

Clean Architecture unter der Lupe Ein Realitätscheck

Werner Eberling (@wer_eb)
werner.eberling@mathema.de

Hristiyan Pehlivanov
hristiyan.pehlivanov@mathema.de

www.mathema.de

Die Sprecher



Werner Eberling

Principal Consultant / Autor

Email: werner.eberling@mathema.de

Twitter: [@Wer_Eb](https://twitter.com/Wer_Eb)



Die Sprecher

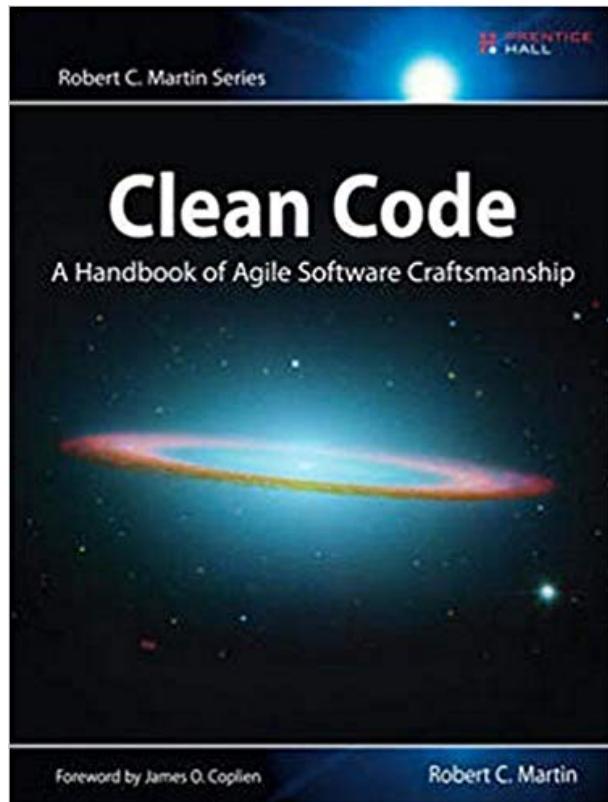


Hristiyan Pehlivanov

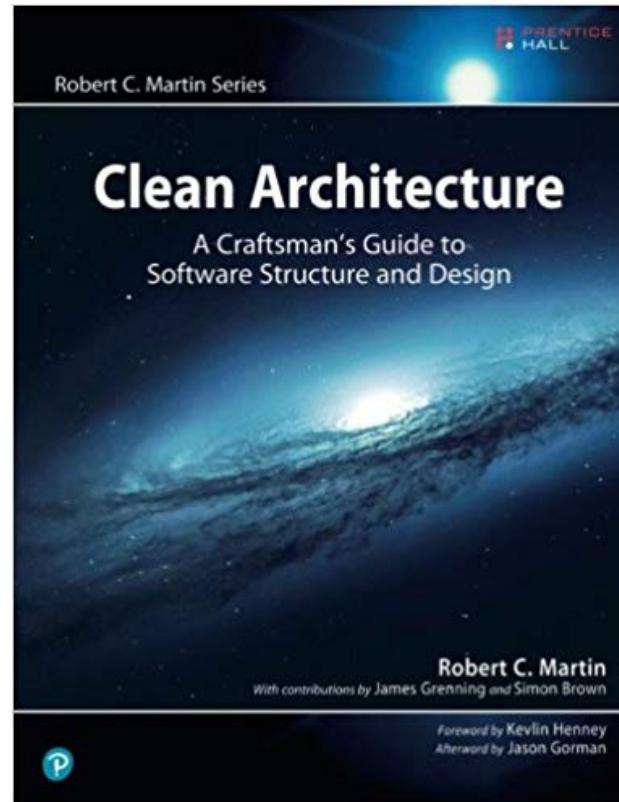
Consultant

Email: hristiyan.pehlivanov@mathema.de

Von Clean Code zu Clean Architecture?



= ??? =>



Recap: Wann ist Code „clean“ ?

