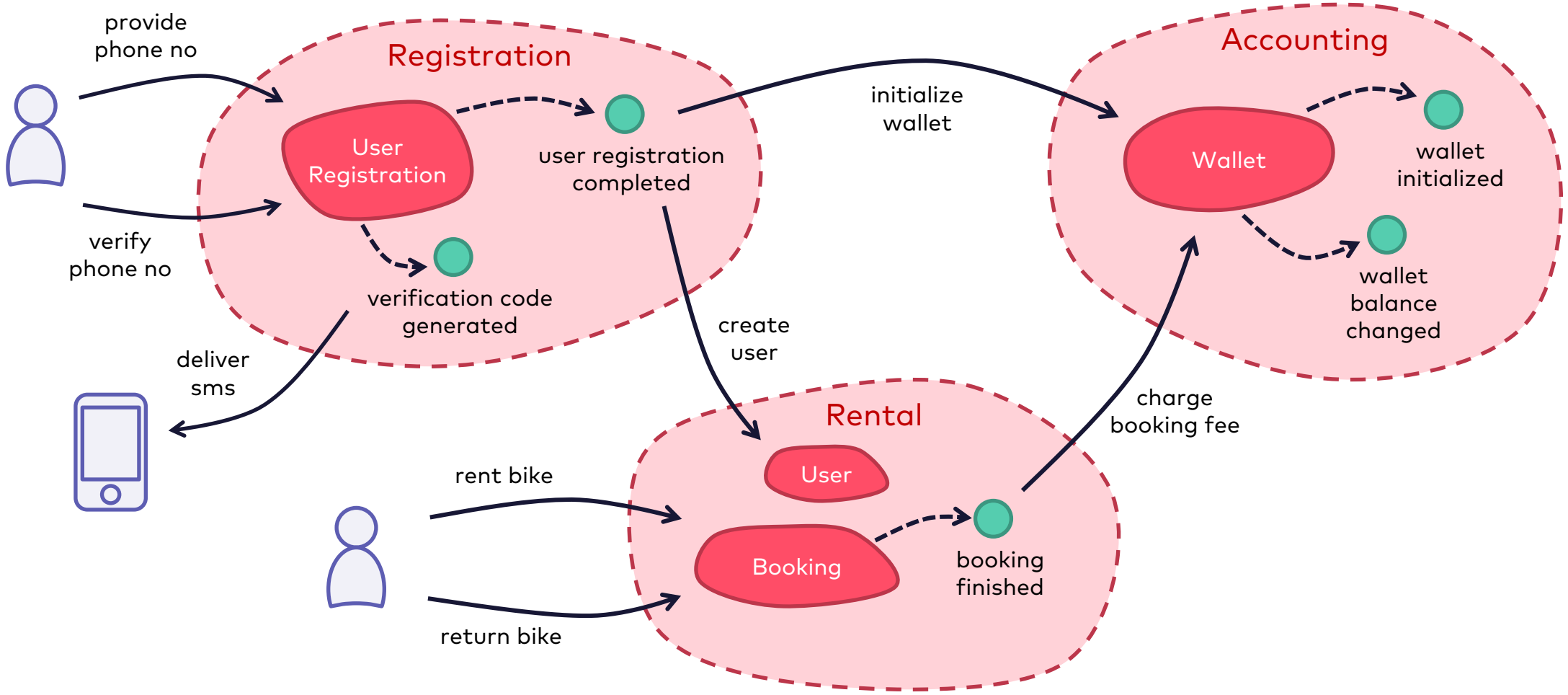
Kein Silver Bullet

Kein produktionsreifer Code

Keine Einführung in DDD

Beispiel: Bike Sharing Domäne

Beispiel: Bike Sharing Domain



Onion Architecture

"The database is not the center – it is external."

Jeff Palermo, 2008

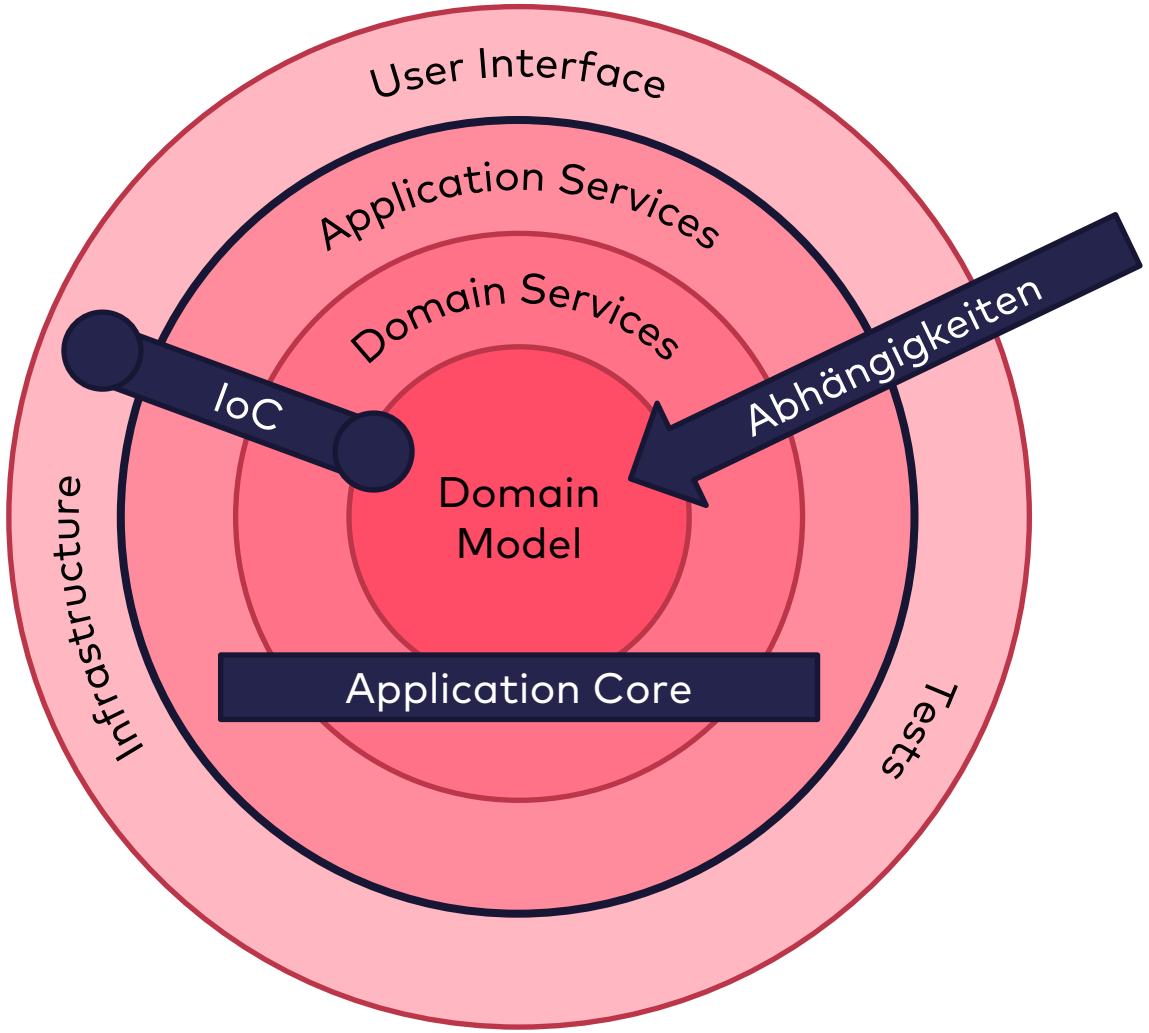
https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/

Onion Architecture

- Fachlichkeit im Zentrum
- Technologien am Rand

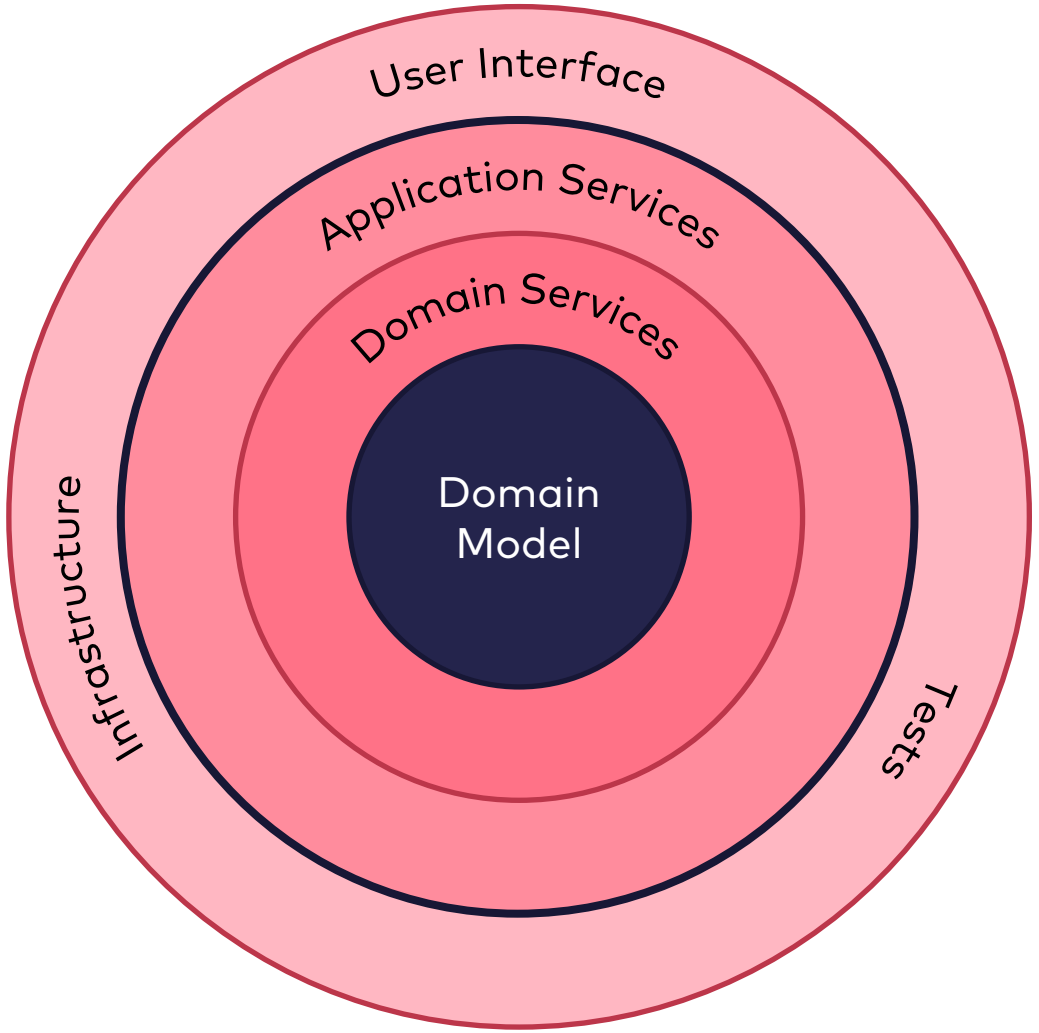
- Hexagonal Architecture
- Ports & Adapters
- Screaming Architecture

