Architektur von Softwaresystemen

Keep it stupid simple

Inhaltsverzeichnis

Agile Methoden
Clean Code
Clean Architecture
Domain Driven Design
SOLID Principles
Conventions over configuration
Don’t repeat yourself
Service/Container Architecture

Inhaltsverzeichnis

Architektur

Design Pattern

Domain Model

Infrastruktur

Use Cases

API/View

Entity
Value Object
Aggregate
Ubiquitous Language (Allgegenwärtige Sprache)

Agile Methoden
Clean Code
Clean Architecture
Domain Driven Design
SOLID Principles
Conventions over configuration
Don’t repeat yourself
Service/Container Architecture

CQRS
MediatR
Specification Pattern
Repository Pattern
MVVM vs MVC

Inhaltsverzeichnis

Architektur

Design Pattern

Domain Model

Infrastruktur

Use Cases

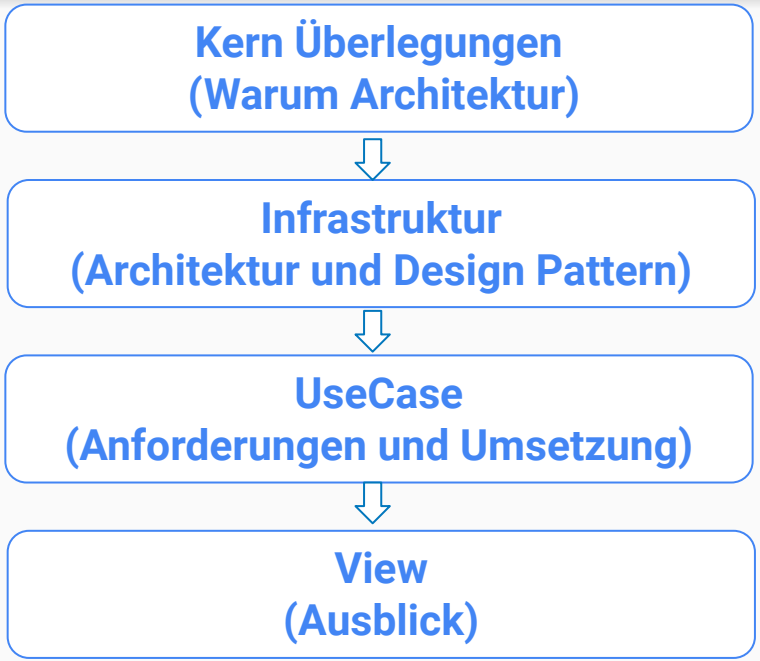
API/View

Entity
Value Object
Aggregate
Ubiquitous Language (Allgegenwärtige Sprache)

Agile Methoden
Clean Code
Clean Architecture
Domain Driven Design
SOLID Principles
Conventions over configuration
Don’t repeat yourself
Service/Container Architecture