- Lesbar
- Leicht Verständlich
- Gut weiterentwickelbar
- Keine Überraschungen

Recap: Wann ist Code „clean“ ?

- Gute Namensgebung
- Klarer Aufbau, Verwendung von Design Patterns
- Gut testbar
- Klare Zuständigkeiten („Do exactly one thing“)

Ergo: Wann ist Architektur „clean“ ?

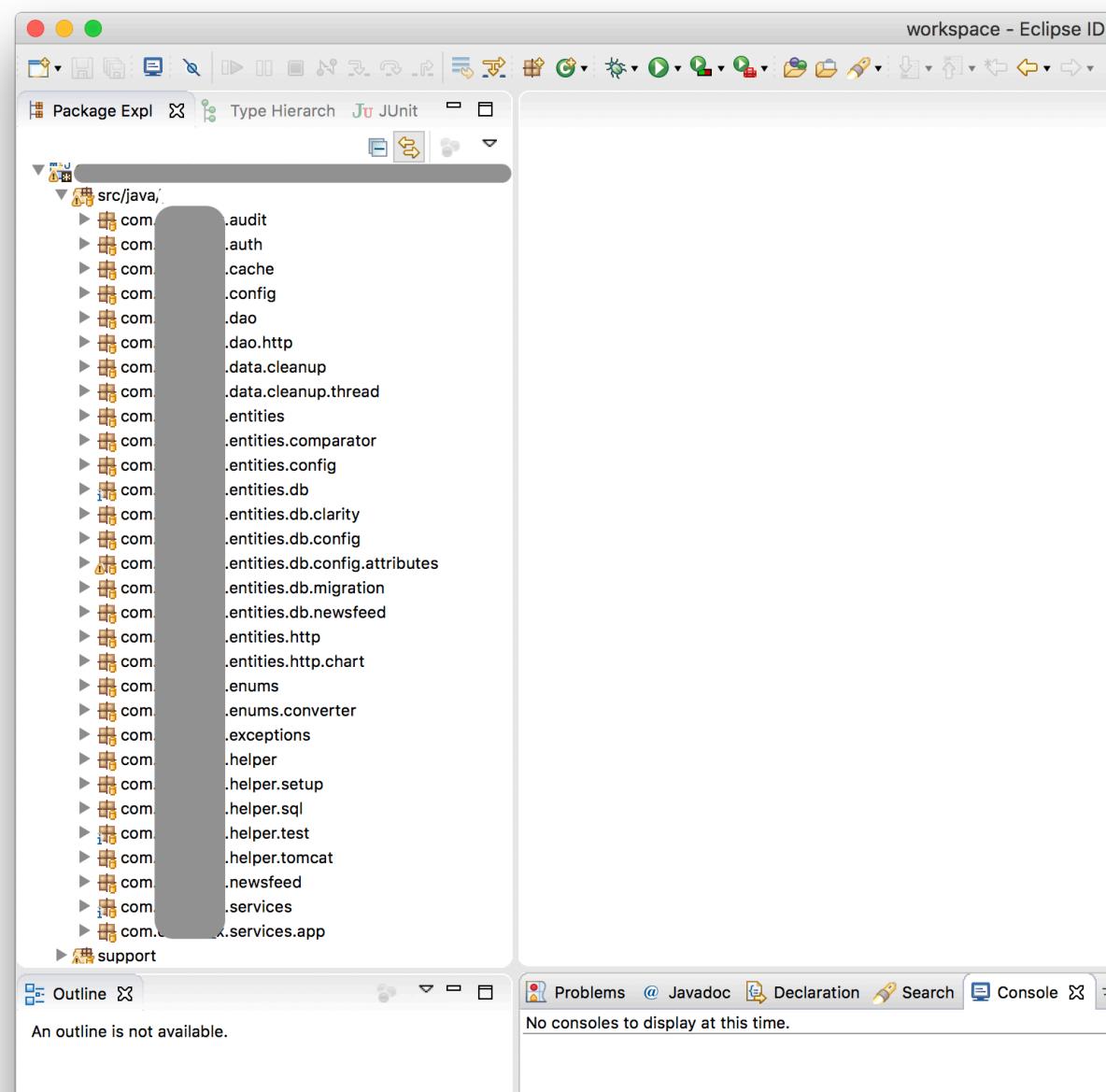
- Gute Namensgebung
- Klarer Aufbau, Verwendung von Architektur Patterns
- Gut testbar
- Keine Überraschungen

Ergo: Wann ist Architektur „clean“ ?