- Konzentrische Ringe
- Abhängigkeiten nach innen
- Inversion of Control



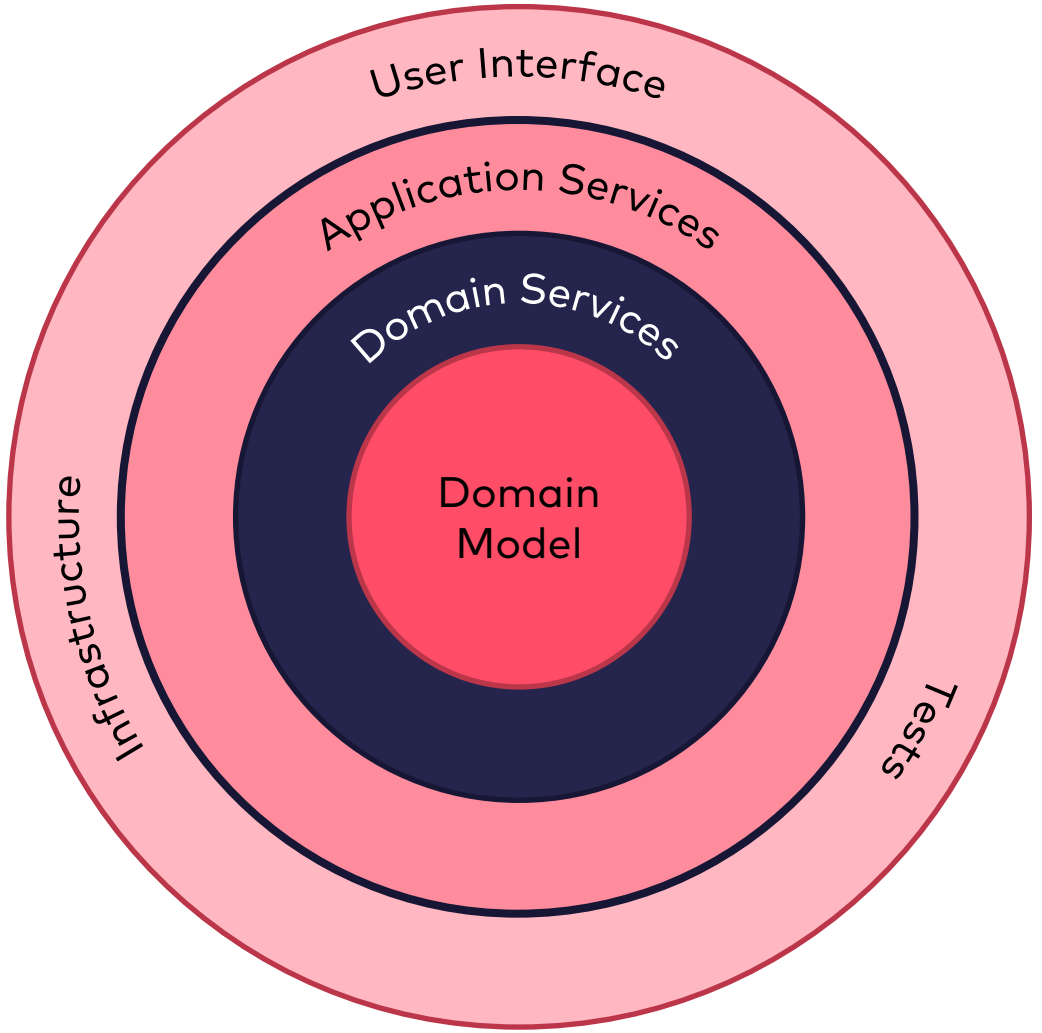
Domain Model

- **Abbildung des fachlichen Modells**
- **Zustand und Fachlogik**
  (kein blutleeres Domain Model)



Domain Services

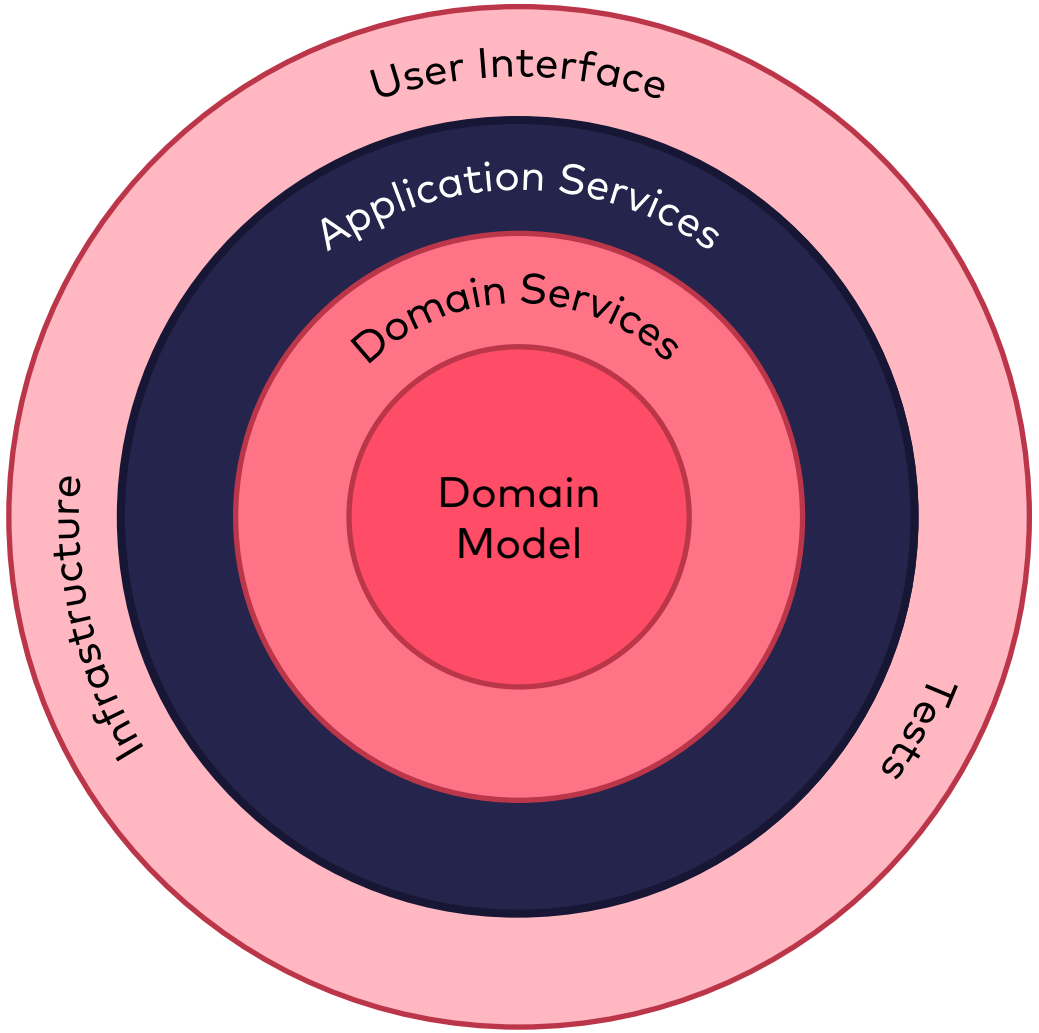
- **Abbildung von übergreifender Fachlogik**
- **Kapselung der Fachlogik und Schutz der Invarianten des Domain Models**



