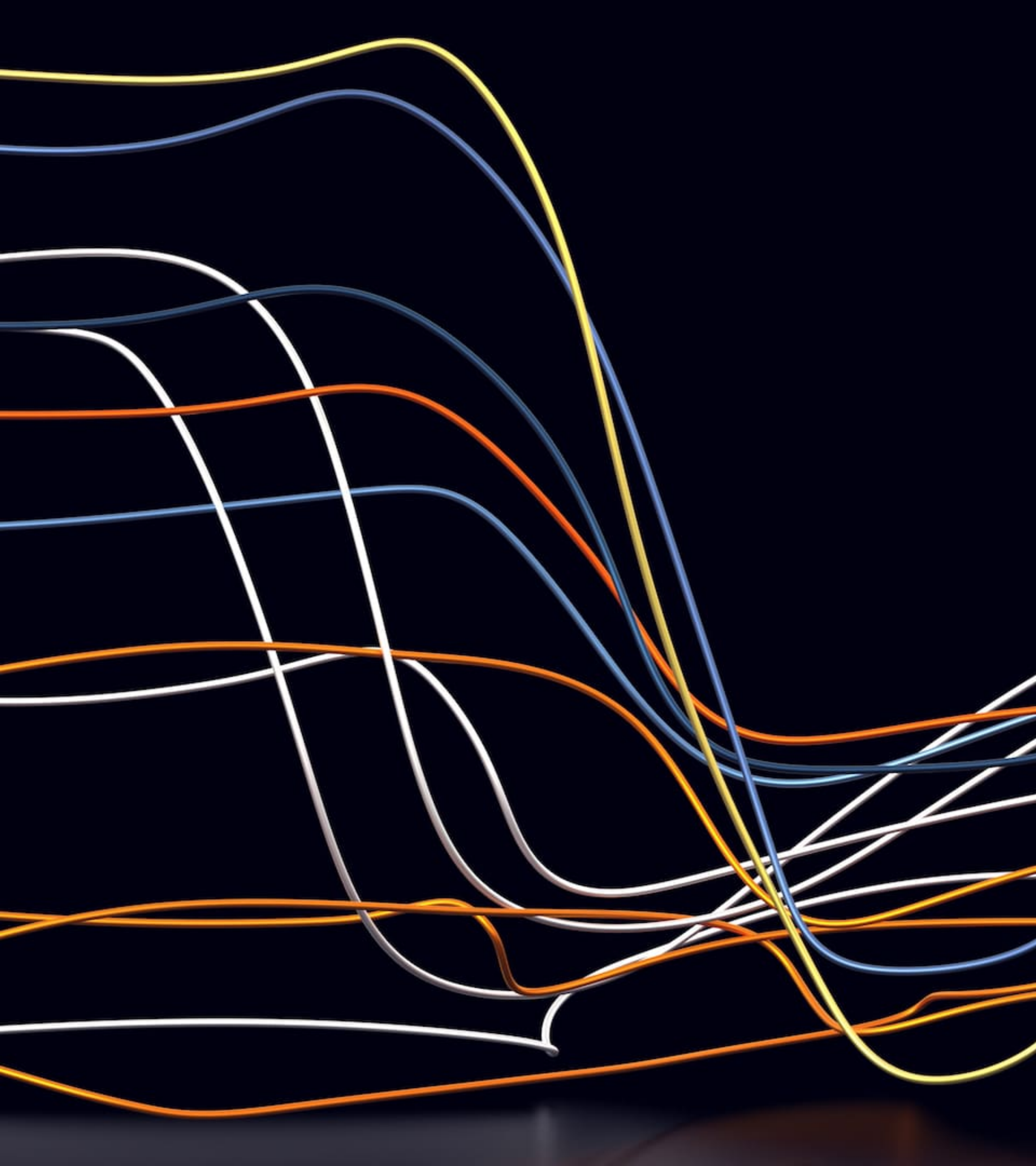




Technology Day 2025

Von fachlich sauberen Zwiebeln und hexagonalen Adaptern

INNOQ



FABIAN WALTHER
INNOQ.SOCIAL/@FABIAN

Disclaimer

Kontext

- Fokus auf Geschäftsprozessanwendungen in Java
- Keine Gameengines, Kryptographie-Bibliotheken, Embedded-Systeme
- Unterschiedliche Settings von 1 bis N Entwicklungsteam(s)
- Java / JVM -> Objektorientierung, Architektur Clean-, Onion- und Konsorten, DDD taktisch und strategisch
- Von der Arbeit mit Code zur Arbeit mit Menschen

Die Auswahl ist willkürlich. Was gab und gibt es noch?