- Gute Namensgebung
- Klarer Aufbau, Verwendung von Architektur Patterns
- Gut testbar
- Keine Überraschungen

Klingt gut, aber was
bedeutet das im Detail?

Aus dem Projekt... (Zooming)

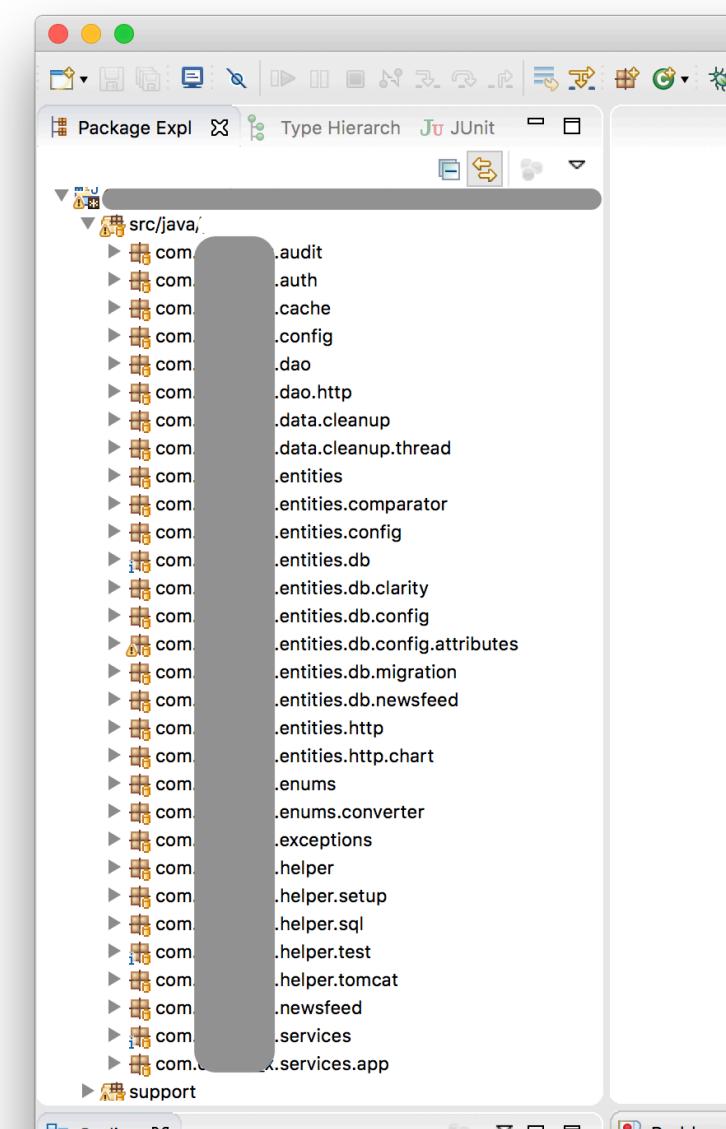


Exkurs: Architektur im wahren Leben

Kirchengrundriss

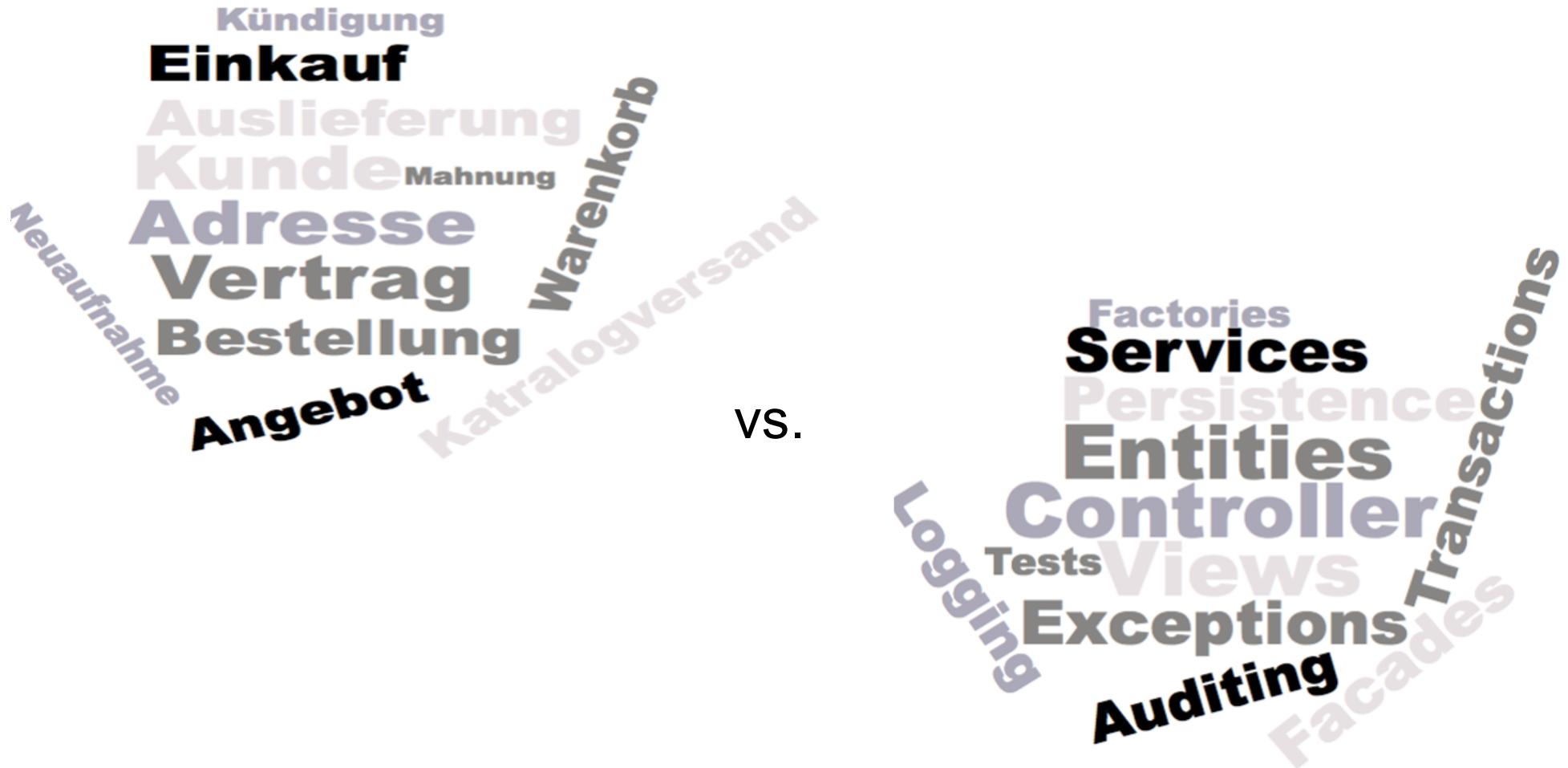
Wohnungsgrundriss

So würde es wohl aussehen...