CQRS
MediatR
Specification Pattern
Repository Pattern
MVVM vs MVC

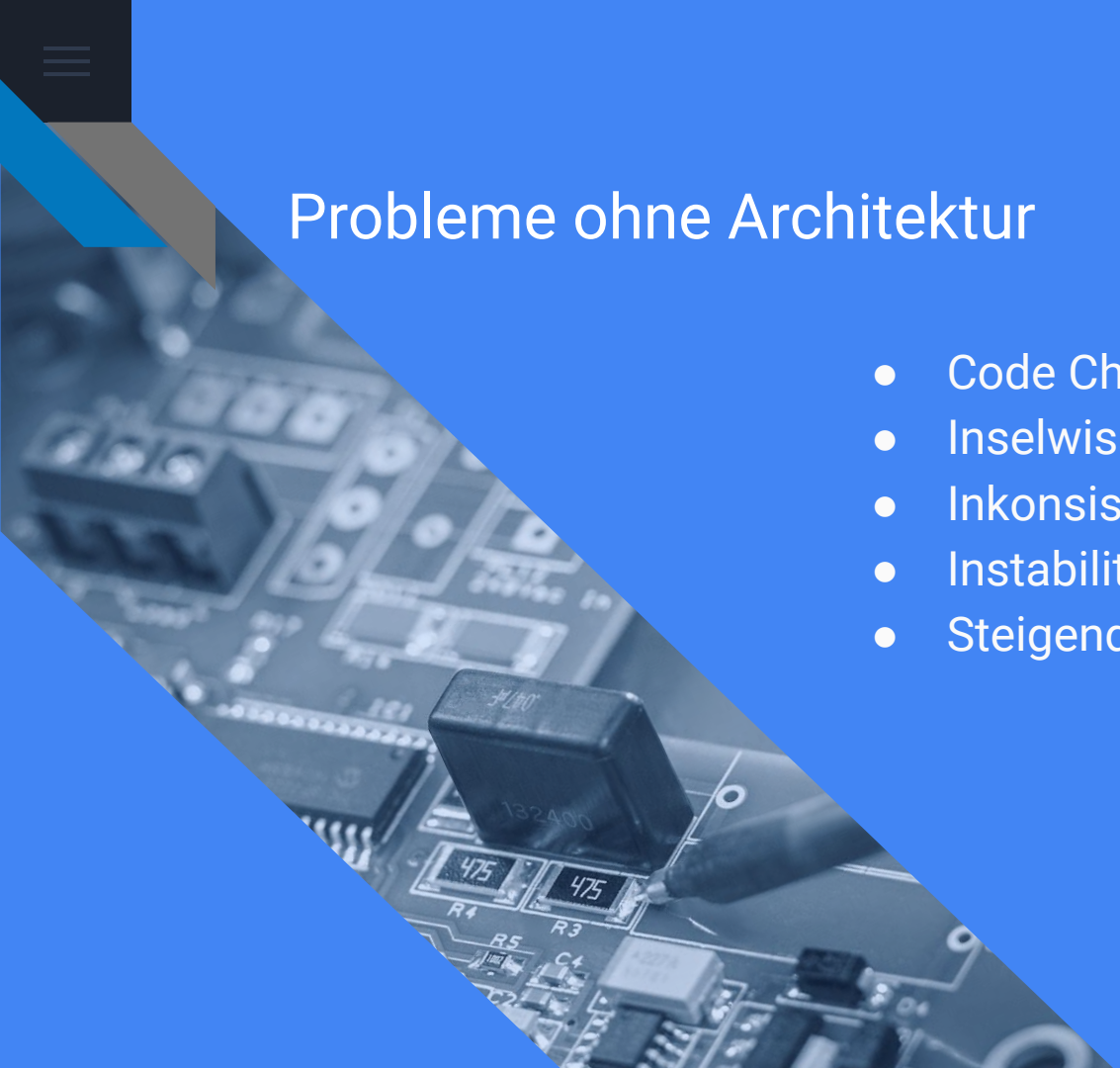
Übersicht


Kern Überlegungen
(Warum Architektur)
Infrastruktur
(Architektur und Design Pattern)
UseCase
(Anforderungen und Umsetzung)
View
(Ausblick)

Warum Architektur?

- Erweiterbarkeit
- Skalierbarkeit
- Flexibilität
- Modularität
- Testbarkeit
- … KISS!

Probleme ohne Architektur



- Code Chaos
- Inselwissen
- Inkonsistenz
- Instabilität
- Steigender Aufwand bei Features

Lösung: Architektur

- Bessere Wartbarkeit
- Kostenreduktion
- Skalierbarkeit
- Erhöhte Flexibilität
- Bessere Codequalität

Infrastruktur

- Bekannte Muster für bekannte Probleme
- Software soll selbst dokumentiert sein

Clean Architecture - Mehr als nur saubere Fenster

- Rise of Clean Architecture
- Robert C. Martin
- Kombination aus SOLID, Design-Pattern und Agilen Methoden
- Onion Architecture, Ports and Adapters

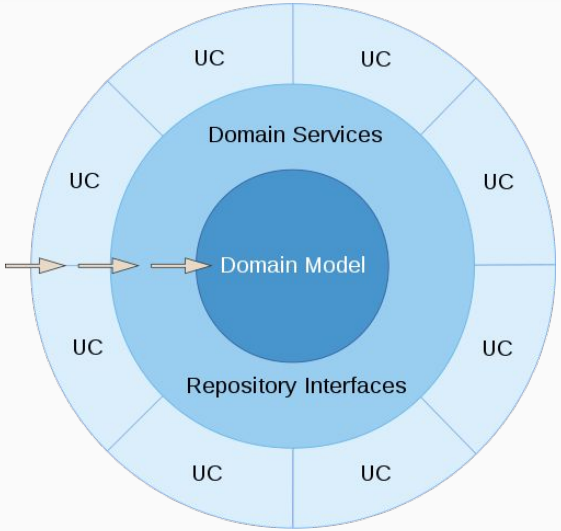
“es hilft dabei, den digitalen Schrank aufzuräumen und Freude am Programmieren zu finden”

“Wir bringen Ordnung in das Chaos und verhindern, dass unser Code aussieht wie ein Haufen LEGO-Steine nach einem Kindergeburtstag.”

