MHP
A PORSCHE COMPANY

Clean Architecture: fundament wartbarer Software oder nur ein Hype?

Fedor Zholud

Über mich

https://www.linkedin.com/in/fedor-zholud/

https://github.com/FedorZholud

Mehrere Jahre Berufserfahrung und an verschiedenen Projekten und Produkten beteiligt.

Habe Systeme von Grund auf aufgebaut und auch bestehende Anwendungen weiterentwickelt.

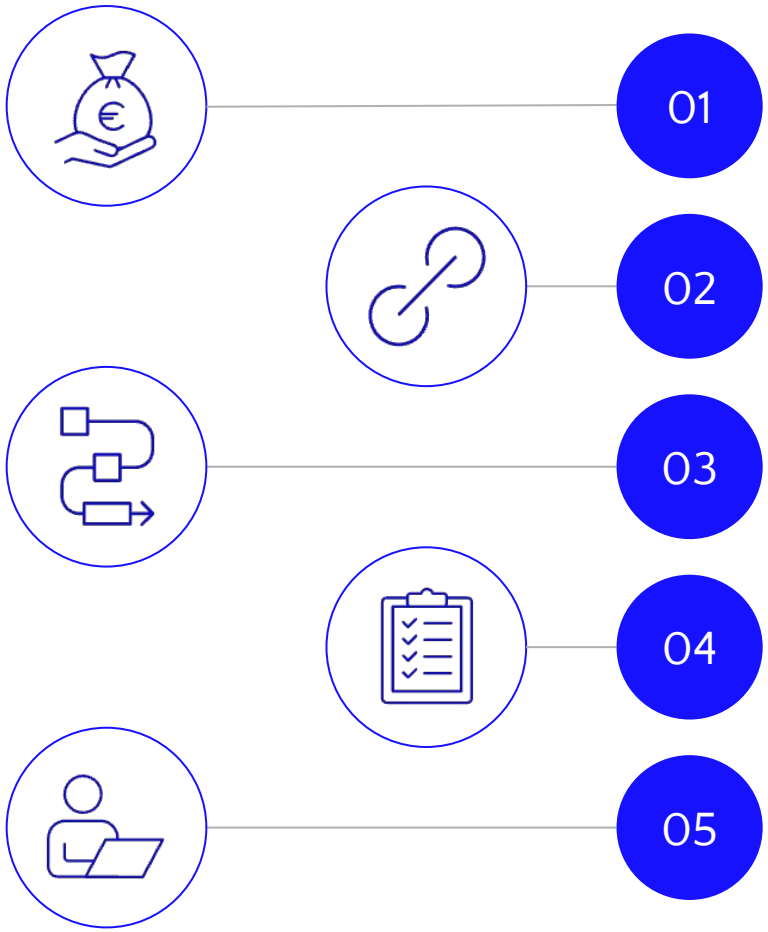
Dabei habe ich sowohl klassische Systeme als auch Serverless-Lösungen entwickelt.

Begeistert von Architekturansätzen und -patterns

Aktuell arbeite ich bei MHP – A Porsche Company.

Architektonische Erosion

Typische Symptome architektonischer Erosion



01 Änderungen werden “teuer”