Motivation

„Wozu zum *** sollte man eine hexagonale
Architektur brauchen?“**

„Architektur ist kein Problem was gelöst werden muss!“

„Architektur soll Probleme lösen!“

Was ist Software-Architektur (für mich)?

ISO 42010 (frei übersetzt):

„Die grundsätzliche Organisation eines Systems,
wie sie sich in dessen Komponenten,
deren Beziehung zueinander und
zur Umgebung widerspiegelt,
sowie die Prinzipien, die
für dessen Design und
Evolution gelten.“

Quelle: INNOQ / ISAQB Foundation Training

„<...> Prinzipien, die
für dessen Design und
Evolution gelten.“

Quelle: INNOQ / ISAQB Foundation Training

**Warum will ich diese Prinzipien ausarbeiten und ein
Design festhalten?**

Qualitäten!

Domänenzentrische Architekturstile

Die Geschichte

- 1. Hexagonal (Ports & Adapters) -
Alistair Cockburn ~2005**
- 2. Onion - Jeffrey Palermo ~2008**
- 3. Clean - Robert C. Martin ~2012**

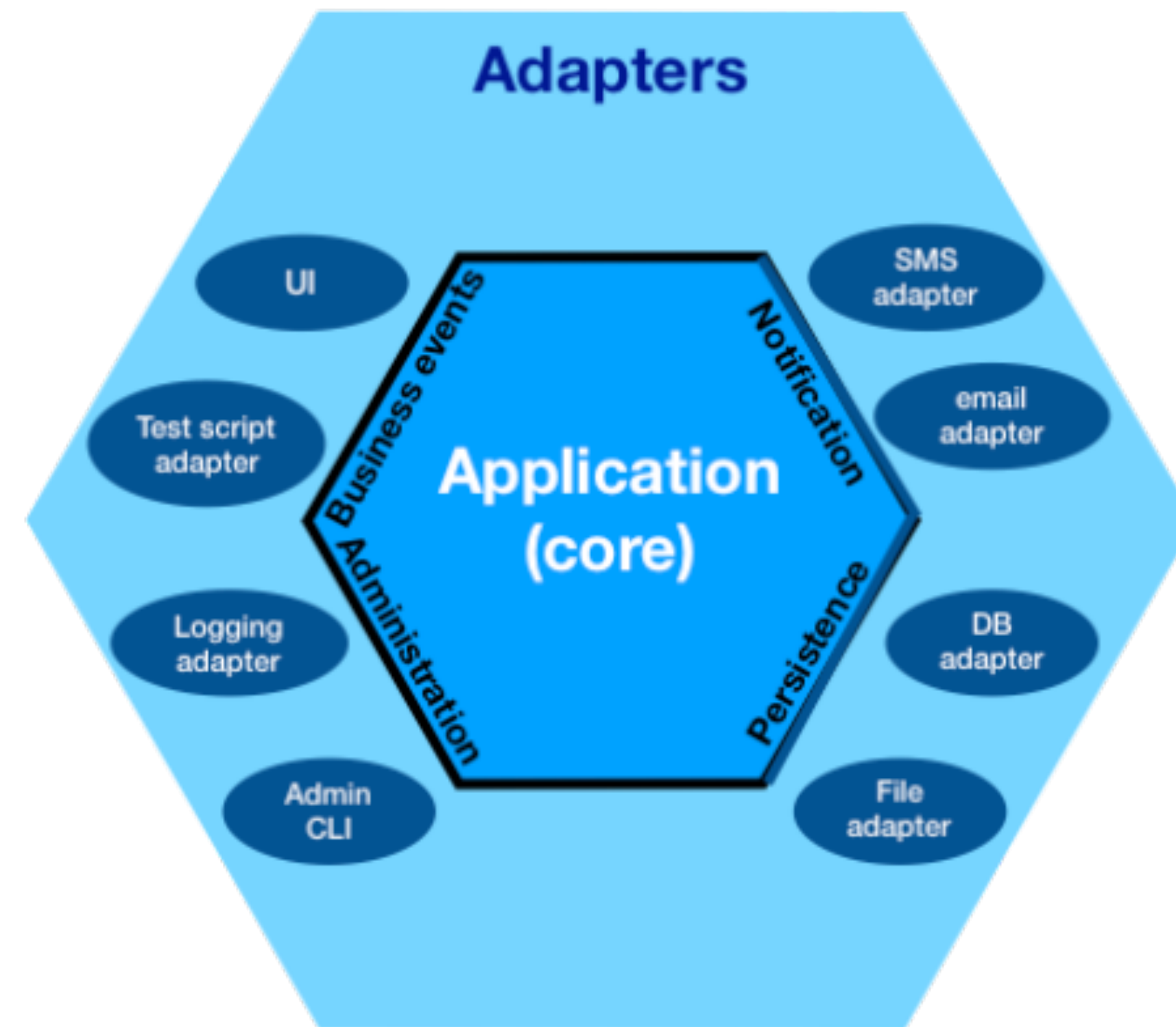
Hexagonal

Alistair

Cockburn

2005

Ports & Adapters



Hexagonal

Alistair Cockburn

2005

Ports & Adapters

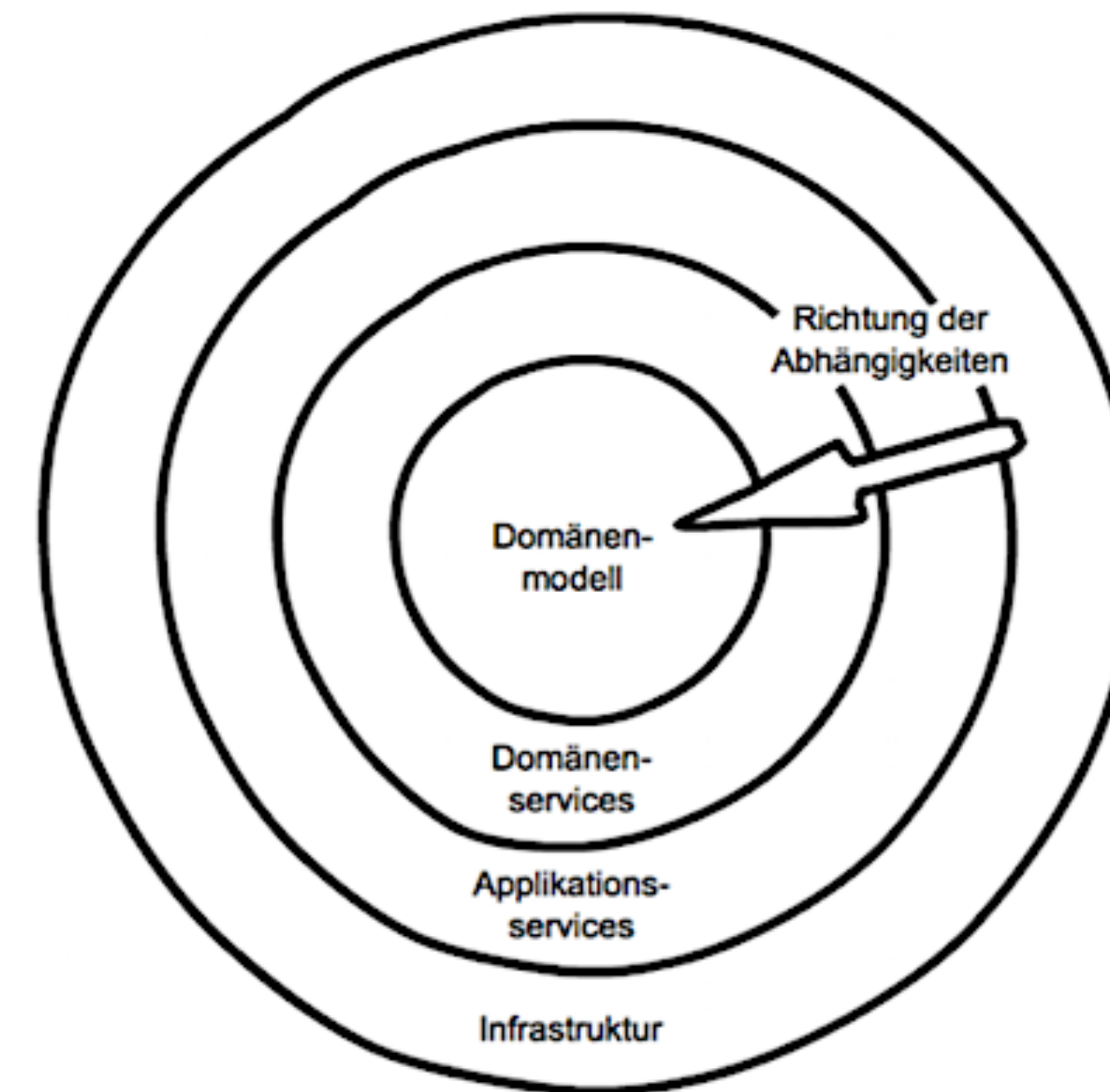
- Innenwelt (Domäne) bietet Ports
- Außenwelt (Infrastruktur) implementiert Adapter
- UI, Persistence, Kommunikation sind extern
- Besonders: Die Daten(Banken) stehen nicht im Zentrum, sondern Fachlogik
- Testbarkeit ohne Infrastruktur