Clean Architecture

Abhängigkeiten werden zur Laufzeit ermittelt

Einfacher austausch von Code Teilen

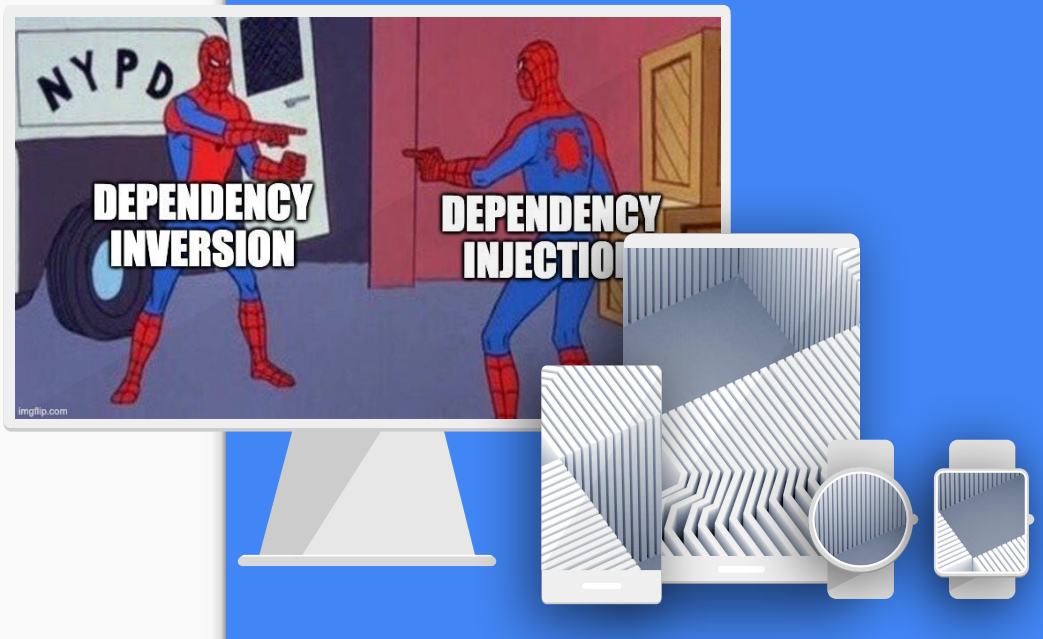


Domain Model hat keine Abhängigkeiten

Äußere Schichten greifen auf innere Schichten zurück

Dependency Injection

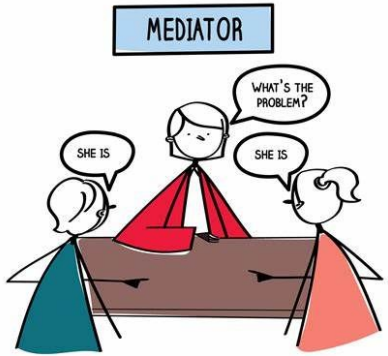
- Lösung von Abhängigkeiten
- Lockere Kopplung
- Erleichterung von Tests



CQRS

Die Teilung von Schreiben und Lesen, ohne sich zu verheddern

- Aufteilung des Objektmodells in lesen und schreiben
- Anpassbarkeit an veränderte Geschäftsordnungen

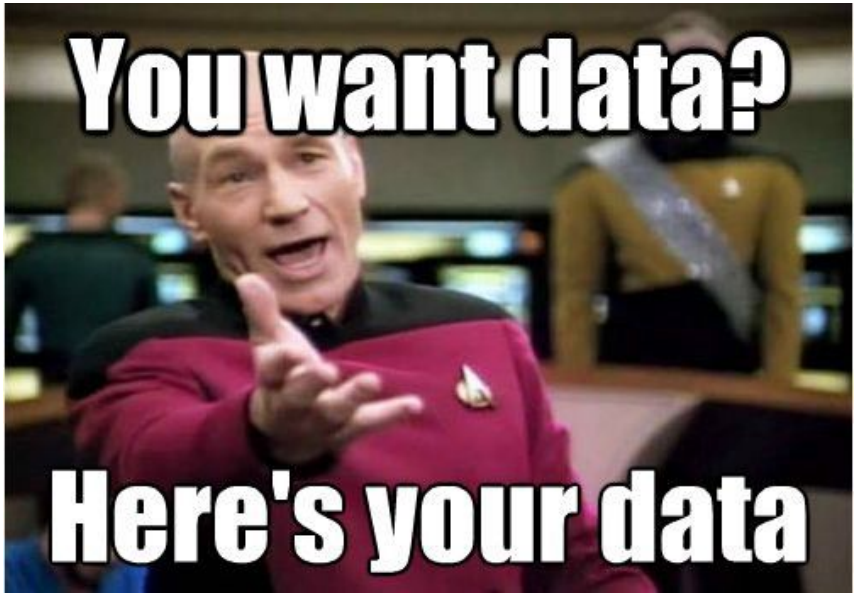


CQRS ist eine gespaltene Persönlichkeit - eine Seite schreibt, die andre liest, und beide sind glücklich