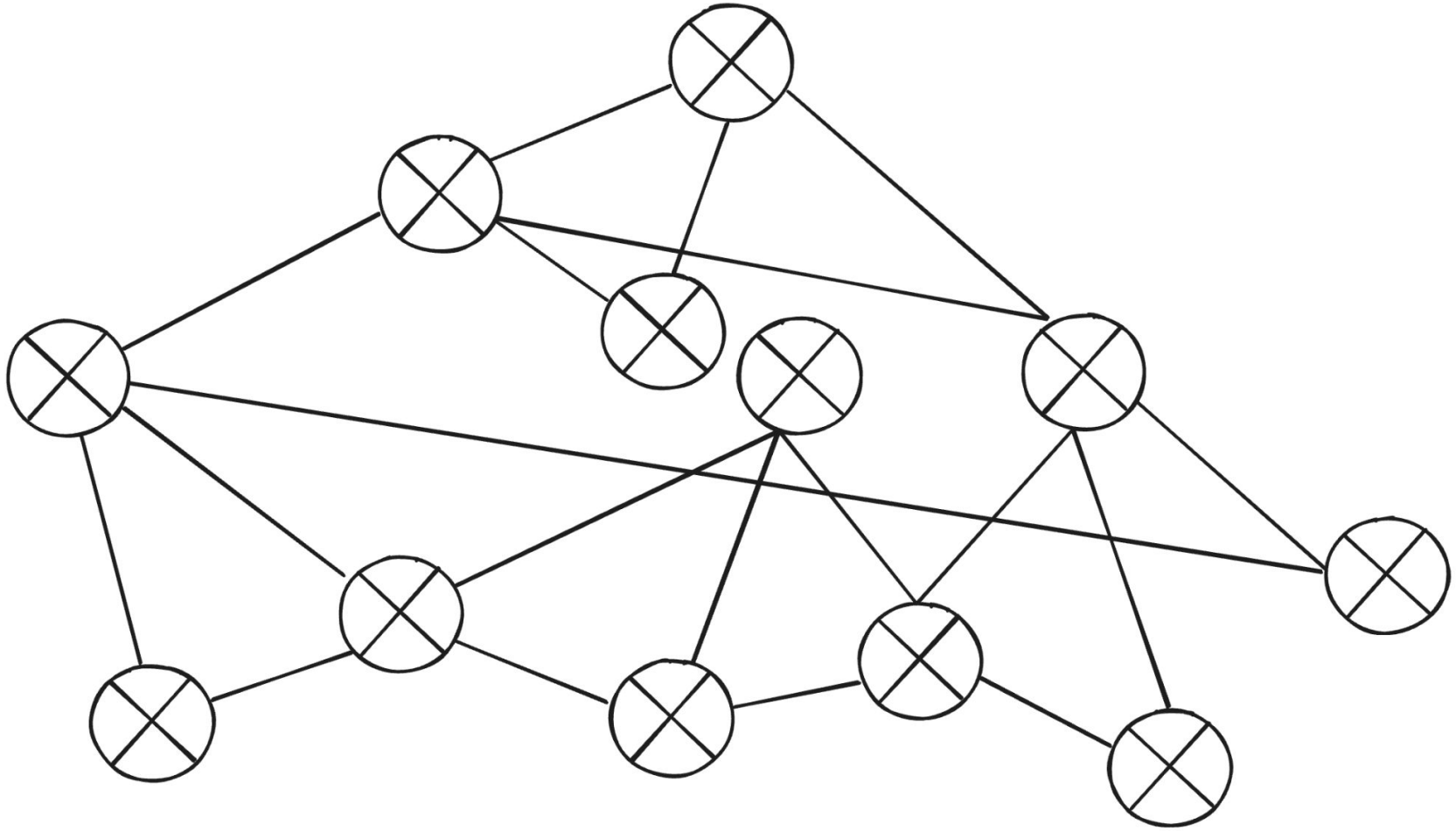
02 Zu starke Kopplung

03 Unerwartete Seiteneffekte

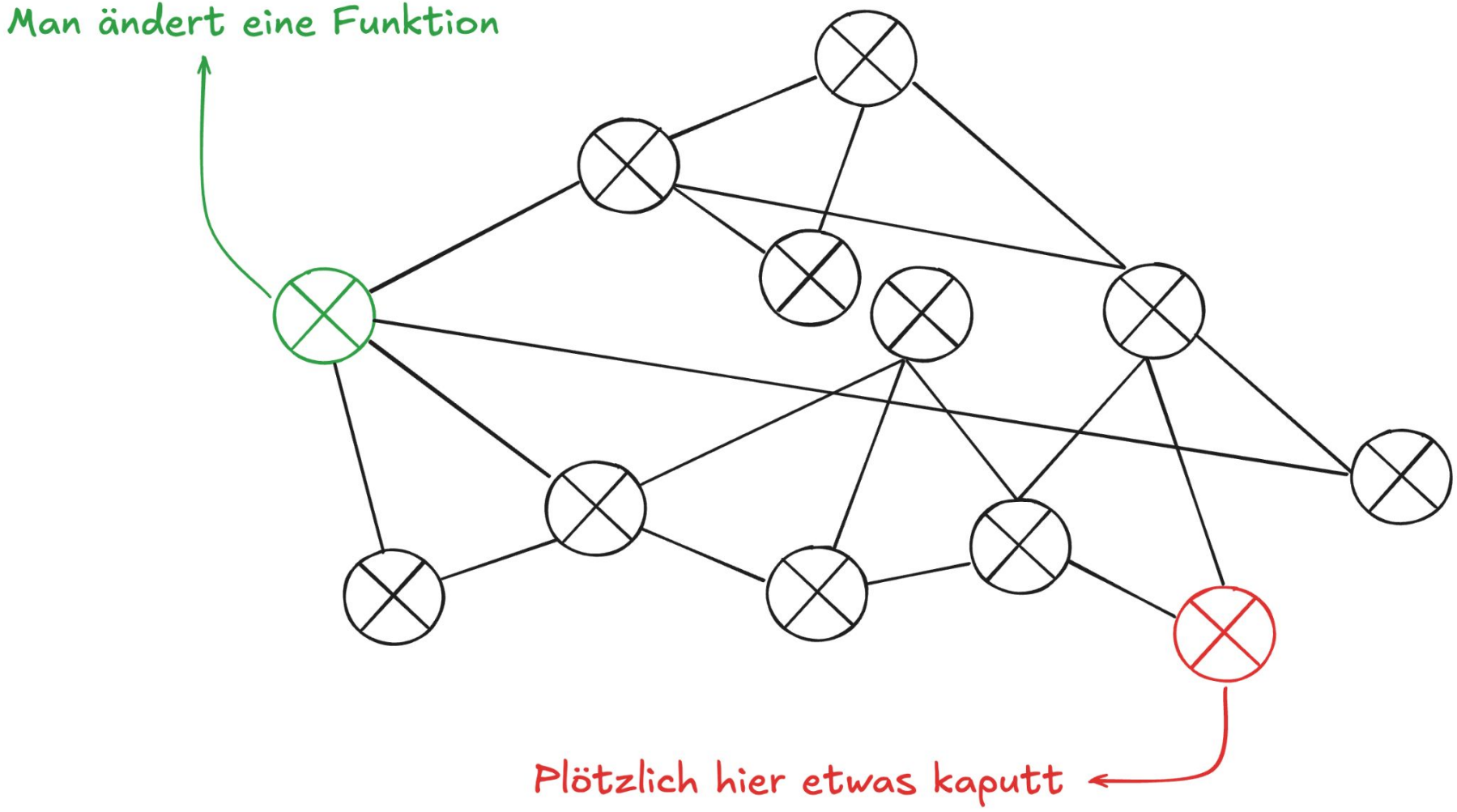
04 Testbarkeit nimmt ab

05 Onboarding wird mühsam

Zu starke Kopplung



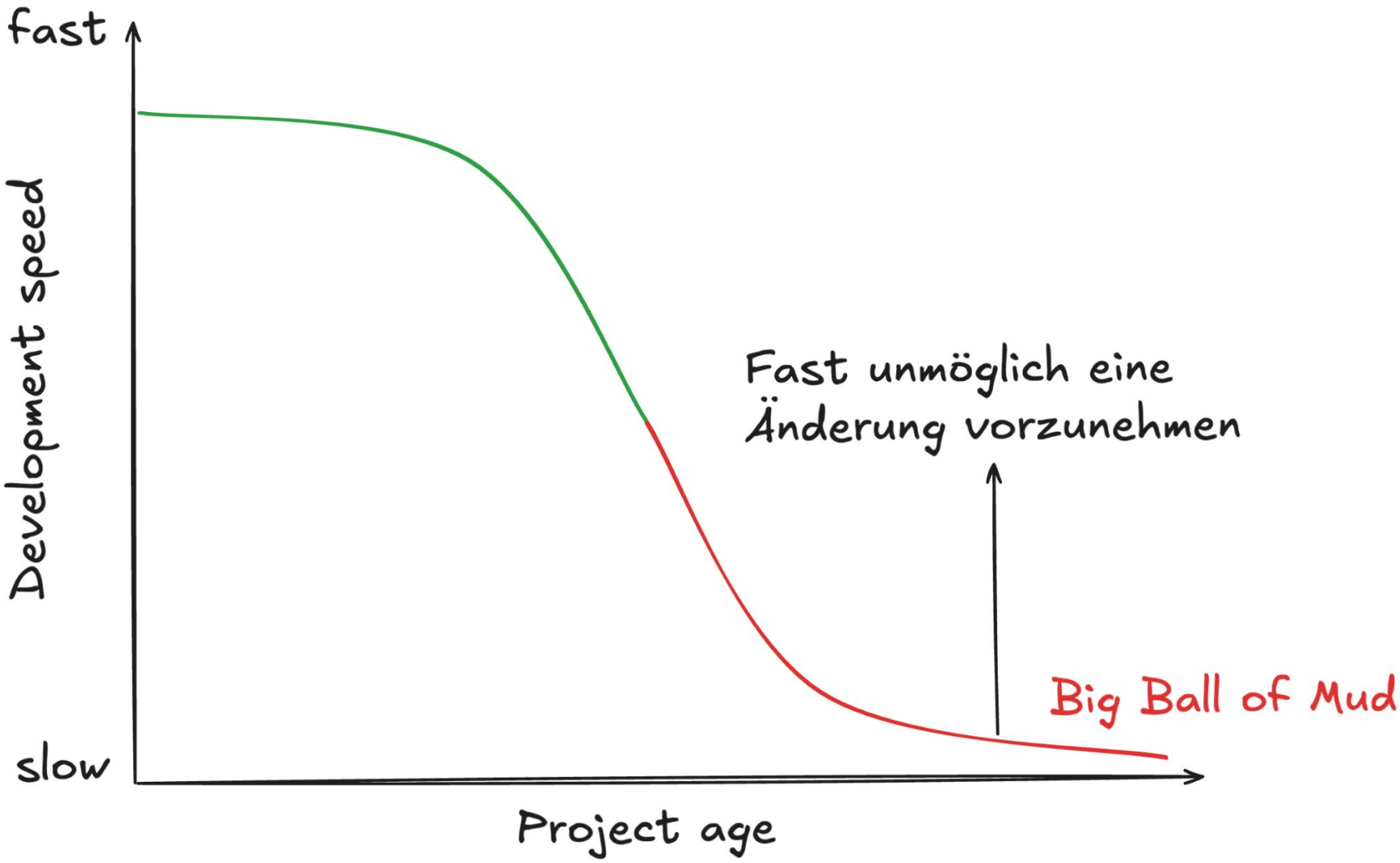
Unerwartete Seiteneffekte



Unerwartete Seiteneffekte

Source: https://youtu.be/PD00RxXEiwo

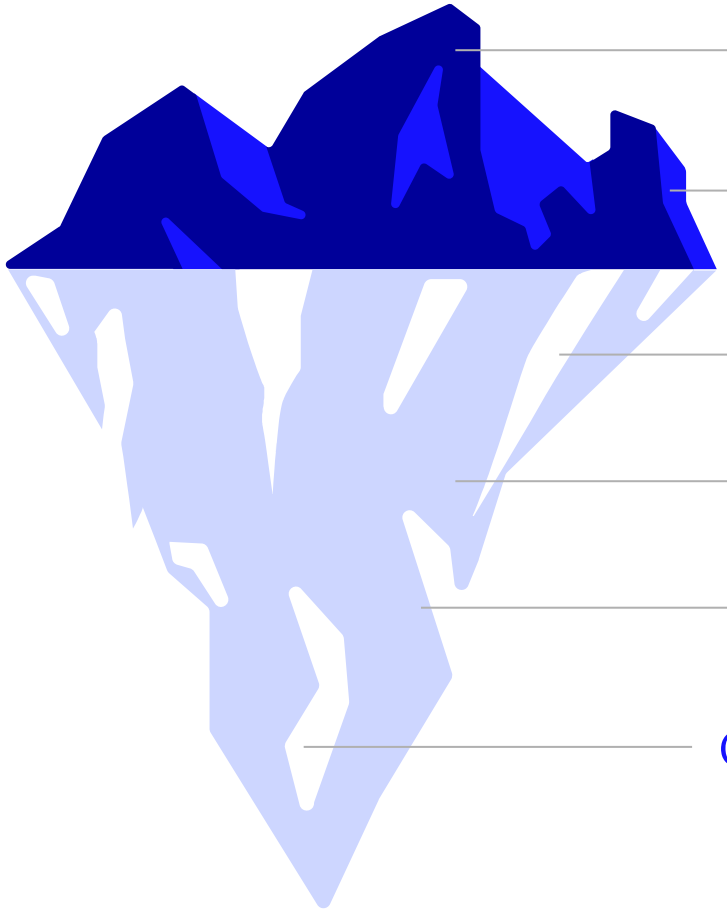
Big Ball of Mud



Warum entstehen Big Balls of Mud?