Application Services

- **Orchestrierung der fachlichen Anwendungsfälle**
- **keine Fachlogik**

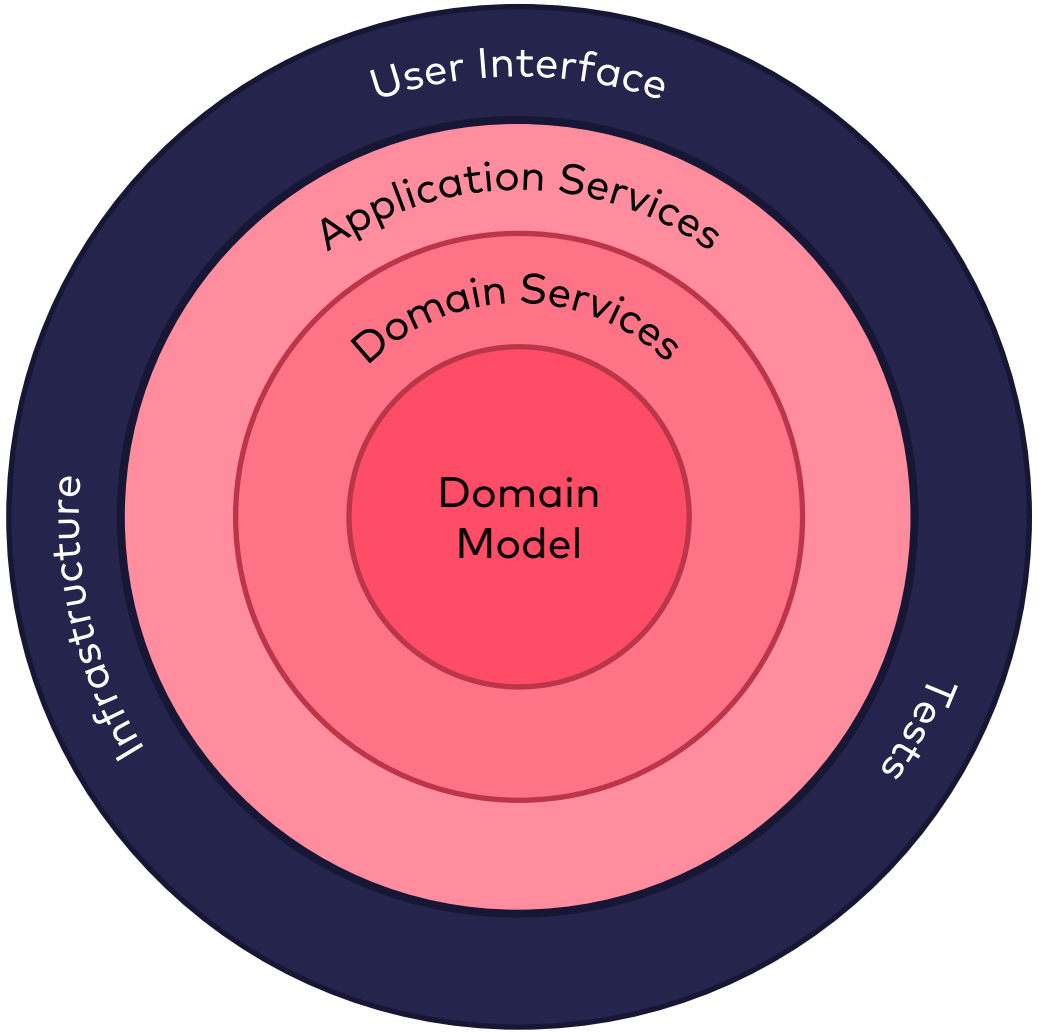
- **Querschnittsfunktionen**



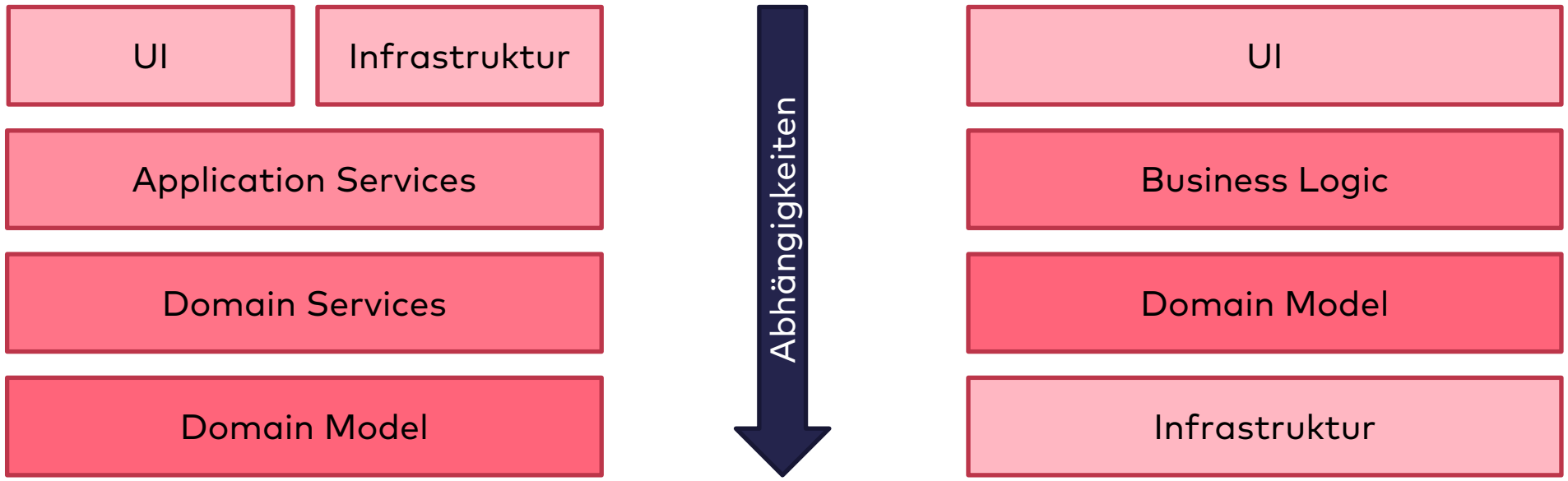
Infrastructure, User Interface, Tests

- „alles andere“

- Anbindung an Infrastruktur (DB, Message Bus, Umsysteme, ...)
- User Interface / Web APIs
- Fachliche Tests gegen Funktionalität der Domäne



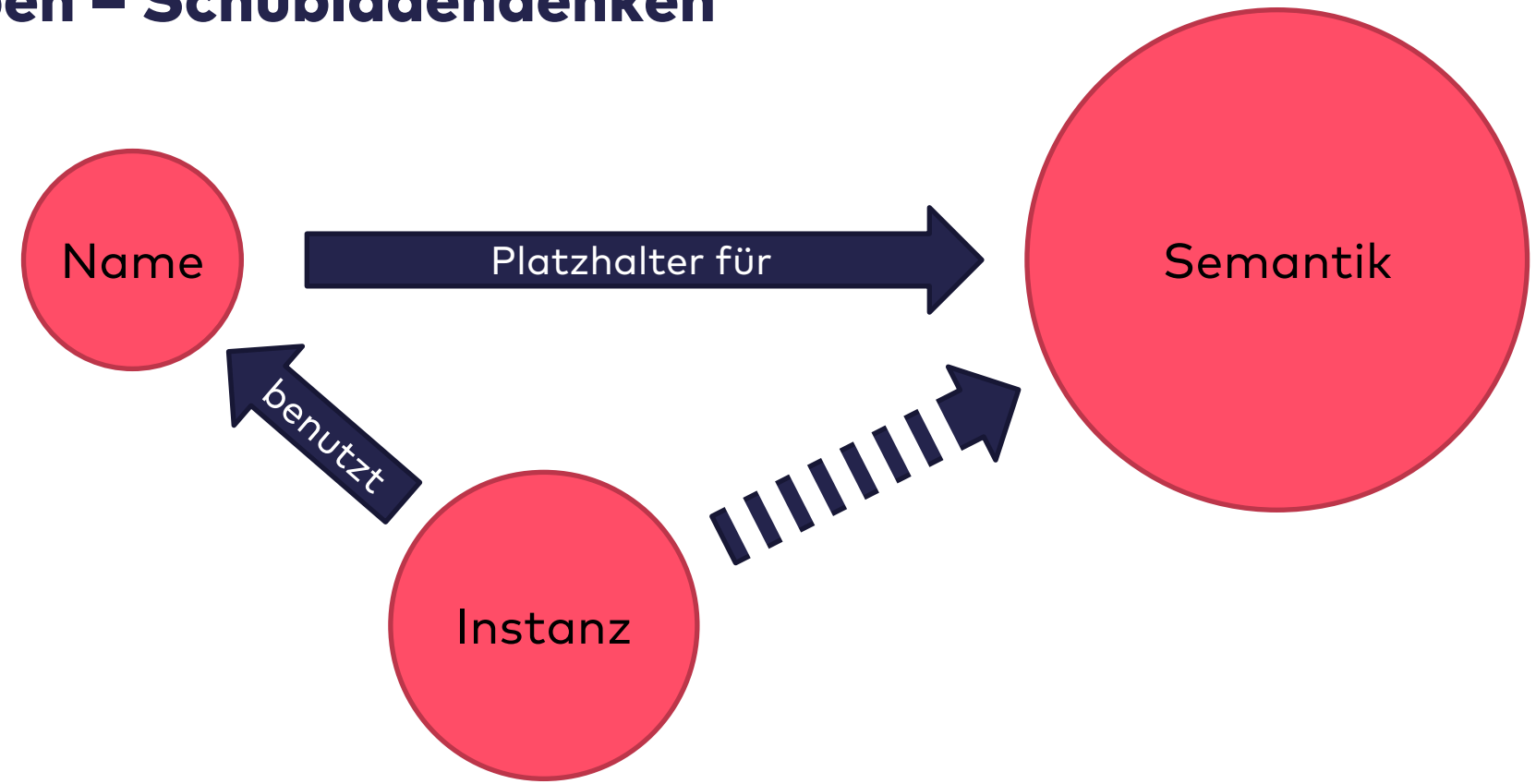
Onion Architecture vs. N-Tier Architecture



Onion Architecture **N-Tier Architecture**

Stereotypen

Stereotypen – Schubladendenken



- **“Schublade“ für konkrete Ausprägungen**
- **Semantik “erben“ durch Benutzung des Namens**

Stereotypen – Umsetzungsoptionen

Ansatz	Vorteile	Nachteile
Klassennamen	• simpel	• lange Namen • Verschmutzung durch Suffix • keine Verwendung im Typsystem
Basisklassen	• keine Verschmutzung des Namens • Verwendung im Typsystem	• Einfluss auf fachliche Hierarchien • Gefahr von Container für technische Funktionalität
Annotationen	• nicht invasiv • Konzept für Beschreibung • kein Einfluss auf Hierarchie • Verwendung auf Methoden, etc. • selbst kein direktes Verhalten	• keine Verwendung im Typsystem

Stereotypen – Umsetzung als Annotation

```java
/**
 * Represents an aggregate with core domain logic and related state. An aggregate
 * ensures its domain invariants and represents the minimal scope of a business
 * transaction ...
 */
@Target(TYPE)
@Retention(RUNTIME)
@Documented
public @interface Aggregate {

}
```