Onion

Jeffrey Palermo

2008

Mehr Domäne



Onion

Jeffrey Palermo

2008

Mehr Domäne

- The application is built around an independent object model
- Inner layers define interfaces. Outer layers implement interfaces
- Direction of coupling is toward the center
- All application core code can be compiled and run separate from infrastructure

Clean

Robert C. Martin

2012

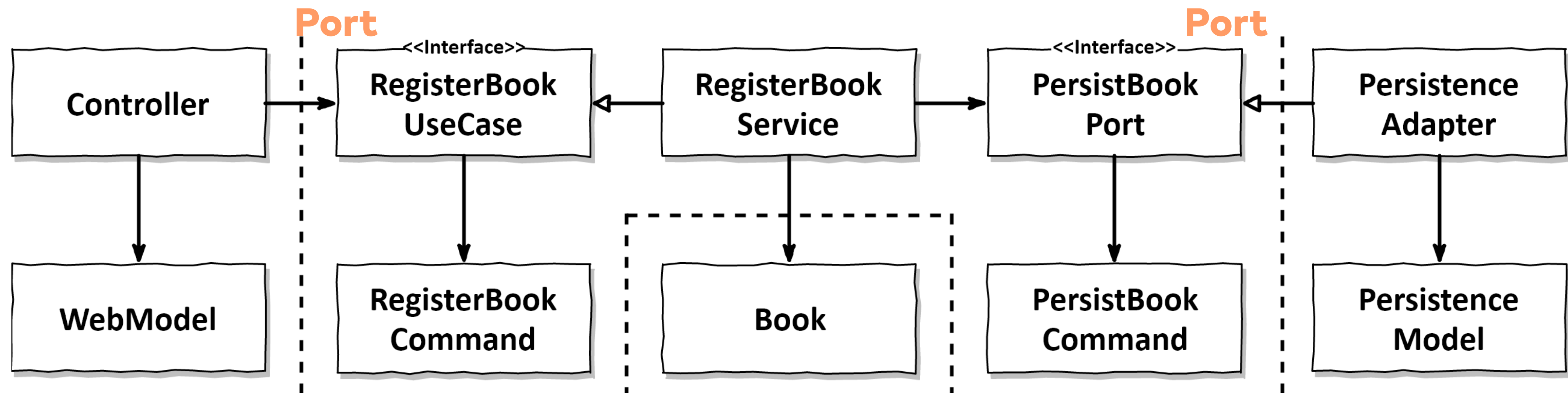
(Noch) Mehr Domäne

- Baut bewusst auf den vorherigen auf
- Formuliert klare Regeln
- Formalisiert die eher loseren Konzepte
- Starker Fokus auf eigene Use-Case-Schicht
- Lustigerweise hat Cockburn Use-Cases eigentlich geprägt, aber

Alles irgendwie gleich?