MHP
A PORSCHE COMPANY

Wesentliche Probleme



Niedrige Kohäsion

Zu starke Kopplung

Zeitdruck

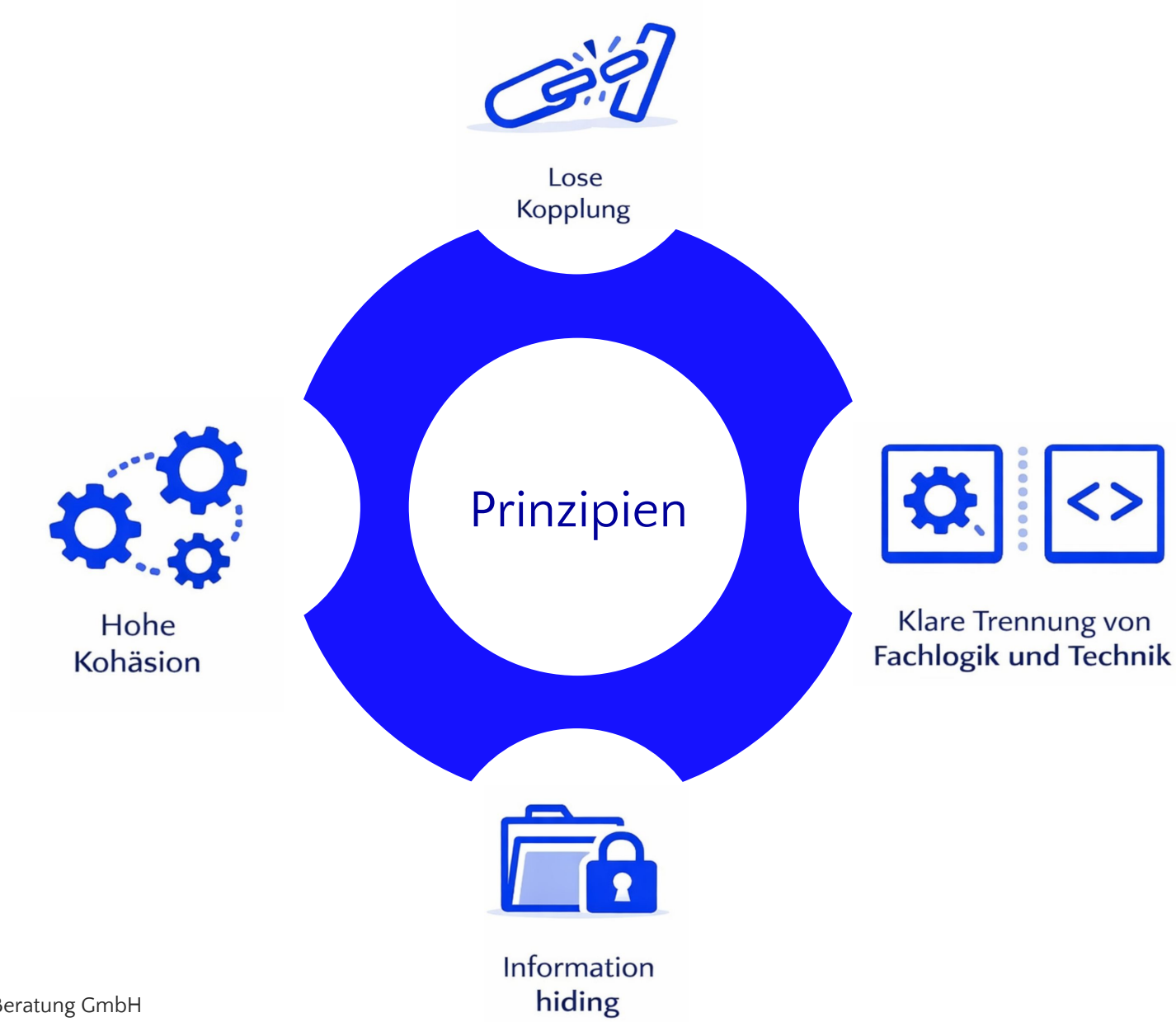
Vermischung von Fachlogik und Technik

Fehlende oder unklare Abhängigkeitsregeln

Gruppenzwang

Typische Ursachen

Architekturprinzipien für Wartbarkeit und Erweitbarkeit



Unterschiedliche Architekturstille im Überblick