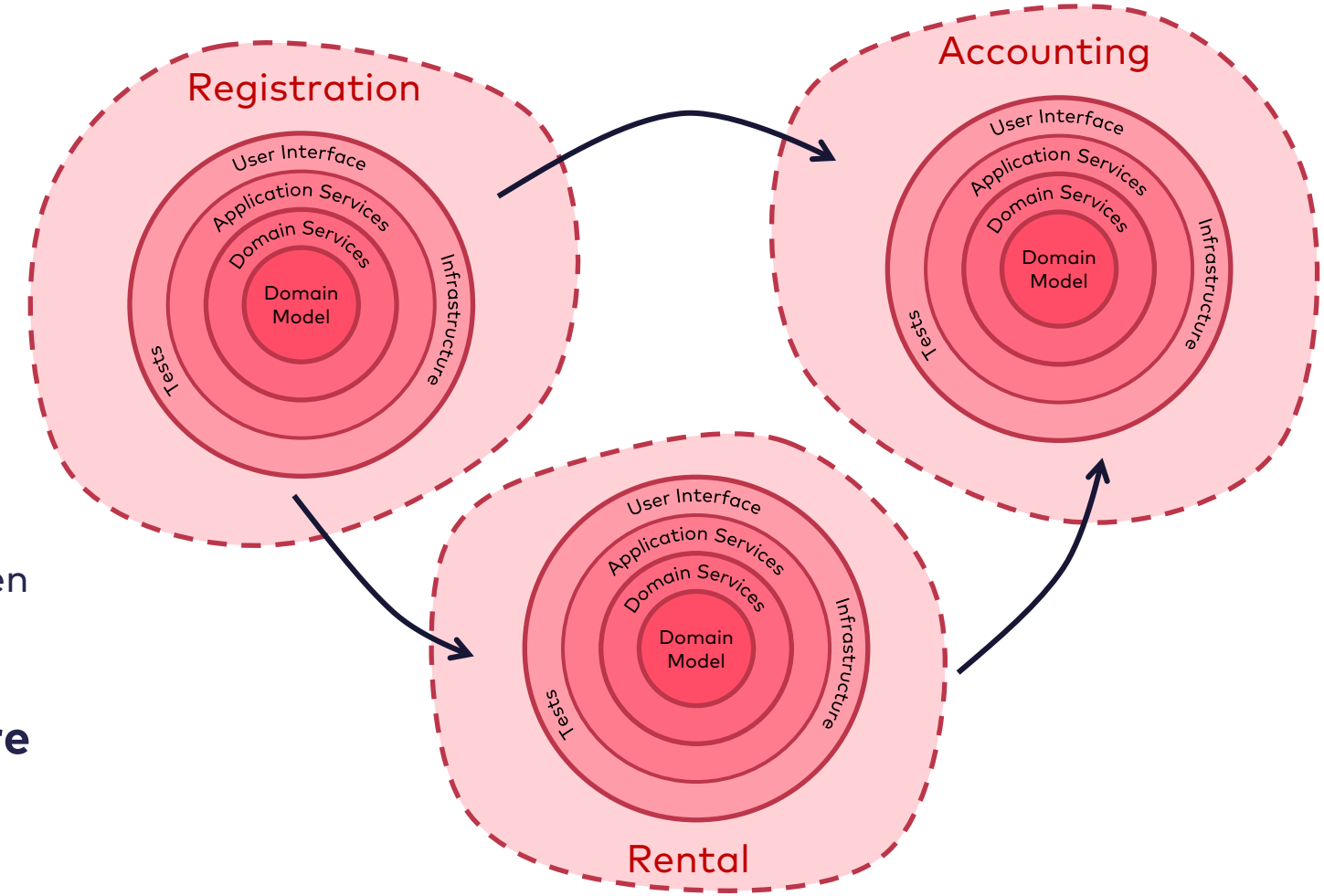
```java
@Aggregate
public class Customer {

}
```

Umsetzung mit Onion Architecture

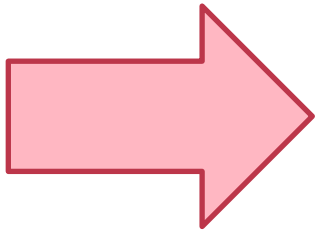
Onion Architecture – Abbildung von Bounded Contexts

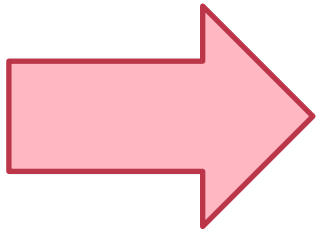
- **Going-In: Eine „Zwiebel“ pro Bounded Context**
- **Weitere Aufteilung bei Bedarf**
  - Kohäsion von Sub-Kontexten
  - Organisation
  - Unabhängigkeitsanforderungen
- **Keine Zwiebeln über mehrere Bounded Contexts hinweg**

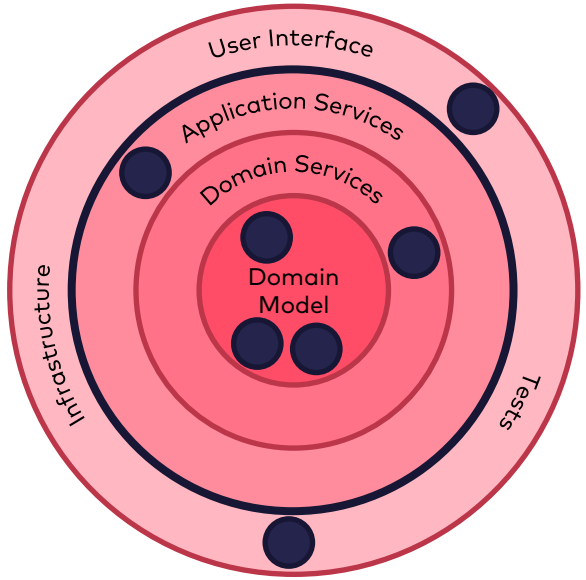


Onion Architecture – Abbildung von taktischen Mustern

Entity
Value Object
Aggregate
Factory
Service
Repository
Domain Event
...