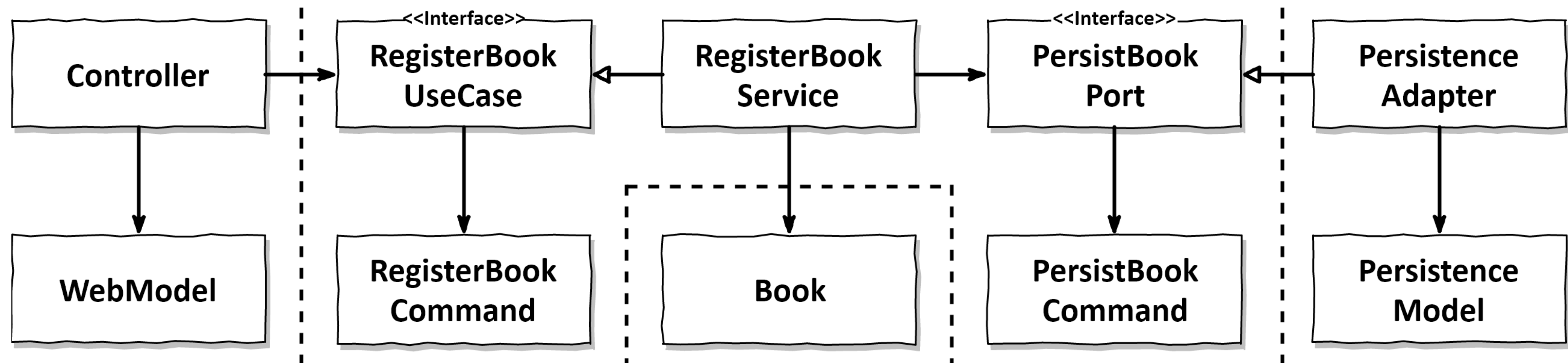
Infrastructure

Application Core



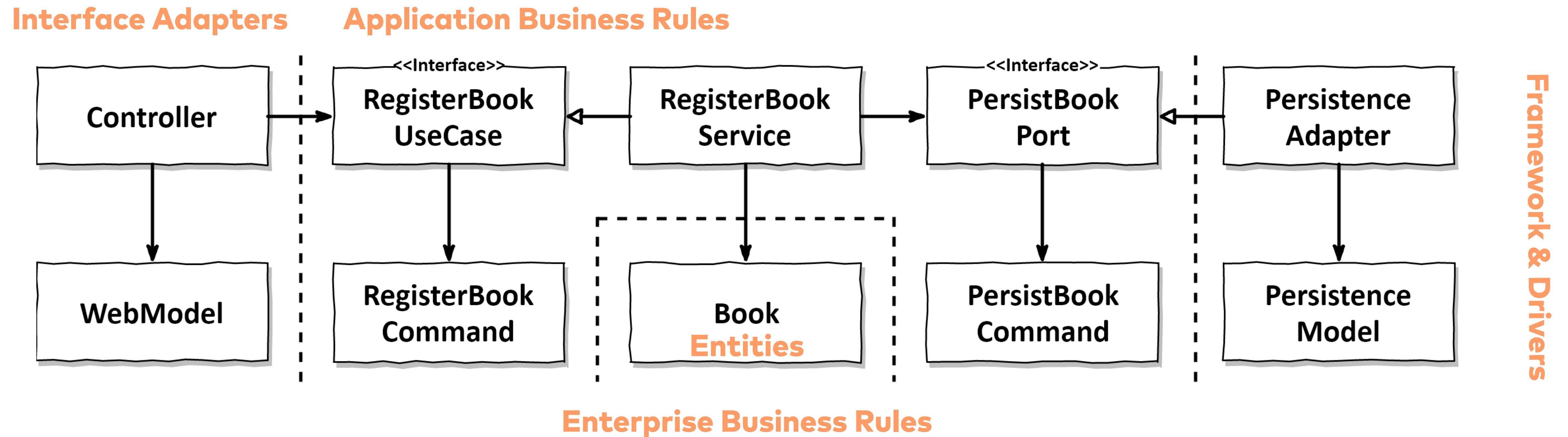
Alles irgendwie gleich?