Hexagonal Architecture

Alistair Cockburn

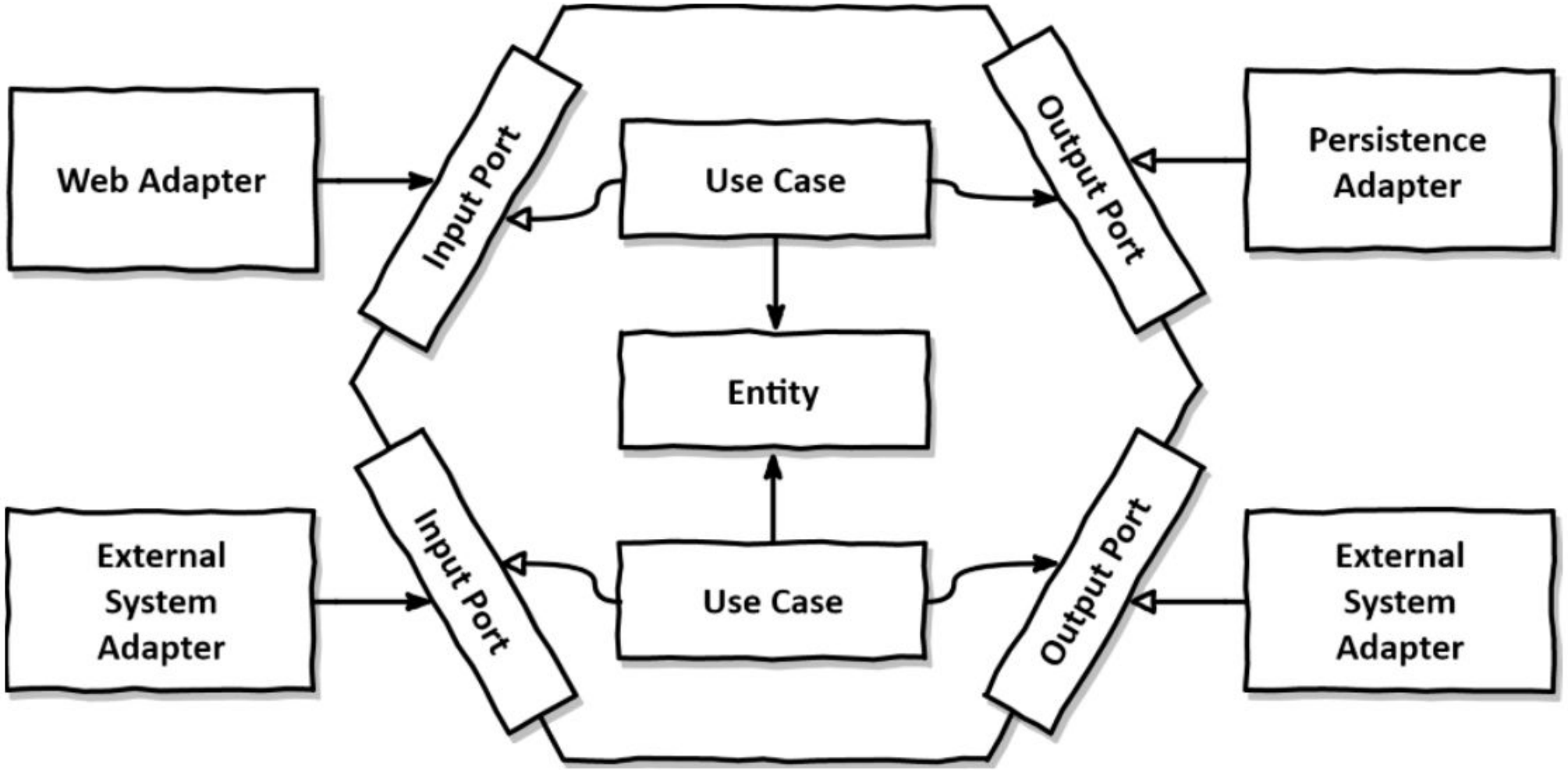
Onion Architecture

Jeffrey Palermo

Clean Architecture

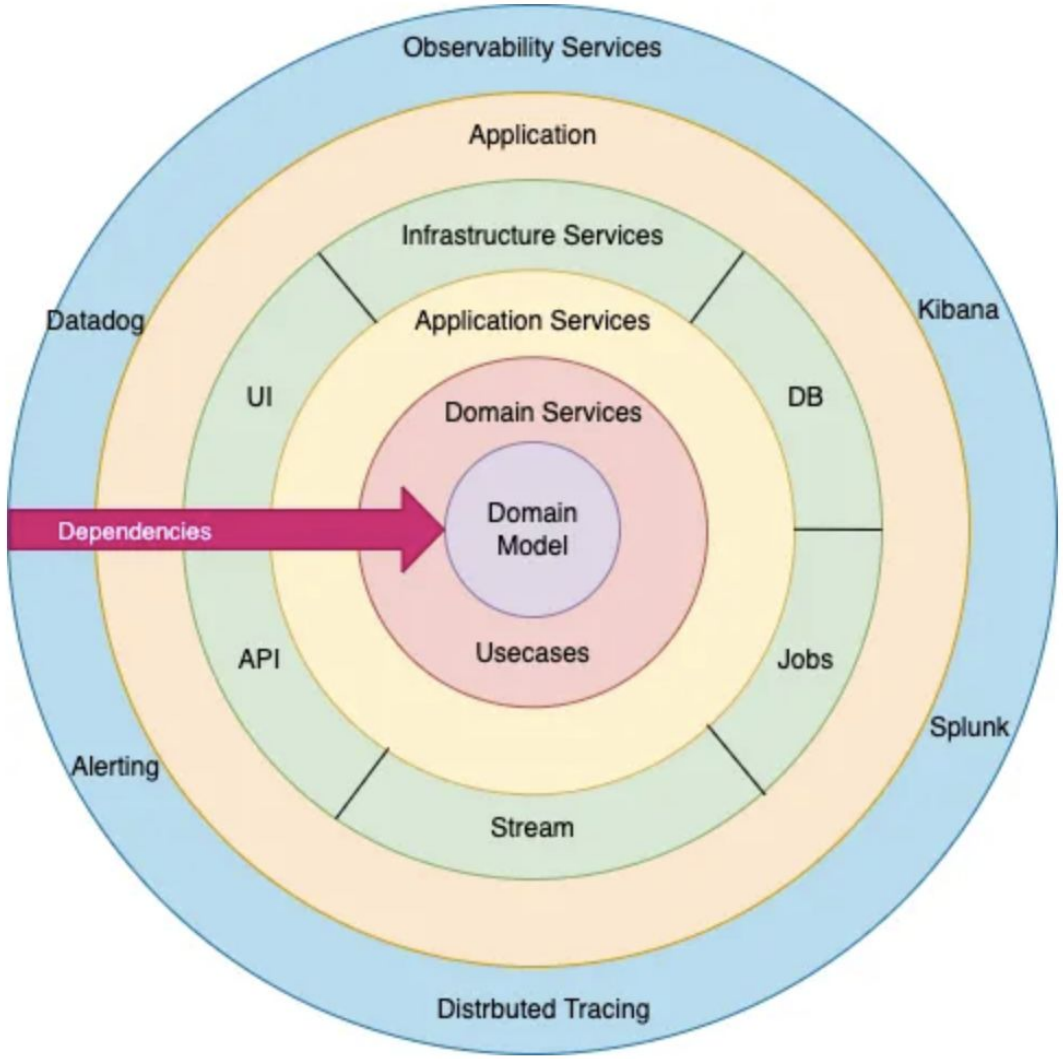
Robert C. Martin
(Uncle Bob)

Hexagonal Architecture



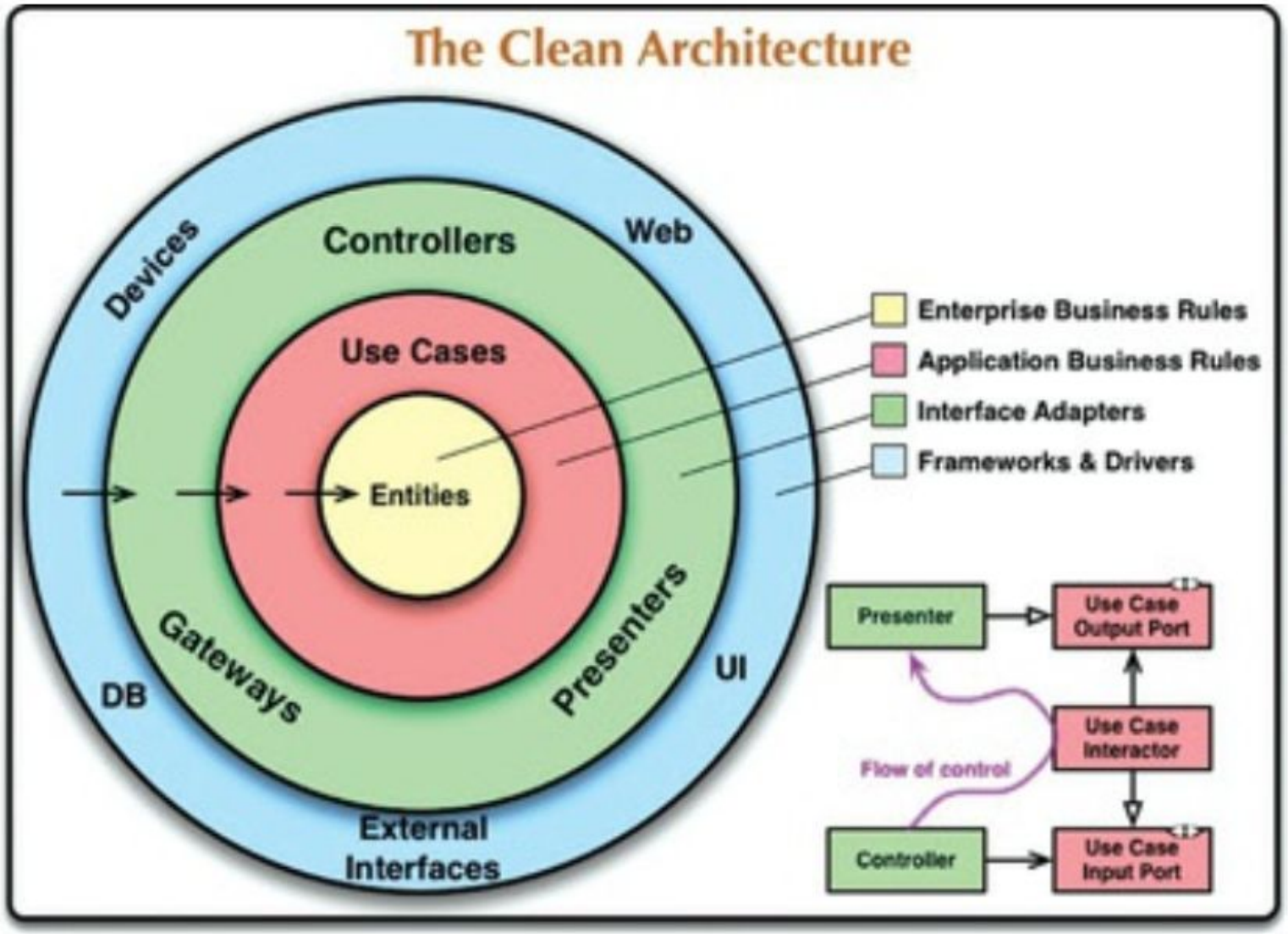
Source: Tom Hombergs — *Get Your Hands Dirty on Clean Architecture*