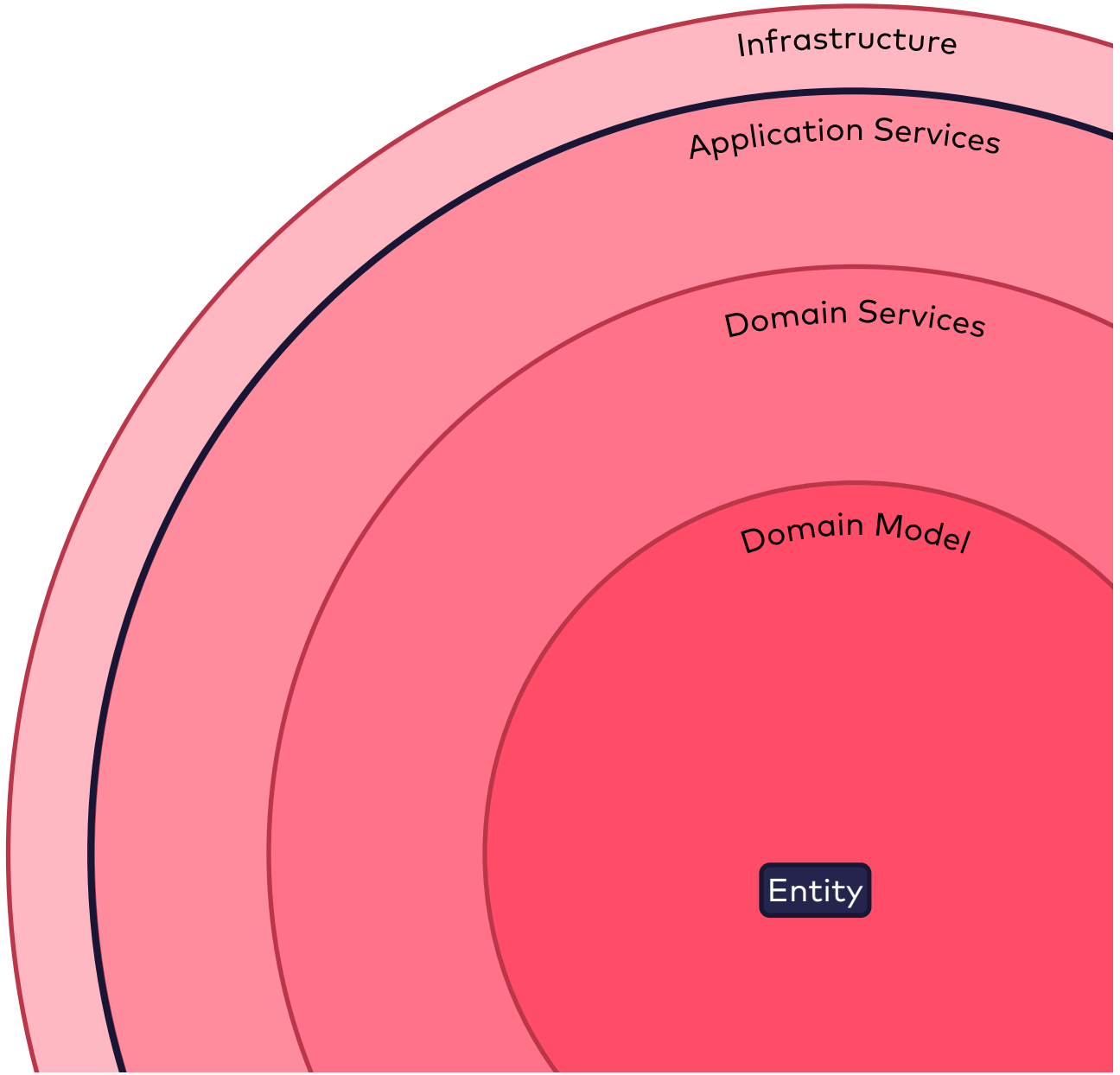


Taktische Muster | **Stereotyp(en)** | **Onion Architecture**

Entity

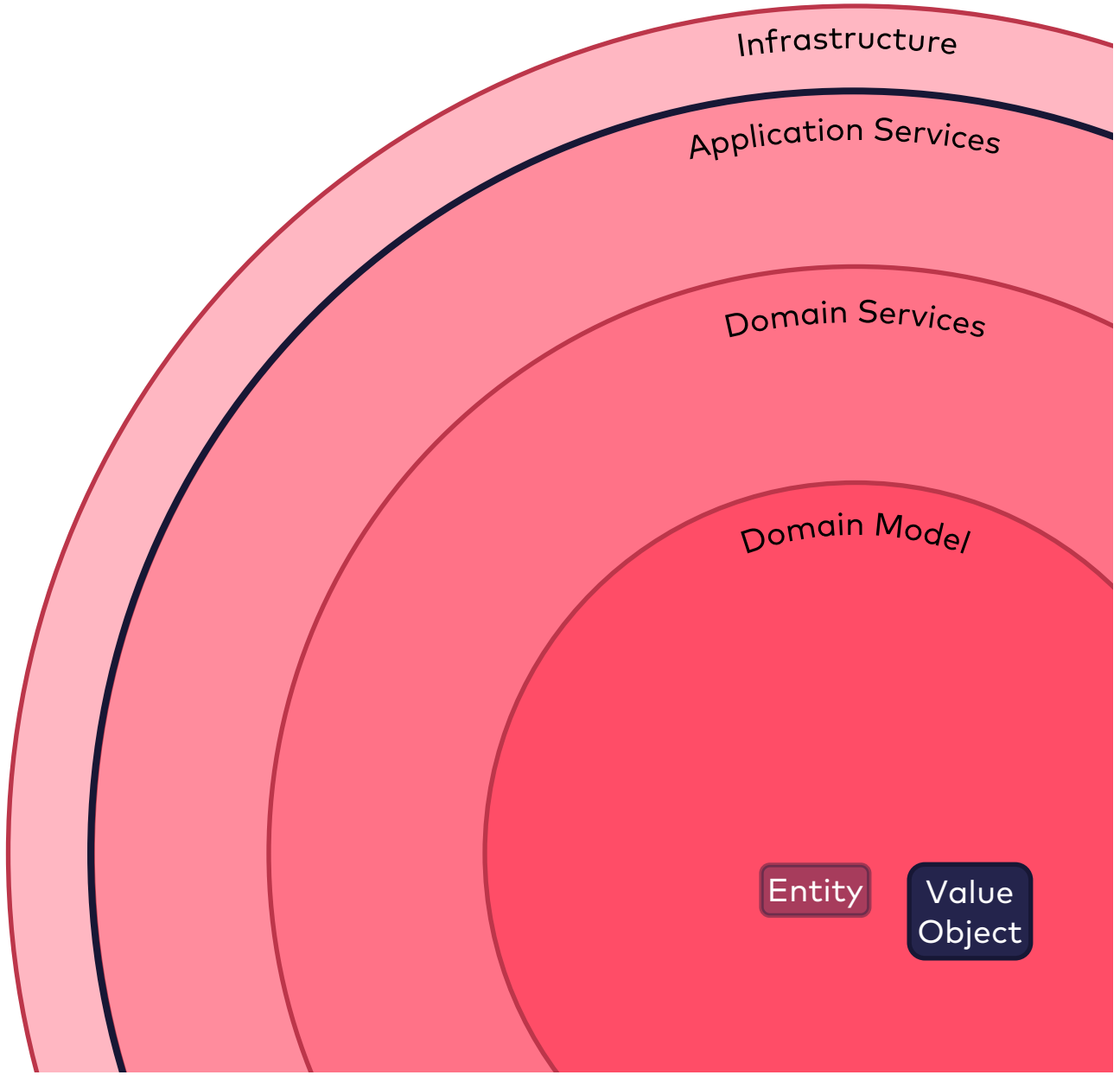
- **Fachliche Identität und Lebenszyklus**
- **Veränderlicher Zustand**
- **Gleichheit über Id**

```
@Entity
```



Value Object

- **Gleichheit über Zustand**
- **Unveränderlich**

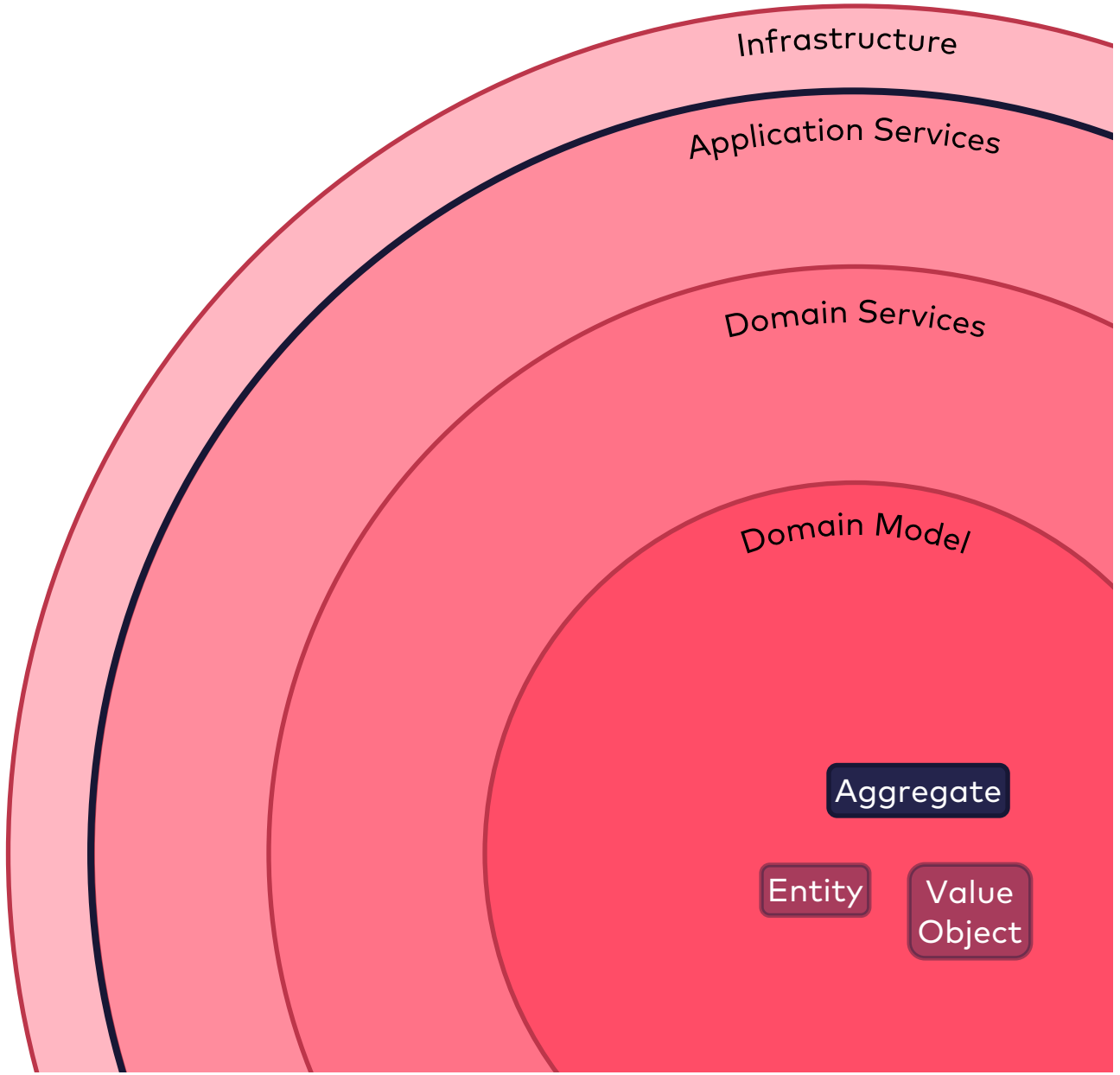



```
@ValueObject
```

Aggregate

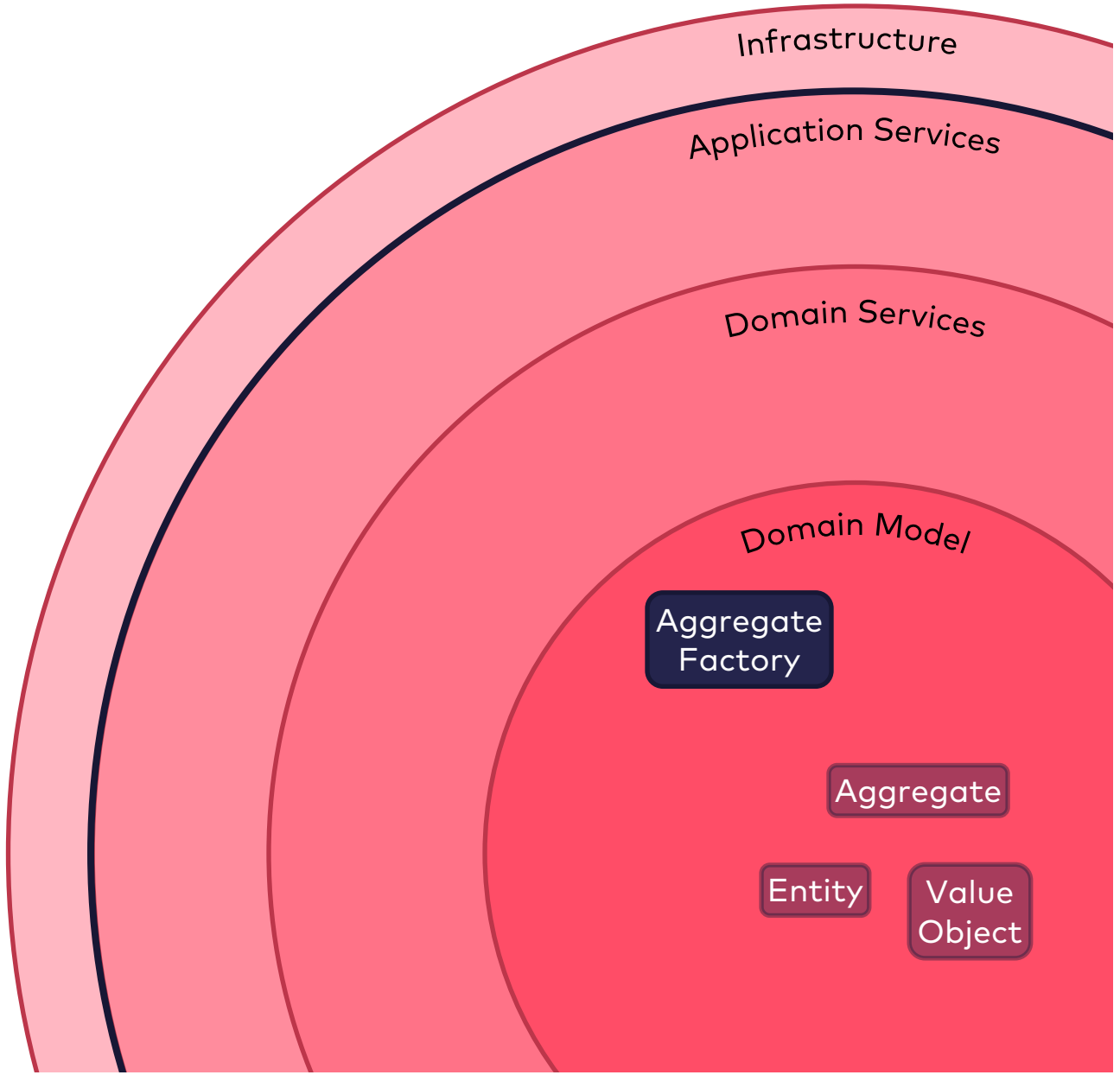
- Verhalten und notweniger Zustand
- Kleinste Einheit mit fachlicher Konsistenz
- Graph aus Entities und Value Objects
- Aggregate Root als „Gatekeeper“

```
@Aggregate
```