Infrastructure Ring Application Services Domain Services



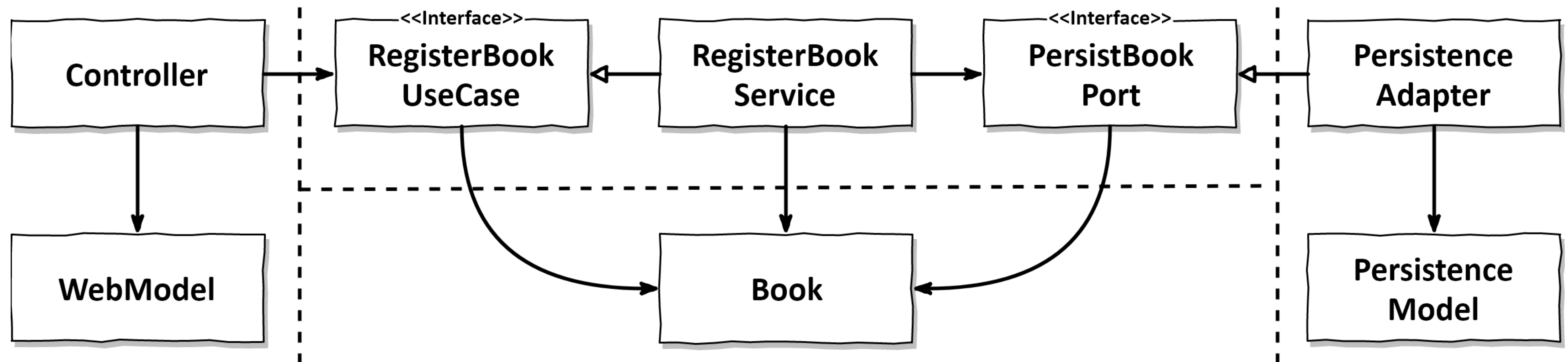
Domain Model

Alles irgendwie gleich?



Richtung zählt. Überspringen erlaubt.

Kein „Cargo Culting“!



Fokus und Schwerpunkte

**Hexagonal: Use-Case
zentriert -> Adapter
nutzen Application mit
Ports und werden genutzt**

**Onion: Domäne bekommt
mehr Raum. Trennung in
Fachmodel und
Orchestrierung (Domain
und Application Layer)**

**Clean vereint und formalisiert beides, aber setzt
explizit auf Use-Cases als leitendes Konzept**

Und wenn ich das nun verwenden will?

Wenn der Sinn hinter Entscheidungen nicht verstanden wird, helfen auch keine erzwungenen ArchUnit-Tests

ArchUnit-Fehler sollten nicht nur erklären, welche Regel verletzt wurde...

**...sondern warum man es mal für notwendig gehalten
hat diese Regel einzuführen.**

Was ist ihr Ziel?

Ist das Ziel evtl. nicht mehr gültig?

Gut

```
onionArchitecture()  
    .domainModels("com.myapp.domain.model..")  
    .domainServices("com.myapp.domain.service..")  
    .applicationServices("com.myapp.application..")  
    .adapter("cli", "com.myapp.adapter.cli..")  
    .adapter("persistence", "com.myapp.adapter.persistence..")  
    .adapter("rest", "com.myapp.adapter.rest..");
```

Viel besser!

```
@ArchTest
public static final ArchRule ONION_ARCHITECTURE_IS_RESPECTED =
    onionArchitecture()
        .domainModels( ...packagelntifiers: "..domain.model..", "..domain.shared..")
        .domainServices( ...packagelntifiers: "..domain.service..").withOptionalLayers(true)
        .applicationServices( ...packagelntifiers: "..application..")
        .adapter( name: "adapters", ...packagelntifiers: "..adapter..")
        .because( reason: "Onion Architecture: Abhängigkeiten zeigen nur nach innen. " +
            "Motivation: Domain bleibt unabhängig von äußeren Schichten (Adaptern), " +
            "dadurch wird die Geschäftslogik testbar und austauschbar.");
```



Lesen!