Onion Architecture



Source: https://medium.com/expedia-group-tech/onion-architecture-deed8a554423

Clean Architecture



Source: Robert C. Martin - *Clean Architecture*

Gemeinsame Idee hinter allen drei Architekturen

Die Fachlogik soll unabhängig sein von Frameworks

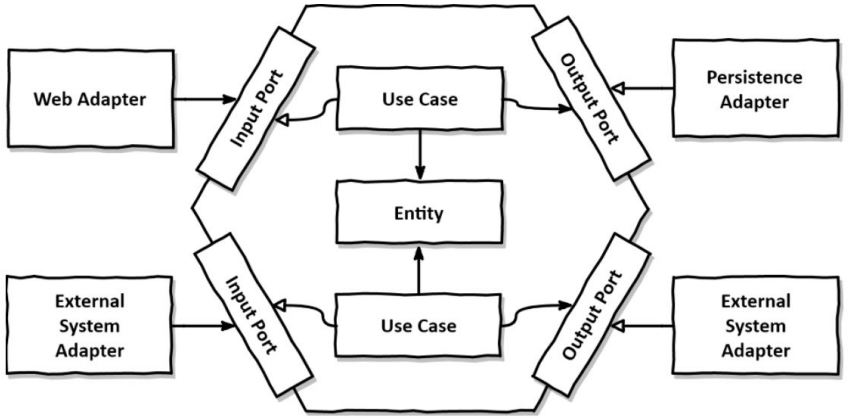
Unabhängig von der Datenbank

Unabhängig von externen Systemen

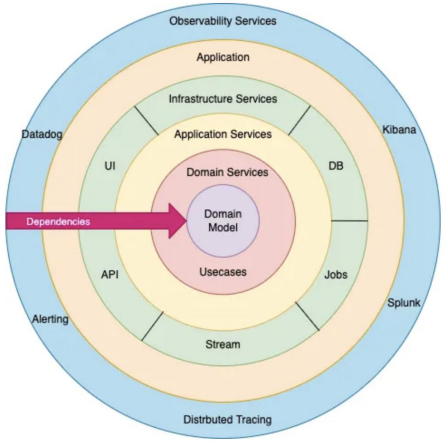
Unabhängig von der Benutzeroberfläche

Soll einfach testbar sein

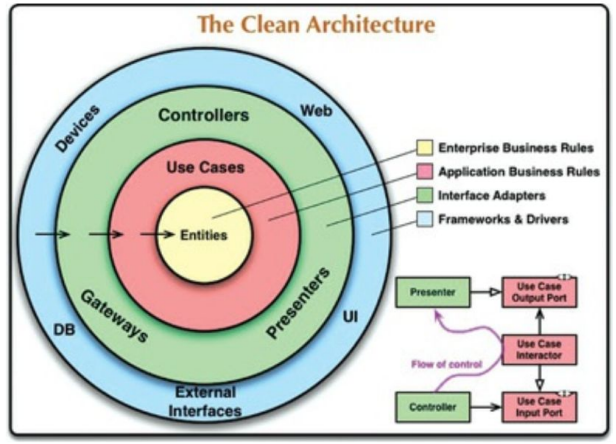
*"Name it **Clean Architecture**, **Hexagonal Architecture** or **Ports and Adapters** Architecture - by inverting our dependencies so that the **domain code** has **no dependencies** to the outside we can decouple our domain logic from all those persistence and UI specific problems and reduce the number of reasons to change throughout the codebase."* - Get Your Hand Dirty on Clean Architecture, Tom Hombergs



=

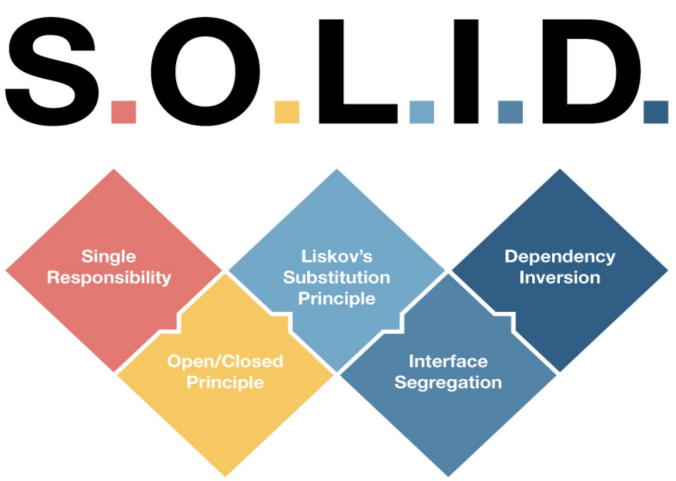


=



Grundlegende Prinzipien

SOLID

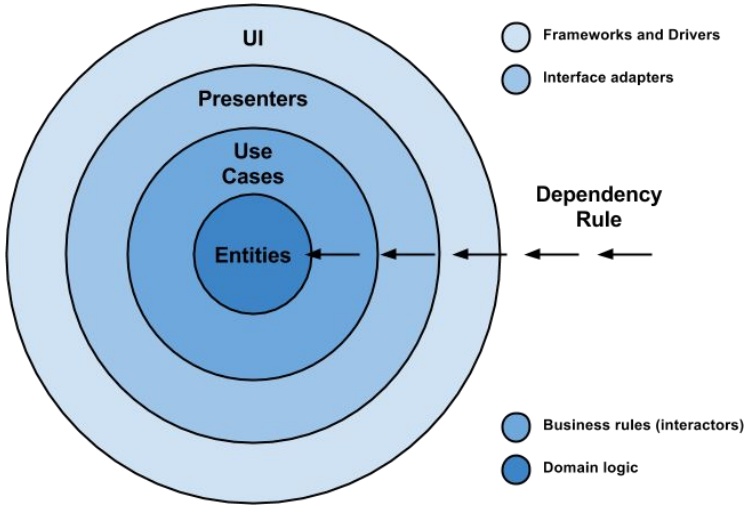


Source: https://androidschool.ru/2024/02/16/solid-in-android-development/

Zwei besonders wichtig:

- Single Responsibility Principle
- Dependency Inversion Principle

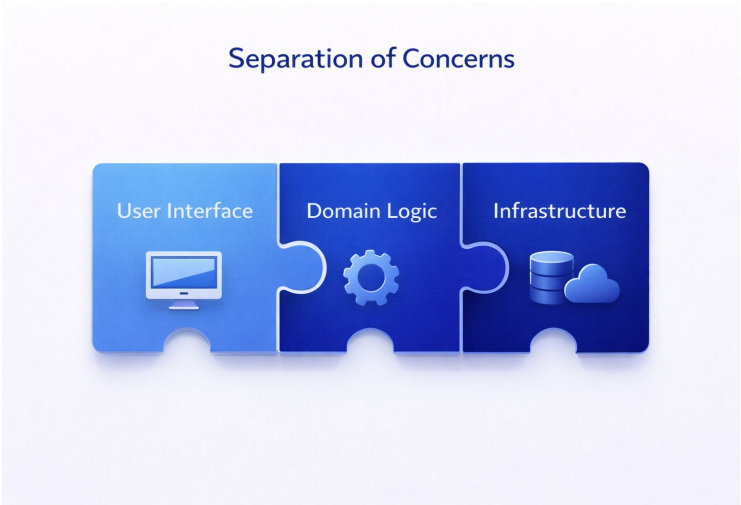
Dependency Rule



Source: https://www.aalpha.net/wp-content/uploads/2022/07/Principles-of-Clean-Architecture.png

- Abhängigkeiten dürfen nur nach innen
- Äußere Schichten dürfen innere Schichten kennen
- Innere Schichten dürfen nichts über Äußere Schichten wissen