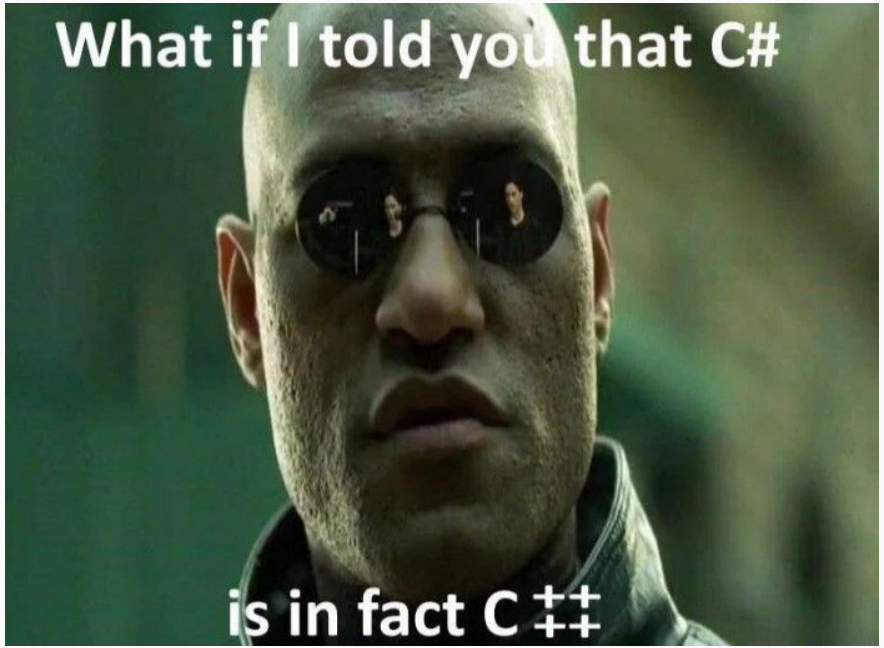
Repository Pattern

Wenn der Datenzugriff zum Selbstläufer wird

- Jedes Datenobjekt hat seine eigene Schnittstelle
- Klare Trennung von Domain und Infrastruktur Schicht
- Mit Spezifikationen des Domain Models erweiterbar

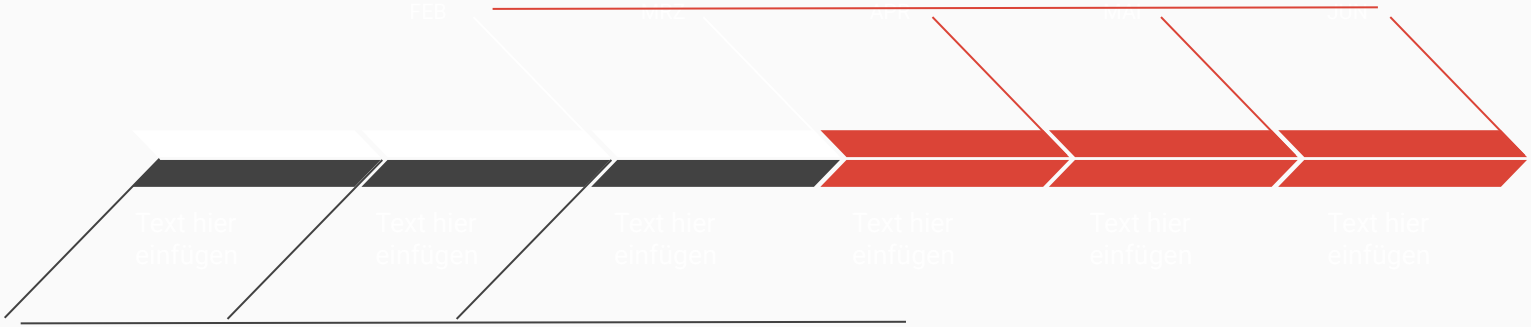


UseCase - Umsetzung oder wie man ein Superheld wird



View - Ausblick nach dem Einblick

- Lesbarkeit
- Erweiterbarkeit
- Testbarkeit



- Anfänglicher hoher Aufwand
- Hohe Lernkurve
- Zusätzliche Komplexität

Vielen Dank!

- "Domain-Driven Design: Tackling Complexity in the Heart of Software" von Eric Evans
- "Patterns of Enterprise Application Architecture" von Martin Fowler
- "Clean Architecture: A Craftsman's Guide to Software Structure and Design" von Robert C. Martin
- "Building Microservices" von Sam Newman Gihub:
- https://github.com/ardalis Steve Smith (MS MVP)
- https://github.com/dotnet-architecture/eShopOnWeb Net:
- https://deviq.com/design-patterns/repr-design-pattern
- https://blog.cleancoder.com/uncle-bob/2011/11/22/Clean-Architecture.html