Factory

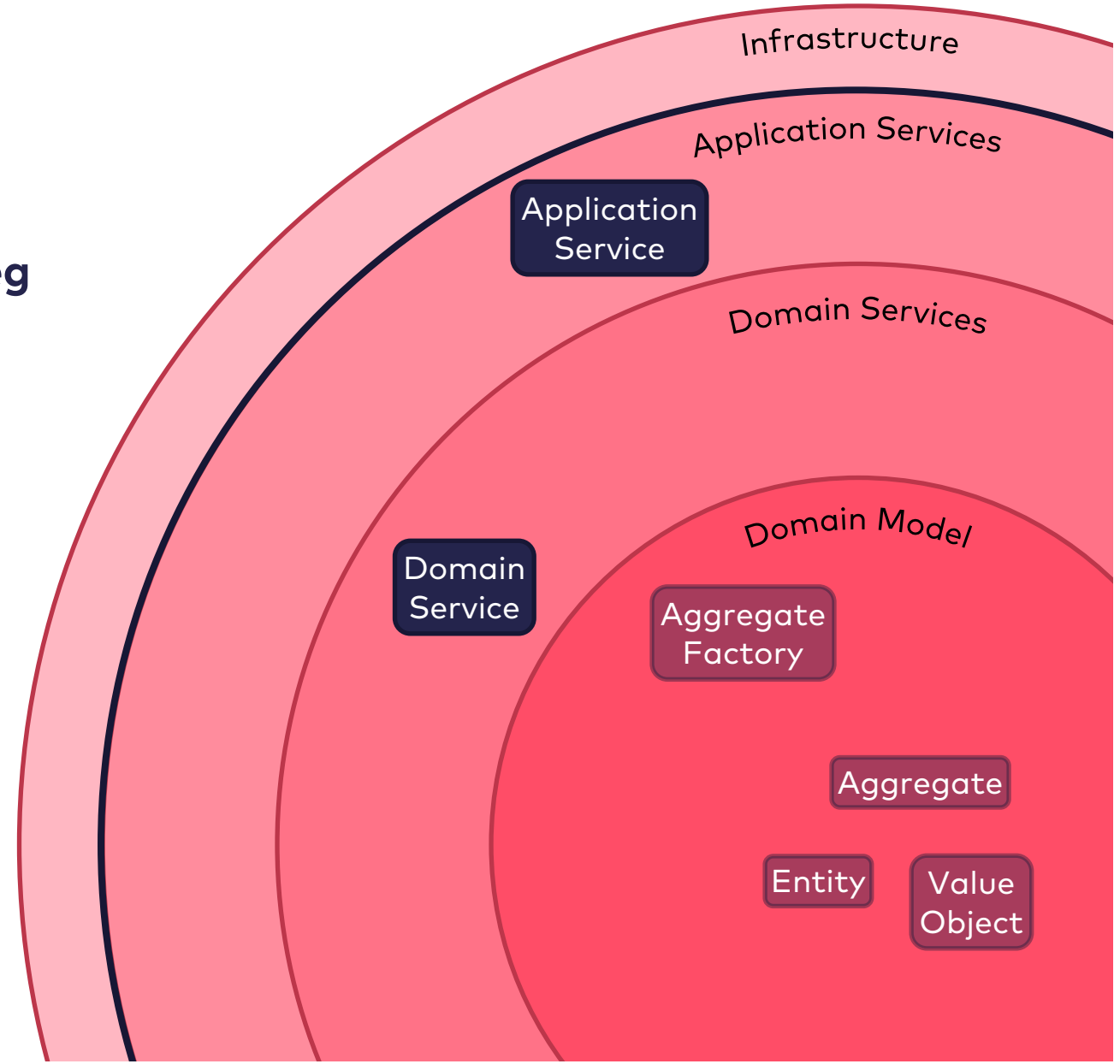
- **Kapselung von Logik zur Erzeugung von Domänen-objekten**




```
@AggregateFactory
```

Service

- **Fachlogik über Aggregate hinweg**
- **Zustandslos**




```
@ApplicationService
@DomainService
```