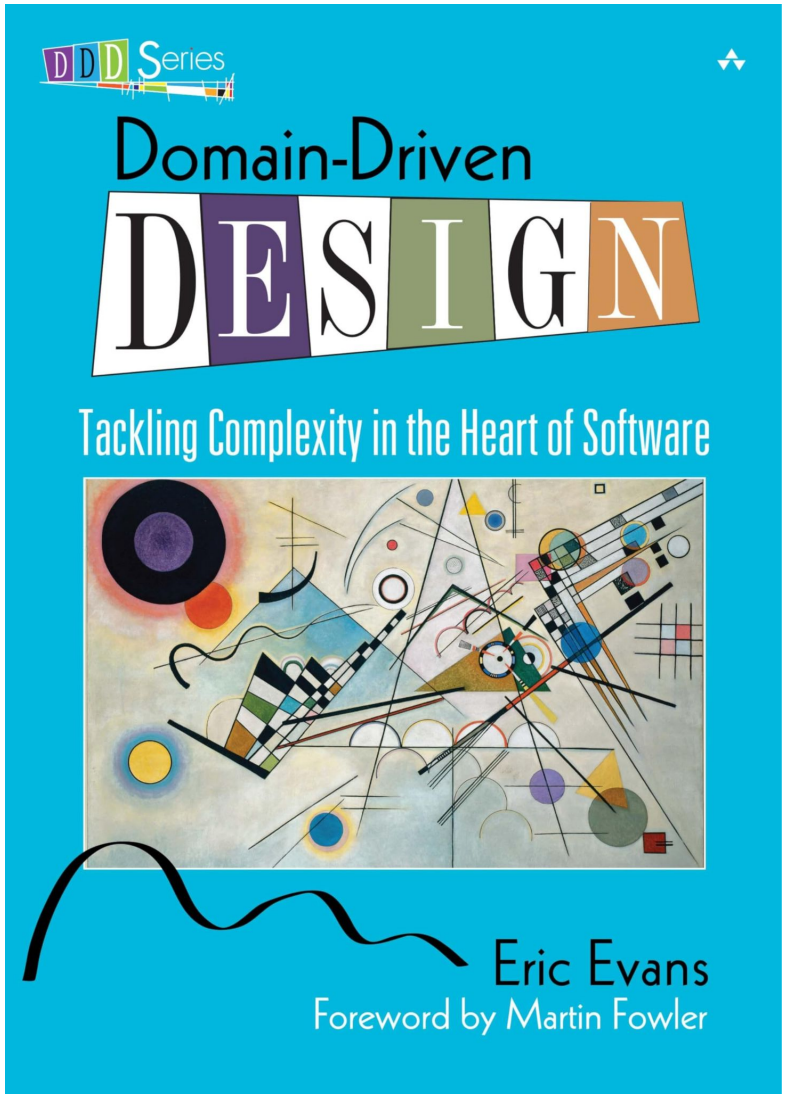
Separation of Concerns



Zum Beispiel:

- Fachliche Regeln gehören nicht in Controller
- Persistenz gehört nicht in die Domäne
- HTTP gehört nicht in Use Cases