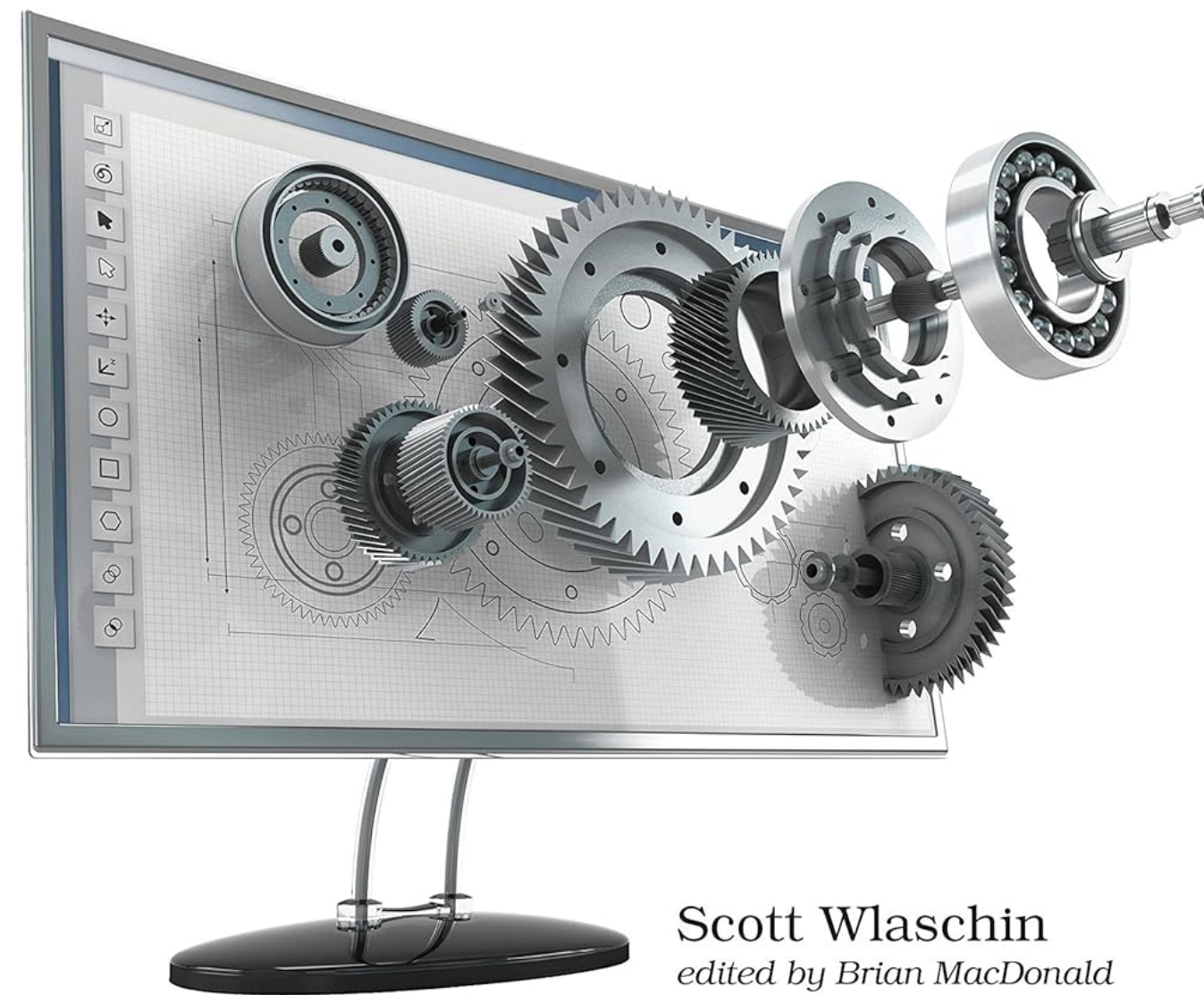


Lesen!

The
Pragmatic
Programmers

Domain Modeling Made Functional

Tackle Software Complexity with
Domain-Driven Design and F#



Scott Wlaschin
edited by Brian MacDonald

Fazit

„Architektur ist kein Problem was gelöst werden muss!“

„Architektur soll Probleme lösen!“

„Wozu zum *** sollte man eine hexagonale
Architektur brauchen?“**

**Habe ich „Probleme“, die zu den gezeigten Ansätzen
passen?**

Go!

Es gibt aber auch 1000 andere Lösungen. Wichtig ist:

**Bewusst machen, welchen Punkt man adressieren
möchte und ...**

... bewusst kommunizieren wie man ihn angeht.

Was bringen die Ansätze mir denn nun?