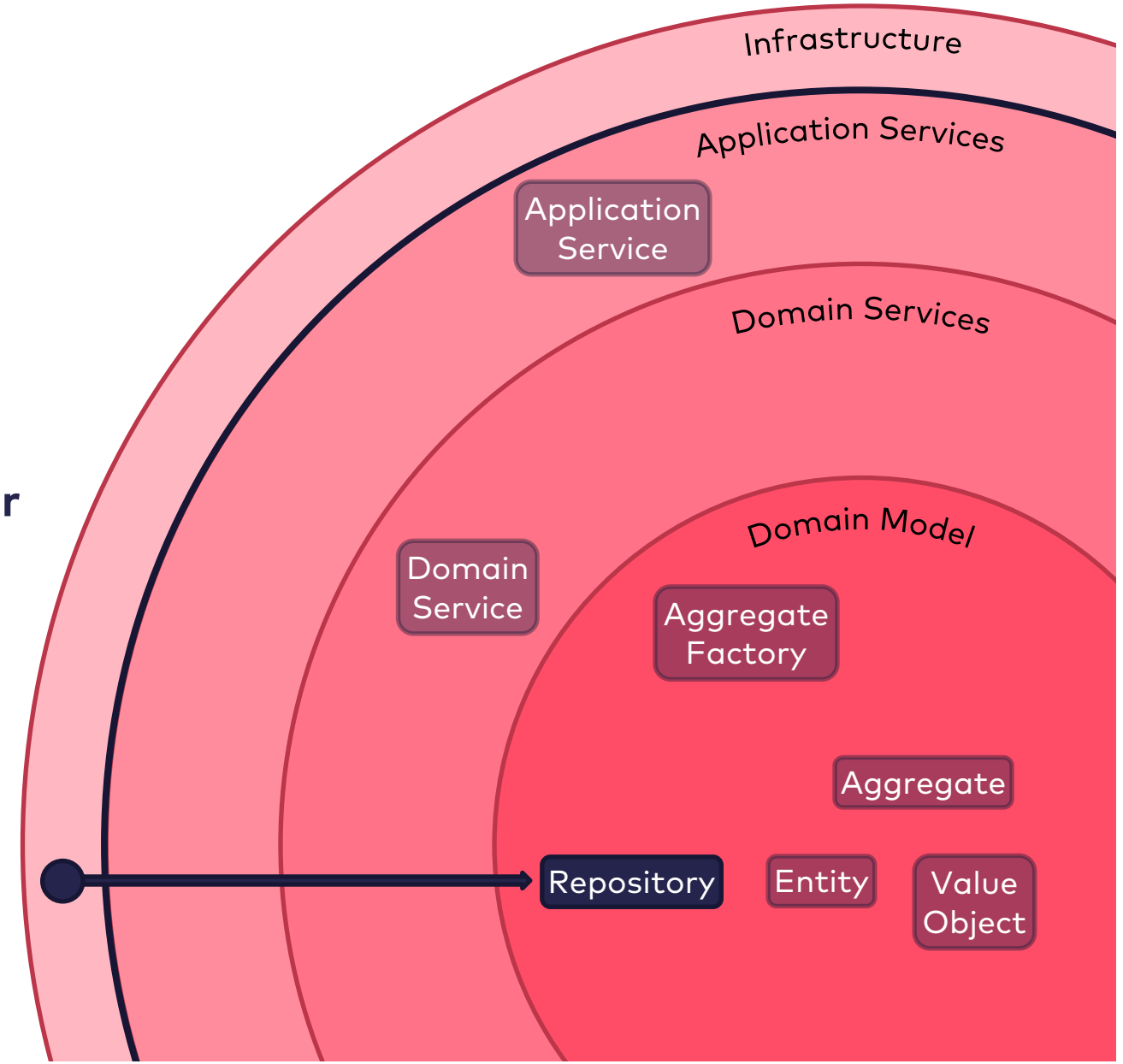
Repository

- „Büchergestell“ für Aggregate

- Interface in Domain
- Implementierung in Infrastruktur
- IoC

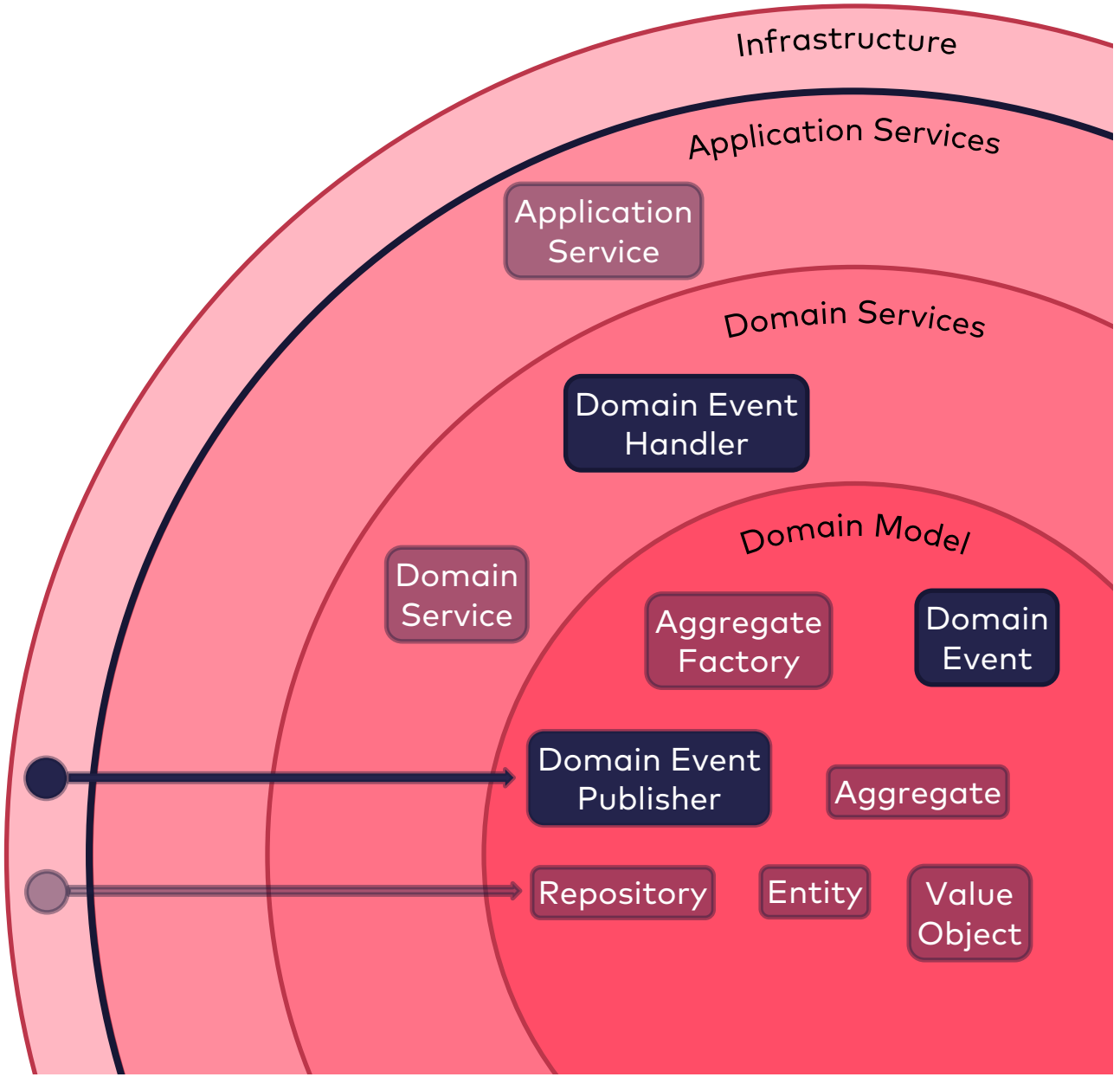
- Keine Fachlogik in Repository-Implementierung

```
@Repository
```



Domain Event

- **Fachliches Ereignis**
- **Fachlich hochwertige Semantik**
- **Unveränderlich**

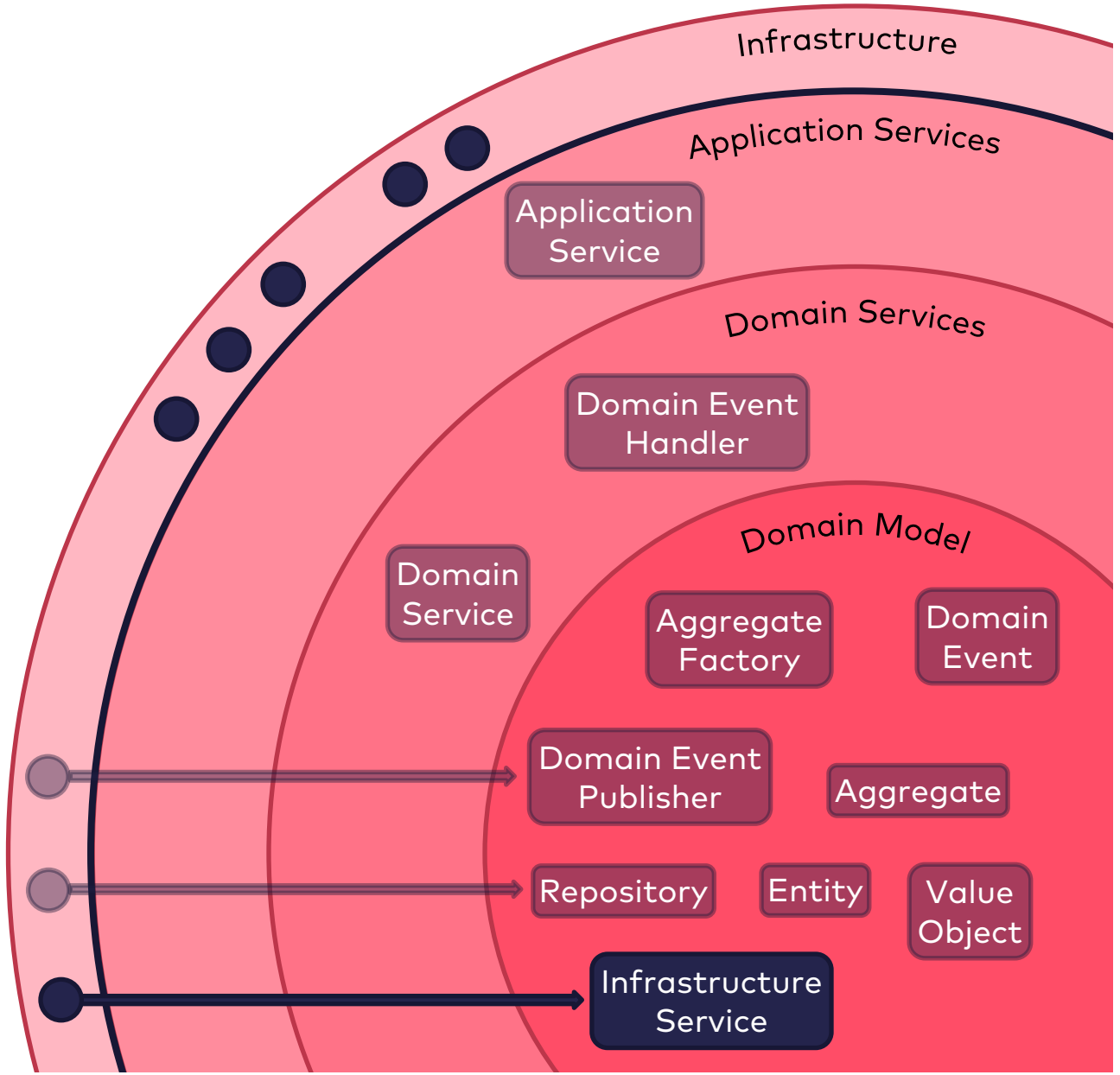



```
@DomainEventHandler
@DomainEventPublisher
@DomainEvent
```

Zusätzliche Stereotypen

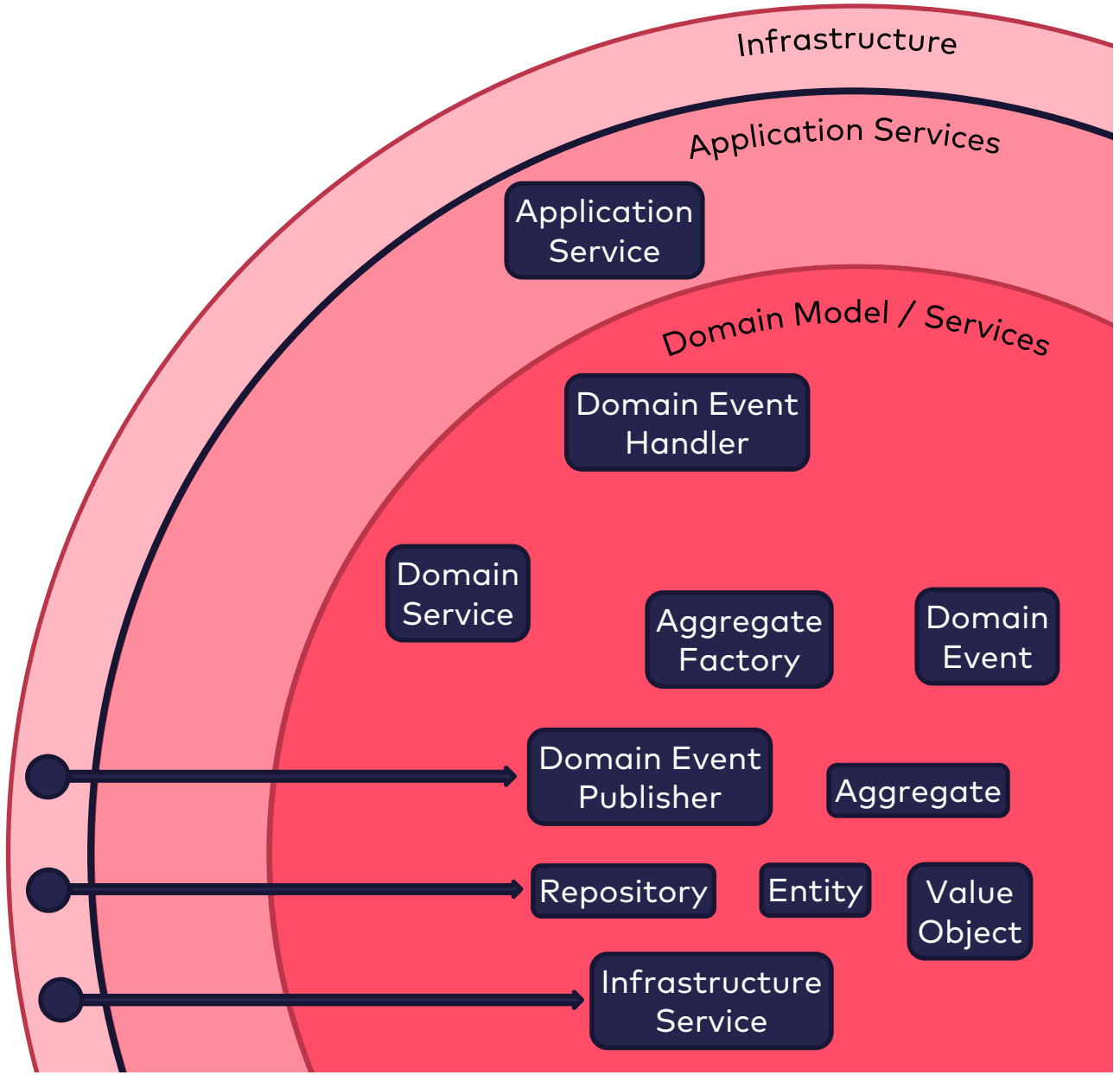
- **Infrastructure Service**
- **Web Services**
- **REST Controller**
- **Message Listener**
- **…**

```
@InfrastructureService
@WebService
@RestController
@MessageListener
@...
```