Wie sich DDD und Clean Architecture ergänzen



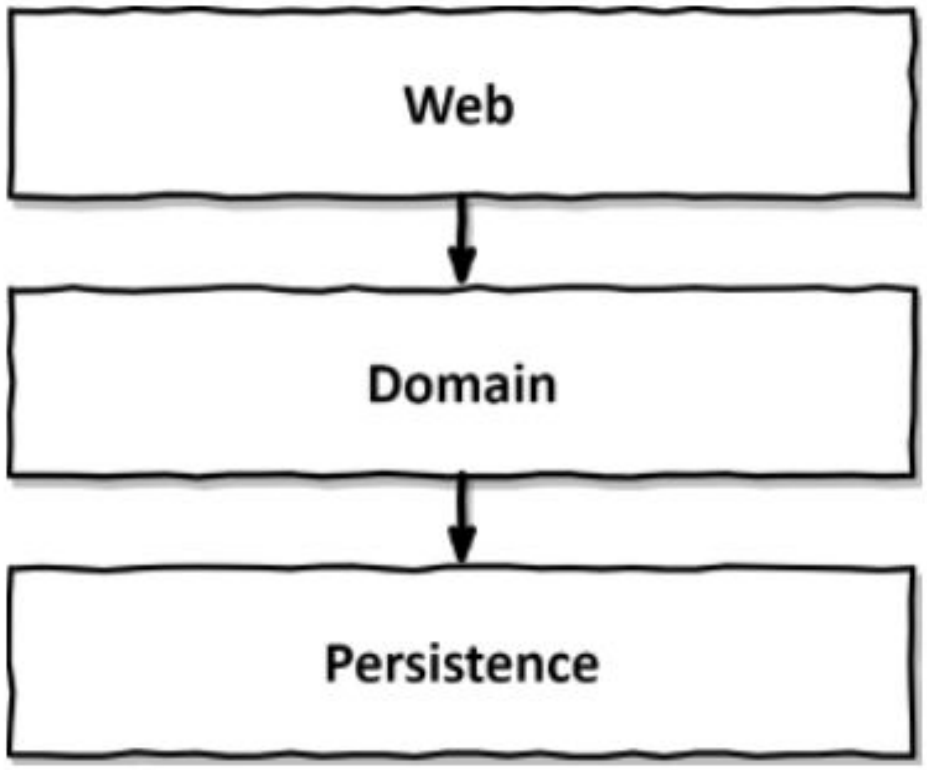
Clean Architecture

- Wie wir trennen
- Wie Abhängigkeiten verlaufen

Domain-Driven Design

- Was wir eigentlich trennen
- Wo fachliche Grenzen liegen

Wo Layer-Architekturen konzeptionell an Grenzen stoßen



Source: Tom Hombergs — *Get Your Hands Dirty on Clean Architecture*

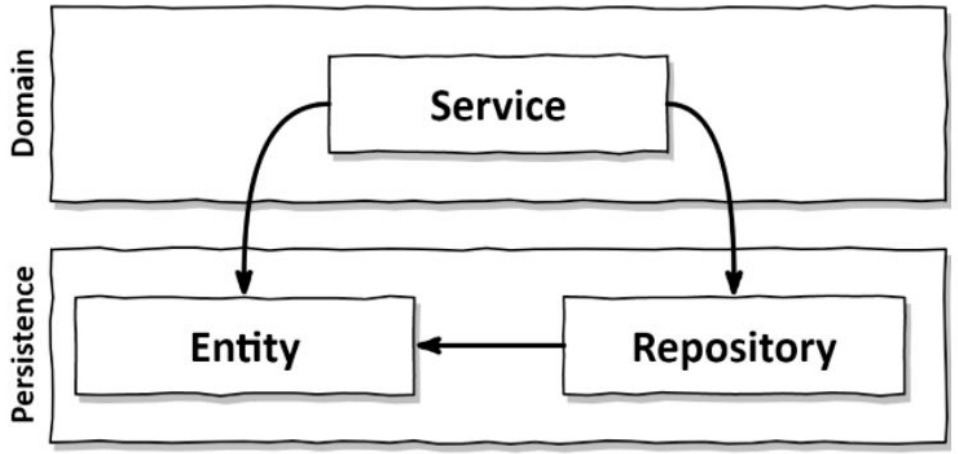
Zentrales konzeptionelles Problem

Layer beschreiben technische Rollen,
aber keine fachlichen Zusammenhänge

Wo Layer-Architekturen konzeptionell an Grenzen stoßen

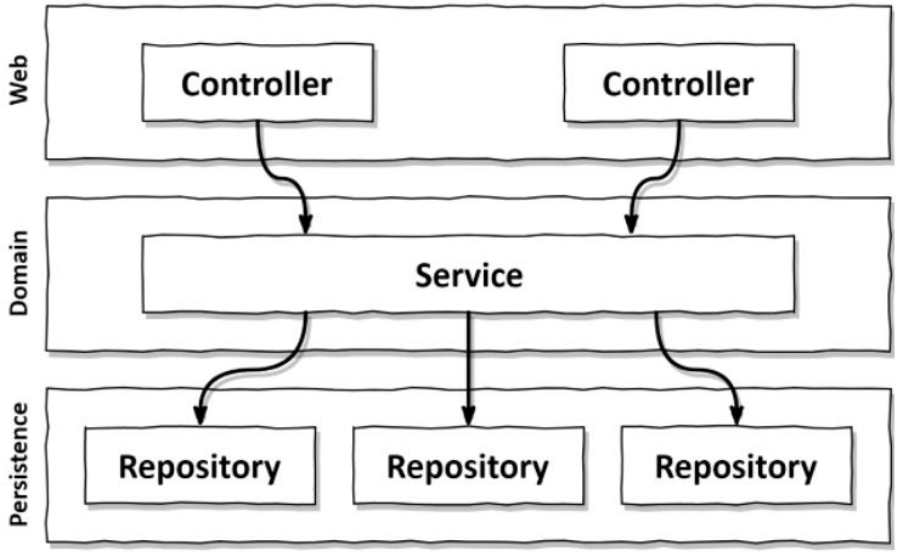
MHP
A PORSCHE COMPANY

Es fördert Database-Driven Design



Source: Tom Hombergs — *Get Your Hands Dirty on Clean Architecture*

Es versteckt die Use Cases



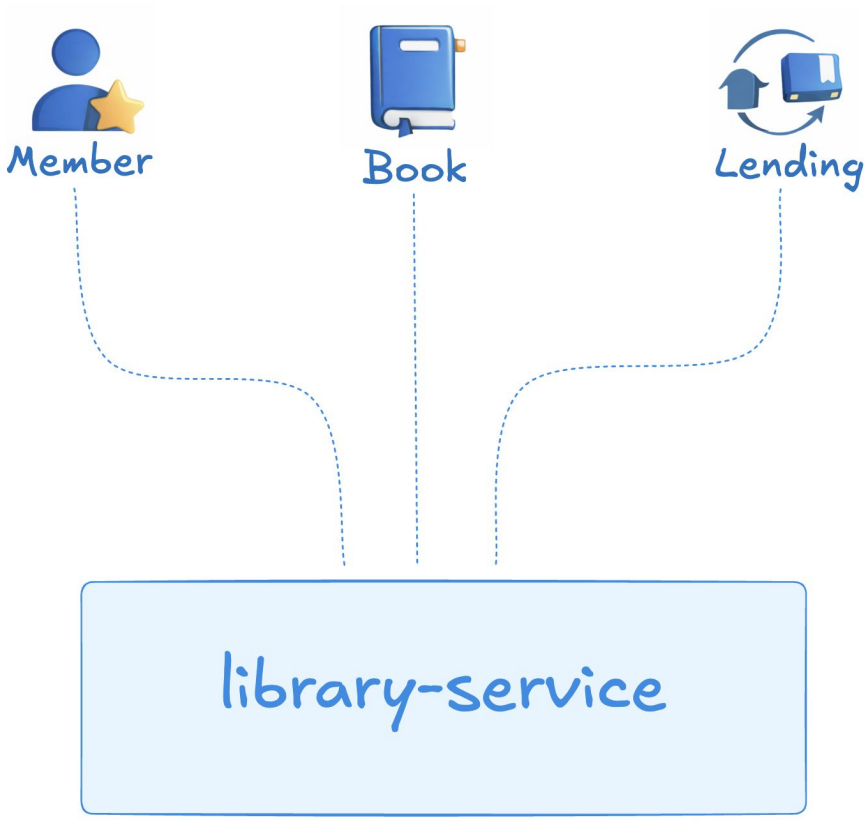
Source: Tom Hombergs — *Get Your Hands Dirty on Clean Architecture*

Schwer testbar

Erschwert paralleles Arbeiten im Team

Bibliothek Beispiel

Digital



Notification Adapter

```kotlin
@Component  fedorzholud
@Profile("!dev")
class EmailLendingNotificationChannel(
    private val mailSender: JavaMailSender,
) : LendingNotificationChannel {

    override fun sendConfirmation(notification: LendingConfirmationNotification) {  fedorzholu
        val text = emailText(
            bookTitle = notification.bookTitle,
            dueAt = notification.dueAt
        )

        val message = SimpleMailMessage().apply {
            setTo(notification.email)
            subject = "Library: Borrow confirmation"
            this.text = text
        }

        mailSender.send(message)
    }

    private fun emailText(bookTitle: String, dueAt: OffsetDateTime): String =  fedorzholud
        "You borrowed '$bookTitle'. Please return it until ${dueAt.format(DATE_FORMATTER)}."

    private companion object {  fedorzholud
        private val DATE_FORMATTER = DateTimeFormatter.ofPattern( pattern: "dd.MM.yyyy")
    }
}
```

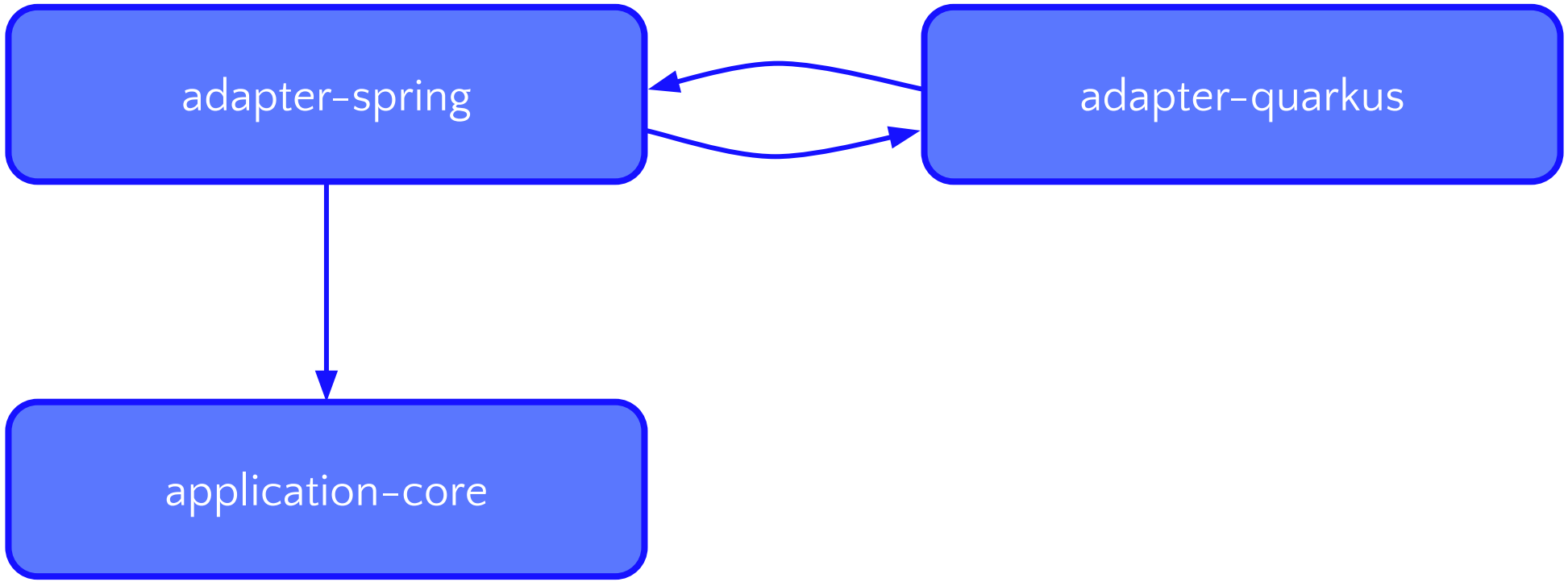
```kotlin
@Component  fedorzholud
@Profile("!dev")
class SmsLendingNotificationChannel(
    @Value("ACxxxxxxxxxxxxxxxxxxx") private val accountSid: String,
    @Value("your-token") private val authToken: String,
    @Value("+1234567890") private val fromNumber: String,
) : LendingNotificationChannel {

    @PostConstruct  fedorzholud
    fun init() {
        Twilio.init(accountSid, authToken)
    }

    override fun sendConfirmation(notification: LendingConfirmationNotification) {  f
        val phoneNumber = notification.phoneNumber ?: return

        val text = smsText(notification.bookTitle, notification.dueAt)

        Message.creator(
            PhoneNumber(phoneNumber),
            PhoneNumber(fromNumber),
            text
        ).create()
    }

    private fun smsText(bookTitle: String, dueAt: OffsetDateTime): String =  fedorzholu
        "Borrowed '$bookTitle'. Due: ${dueAt.format(DATE_FORMATTER)}."

    private companion object {  fedorzholud
        private val DATE_FORMATTER = DateTimeFormatter.ofPattern( pattern: "dd.MM.yyyy")
    }
}
```