**Gemeinsamer Fokus auf Trennung der
Technik von Fachlichkeit**

Fachlichkeit bleibt, Technik geht?

Beides verändert sich, aber vermutlich nicht im gleichen Rhythmus.

**Alle arbeiten aufs gleiche Ziel hin. Mit
unterschiedlichem Wording und Methoden.**

Was zählt ist das Ziel.

Sidequest Dokumentation

**„Einfach nur“ Architekturentscheidungen treffen reicht
nicht.**

**Niemand erinnert sich übermorgen an bewusste
Entscheidungen.**

Bitte dokumentiert nicht was ihr geändert habt.

**Bitte dokumentiert nicht was ihr geändert habt.
Auch nicht warum.**

Bitte dokumentiert nicht was ihr geändert habt.

Auch nicht warum.

**Sondern: Was war der fachliche Treiber für die
Änderung?**

Zu häufig gesehen:

~~**int schwellwert = 7;**~~

**//Schwellwert nach Rückmeldung Fachbereit
angepasst**

int schwellwert = 12;

😊 Besser

**//Schwellwert angepasst, da Ansicht nach 7 Sekunden
meist noch nicht geladen.**

int schwellwert = 12;

 Viel besser

**//Schwellwert angepasst, da Ansicht nach 7 Sekunden
meist noch nicht geladen. (Es werden jetzt auch noch
die Widgets von Team XY eingebunden. Deshalb dauert
länger.)**

int schwellwert = 12;

Es reicht niemals zu sagen: „Wir machen hier Onion-Architektur. Hier ist dein Ticket, leg los.“

**Es ist wie bei: „Willkommen im Team. Wir machen hier
SRUM*. Los gehts.“**

- * „Achja, Retros lassen wir weg. Brauchen wir nicht.“
- * "Der Prozess steht. Wir ändern da nichts. Hat sich bewährt."
- * „Um das Daily kurz zu halten, keine technischen Diskussionen. Das bremst nur.“

Die Entscheidung für eine Architektur muss erklärt werden. Immer wieder. Und wieder.

**Dokumentation ist schön, aber lebendig werden
Architekturentscheidungen nur durch gutes Vorbild.
Und Begründung. Wirklich.**

Architektur von außen funktioniert nicht (langfristig).

Zum Schluss: Zurück zu den Qualitäten

„<...> Prinzipien, die
für dessen Design und
Evolution gelten.“

Quelle: INNOQ / ISAQB Foundation Training

ISO 25010:2011 Qualitätsmodell

Funktionale Eignung

X bietet Funktionalität, die den angegebenen und implizierten Bedürfnissen entspricht

Zuverlässigkeit

X führt Funktionen unter den festgelegten Umgebungen aus

Sicherheit

X schützt Informationen und Daten

Wartbarkeit

X kann modifiziert werden, um es zu verbessern, zu korrigieren oder an Änderungen anzupassen

Übertragbarkeit

X kann auf verschiedenen Umgebungen betrieben werden

Benutzbarkeit

X kann von festgelegten Benutzern verwendet werden, um vorgegebene Ziele zu erreichen

Kompatibilität

X kann Informationen mit anderen Systemen austauschen

Leistungseffizienz

X liefert angemessene Geschwindigkeit mit den bereitgestellten Ressourcen

Eigene Übersetzung aus dem Englischen
X = Produkt, System, Baustein, Artefakt etc.

Weitere Links und Mentions

- <https://www.innoq.com/de/articles/2012/04/quality-driven-software-architecture/>
- <https://www.amazon.de/Clean-Architecture-Praxisbuch-Software-Architektur-Codebeispielen/dp/3747508146>
- <https://www.amazon.de/Your-Hands-Dirty-Clean-Architecture/dp/180512837X>
- <https://www.amazon.de/Domain-Modeling-Made-Functional-Domain-Driven-ebook/dp/BOCY2L7Y1K>

Feedback? Contact!



Fabian Walther

fabian.walther@innoq.com

innoq.social/@fabian

LinkedIn



innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin

Ludwigstr. 180E
63067 Offenbach

Kreuzstr. 16
80331 München

Hermannstrasse 13
20095 Hamburg

Erftstr. 15-17
50672 Köln

Königstorgraben 11
90402 Nürnberg