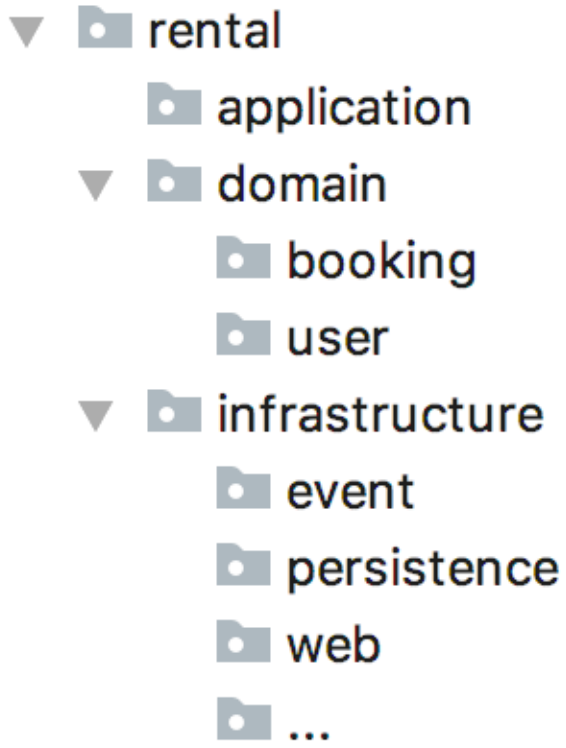


Übersicht

```
@Entity
@ValueObject
@Aggregate
@AggregateFactory
@DomainService
@ApplicationService
@Repository
@DomainEvent
@DomainEventPublisher
@DomainEventHandler
@InfrastructureService
```



Onion Architecture – Abbildung auf Packages



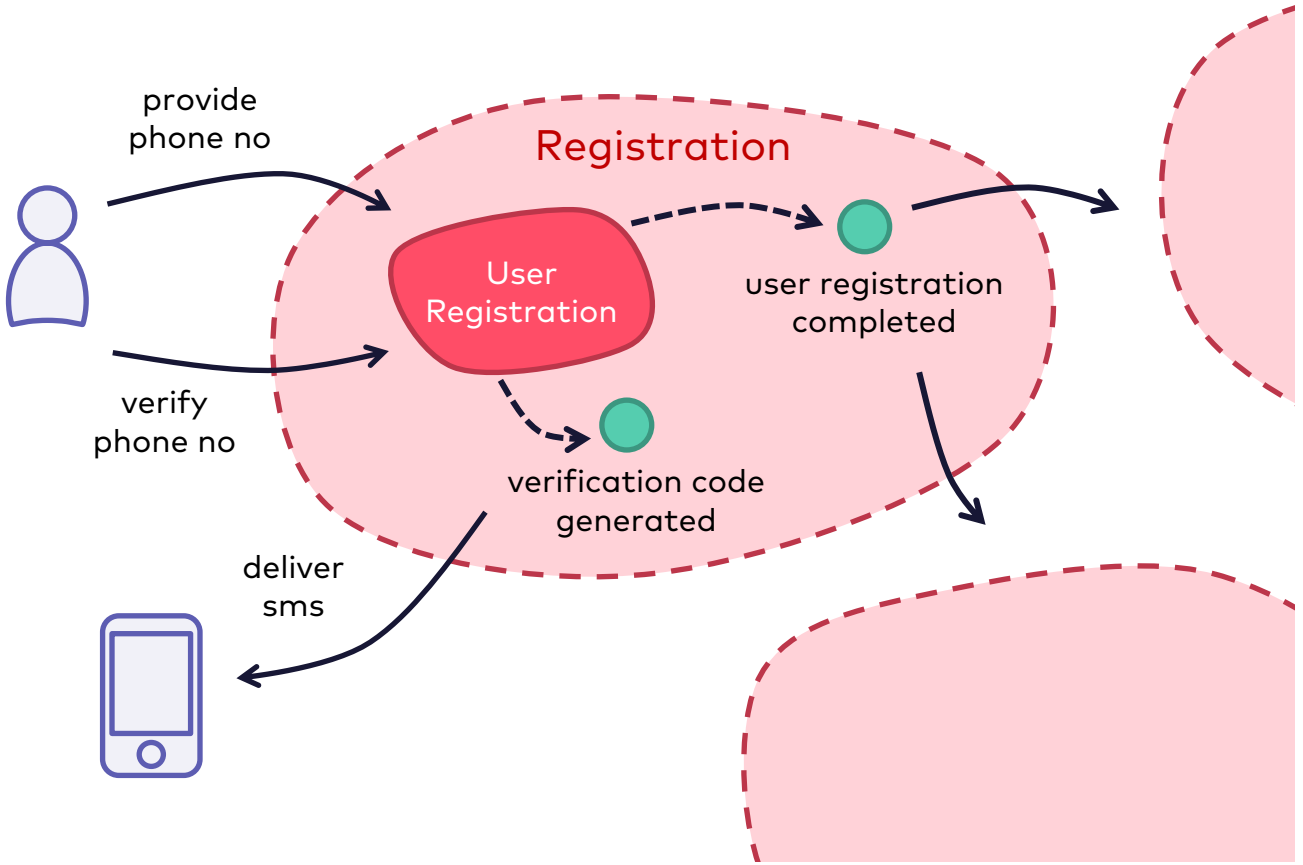
Bounded Context

- Application Services (meist ohne Subpackages)
- Domain Model + Domain Services (fachliche Subpackages)
  - Funktionalität für Buchungen
  - Funktionalität für Benutzer
- Infrastruktur („alles andere“, technische Subpackages)
  - Messaging
  - Datenbank-Zugriff
  - UI / API
  - ...

Code Walkthrough

Anwendungsfall: Benutzer registrieren

- Mobile-Nummer angeben
- SMS erhalten
- Validierungscode eingeben
- Registrierung abschliessen



Anwendungsfall: Benutzer registrieren

https://github.com/cstettler/ddd-to-the-code-workshop-sample

Verwendung von Stereotypen

Stereotypen – Verwendungsmöglichkeiten

- **Definition der "Architektursprache" über Stereotypen**
  - Angelehnt an taktischen Mustern aus DDD
  - Kommunikation im Team / in Designs

- **Mapping von Konzepten auf Code**
  - Nicht einfach nur „Code"
  - Jede Komponente hat ausgezeichnete Bedeutung

Umsetzung von technischen Konzepten mit Stereotypen

- **Framework-Konfiguration**
  - Detektieren von Komponenten → Component Scan für Injection Container
- **Persistenz von Aggregates**
  - Generierung von Default Constructor
  - Generierung von equals() / hashCode() für First Level Cache
- **Transaktionen**
  - Definition von Transaktionsgrenzen auf Application Services
  - Sicherstellen von aktiver Transaktion bei Zugriff auf Repository

Architecture Governance mit Stereotypen

- Beschreibung von Konzepten basierend auf Stereotypen
- Definition von Regeln basierend auf Konzepten
- Überprüfung von Eigenschaften und Abhängigkeiten
- Visualisierung von Abhängigkeiten

https://www.innoq.com/de/blog/architecture-governance-mit-stereotypen/

Key Takeaways

Onion Architecture fördert Trennung von Fachlichkeit und Technologie, verbessert Verständlichkeit und erhöht Testbarkeit

Stereotypen ermöglichen klare Identifikation und Kommunikation von Architekturelementen in Code-Basis – basierend auf DDD-Mustern

Dank Stereotypen kann fachlicher Code weitestgehend von technischen Aspekten freigehalten werden (zum Preis von mehr „Magic“ wie AOP)

Alternative und ergänzende Themen

- CRUD-Architektur für Bounded Contexts mit trivialer Fachlichkeit
- Query-/Read-Model-basierte Architektur für primär lesende Fachlichkeit
- CQRS für Trennung von Write- und Read-Model
- Event Sourcing als Implementierungsstrategie für Aggregate-Persistenz
- …

Weiterführende Links

- DDD mit Onion Architecture (et. al.)
  https://www.innoq.com/de/blog/ddd-mit-onion-architecture-umsetzen/

- Übersicht Hexagonal Architecture
  https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/

- Bike Sharing Sample Code auf GitHub
  https://github.com/cstettler/ddd-to-the-code-workshop-sample

- DDD-To-The-Code Training (2 -3 Tage)
  https://www.innoq.com/de/trainings/ddd-to-the-code/

DDD mit Onion Architecture & Stereotypes

Vielen Dank !!!

Christian Stettler
christian.stettler@innoq.com