Notification Adapter

```
@Component  fedorzholud
@Profile("!dev")
class NotificationWebClientAdapter(
    webClientBuilder: WebClient.Builder,
    @Value("http://localhost:9090") baseUrl: String,
) : LendingNotificationPort {

    private val client: WebClient = webClientBuilder.baseUrl(baseUrl).build()

    override fun sendConfirmation(notification: LendingConfirmationNotification) {  fedorzholud
        val dto = SendNotificationDto(
            email = notification.email,
            phoneNumber = notification.phoneNumber,
            message = buildMessage(notification.bookTitle, notification.dueAt)
        )

        client.post()
            .uri(NOTIFICATIONS_PATH)
            .contentType(MediaType.APPLICATION_JSON)
            .bodyValue(dto)
            .retrieve()
            .onStatus({ it.isError }) { resp ->
                resp.bodyToMono(String::class.java)
                    .defaultIfEmpty( defaultV: "")
                    .map { body ->
                        RuntimeException("Notification failed. status=${resp.statusCode().value()} body=$body")
                    }
            }
            .toBodilessEntity()
            .doOnSuccess { resp ->
                println("Notification sent. status=${resp.statusCode.value()}")
            }
            .doOnError { ex ->
                println("Notification failed: ${ex.message}")
            }
            .subscribe()
    }

    private fun buildMessage(bookTitle: String, dueAt: OffsetDateTime): String =  fedorzholud
        "You borrowed '$bookTitle'. Please return it until ${dueAt.format(DATE_FORMATTER)}."

    private companion object {  fedorzholud
        private val DATE_FORMATTER = DateTimeFormatter.ofPattern( pattern: "dd.MM.yyyy")
        private const val NOTIFICATIONS_PATH = "/api/notifications"
    }
}
```

```
@Component  fedorzholud
@Profile("!dev")
internal class NotificationKafkaProducer(
    private val kafkaTemplate: KafkaTemplate<String, SendNotificationMessage>
) : LendingNotificationPort {

    override fun sendConfirmation(notification: LendingConfirmationNotification) {  fedorzholu
        val message = SendNotificationMessage(
            email = notification.email,
            phoneNumber = notification.phoneNumber,
            message = buildMessage(notification.bookTitle, notification.dueAt)
        )

        kafkaTemplate.send(NOTIFICATION_TOPIC, message)
            .whenComplete { _, ex ->
                if (ex != null) {
                    println("Kafka notification failed: ${ex.message}")
                } else {
                    println("Kafka notification sent")
                }
            }
    }

    private fun buildMessage(bookTitle: String, dueAt: OffsetDateTime): String =  fedorzholud
        "You borrowed '$bookTitle'. Please return it until ${dueAt.format(DATE_FORMATTER)}."

    private companion object {  fedorzholud
        private const val NOTIFICATION_TOPIC = "lending.notifications.send"
        private val DATE_FORMATTER = DateTimeFormatter.ofPattern( pattern: "dd.MM.yyyy")
    }
}
```

Multi-Module-Ansatz

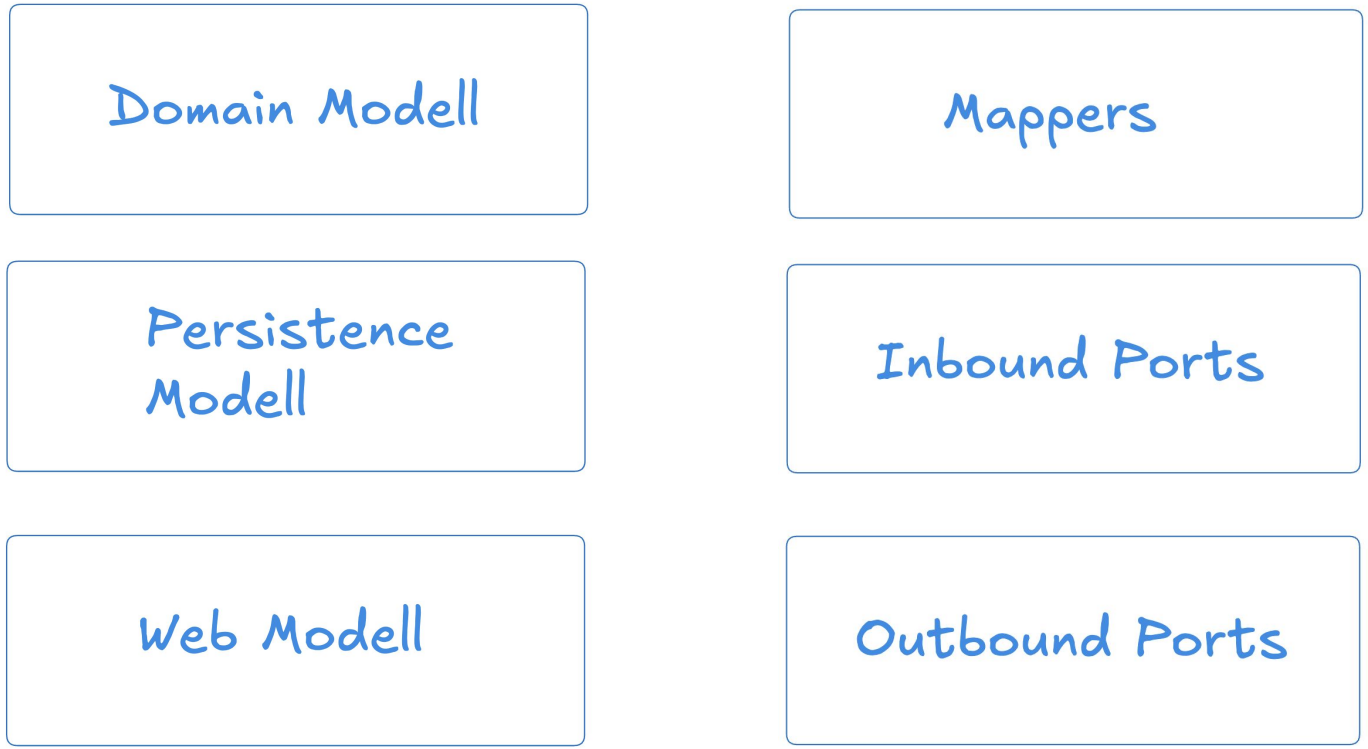


Abschlussgedanke

- Infrastruktur wird austauschbar.
- Business-Logik bleibt stabil.
- Der wichtigste Teil eines Systems ist die Fachlogik.

Steigende architektonische Komplexität

MHP
A PORSCHE COMPANY



Clean Architecture ist kein Selbstzweck. Sie ist ein Werkzeug.

Wann man besser auf Clean Architecture verzichtet

- Einfache CRUD-Anwendungen ohne Fachlogik
- Proof of Concepts oder Prototypen
- Architektonische Komplexität größer als fachliche Komplexität

Wenn die architektonische Komplexität höher ist als die fachliche Komplexität, dann ist Clean Architecture wahrscheinlich nicht die richtige Wahl.

Fazit

Clean Architecture lohnt sich vor allem dann, wenn man mit wachsender oder komplexer Fachlogik, langer Lebensdauer und vielen Änderungen rechnet.

Clean Architecture ist eher Hype oder Overengineering, wenn die Fachlogik simpel ist, das Projekt klein bleibt und schnelle Umsetzung wichtiger ist als langfristige Wartbarkeit.

Clean Architecture: fundament wartbarer Software oder nur ein Hype?

MHP
A PORSCHE COMPANY

Meine Antwort: Jein (oder it’s depends)

Was glauben Sie?

MHP
A PORSCHE COMPANY

Vielen Dank für Ihre Aufmerksamkeit

GitHub: https://github.com/FedorZholud/clean-architecture-library-service

LinkedIn: https://www.linkedin.com/in/fedor-zholud/

Fedor Zholud