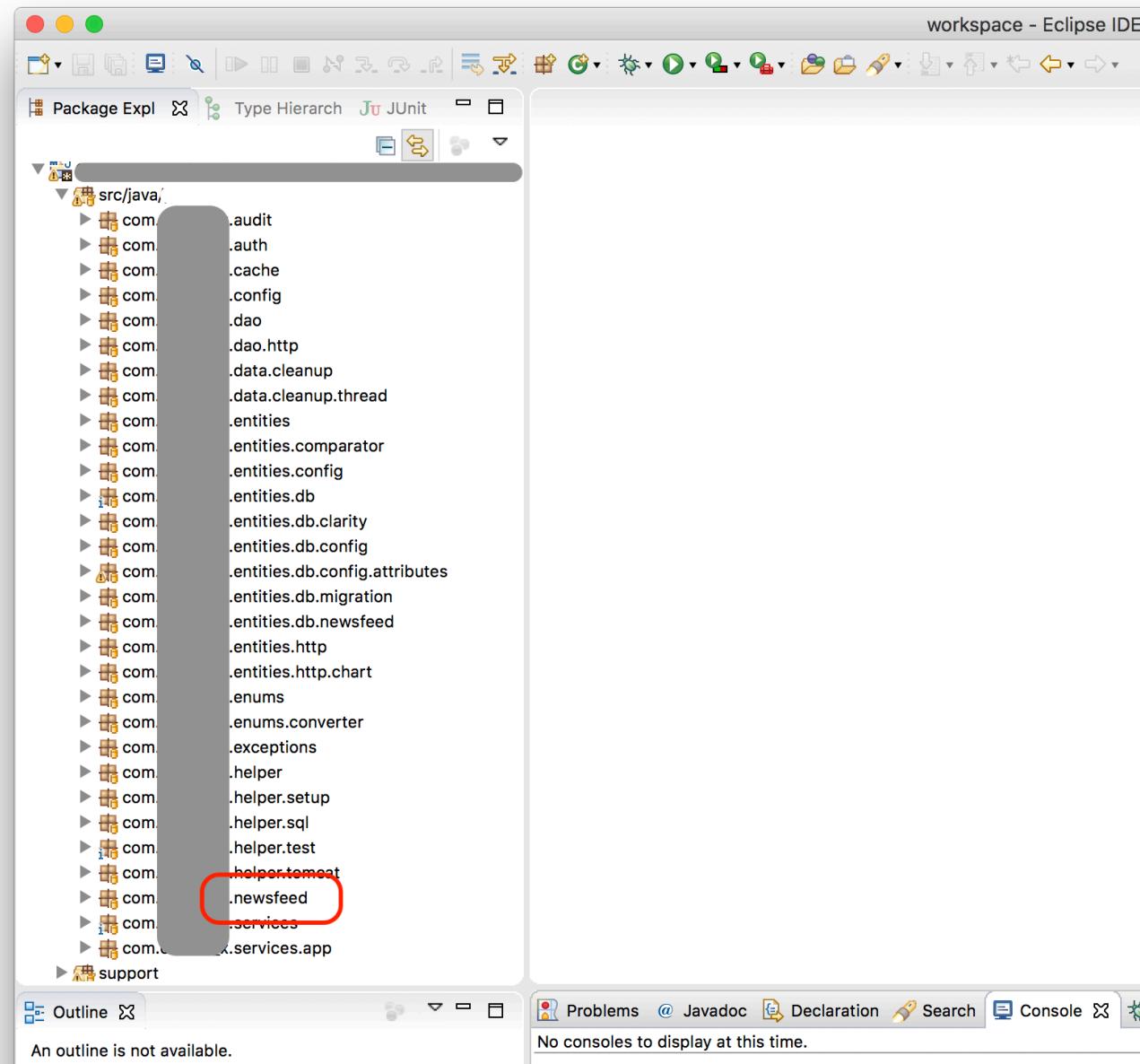


Schmuddel Buddels Haus

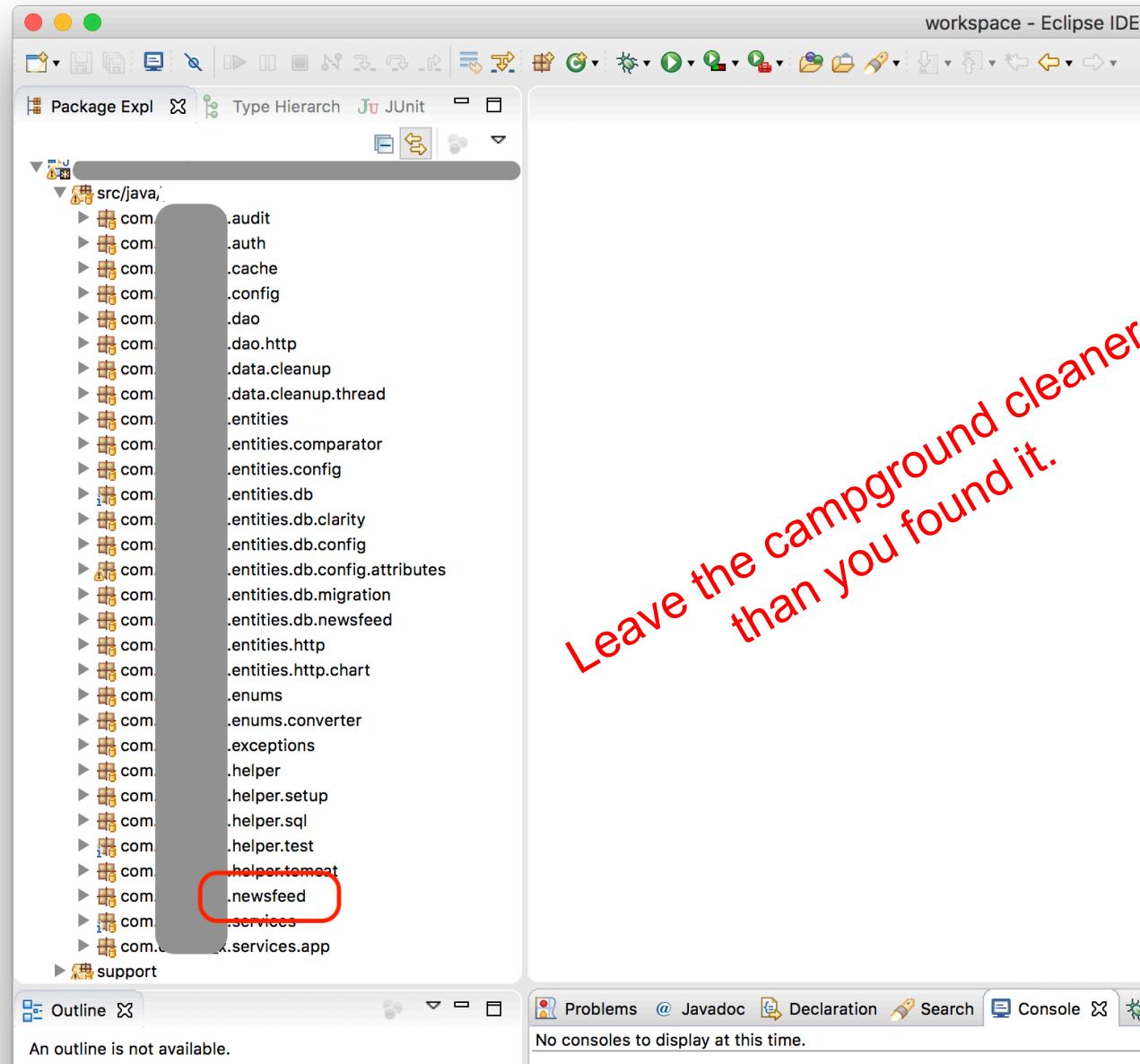
Screaming Architecture



Die Rebellion beginnt mit einem Funken...



Die Rebellion beginnt mit einem Funken...



Bestandteile identifizieren und auseinanderhalten

Wichtiges vs. Details

- **Wichtiges**
 - Fachliche Teilaspekte
 - Fachliche Prozesse
 - Fachliche Abhängigkeiten
- **Details**
 - Patterns
 - Programmiersprachen
 - Frameworks
 - Laufzeitumgebungen

Patterns ändern sich auch im wahren Leben



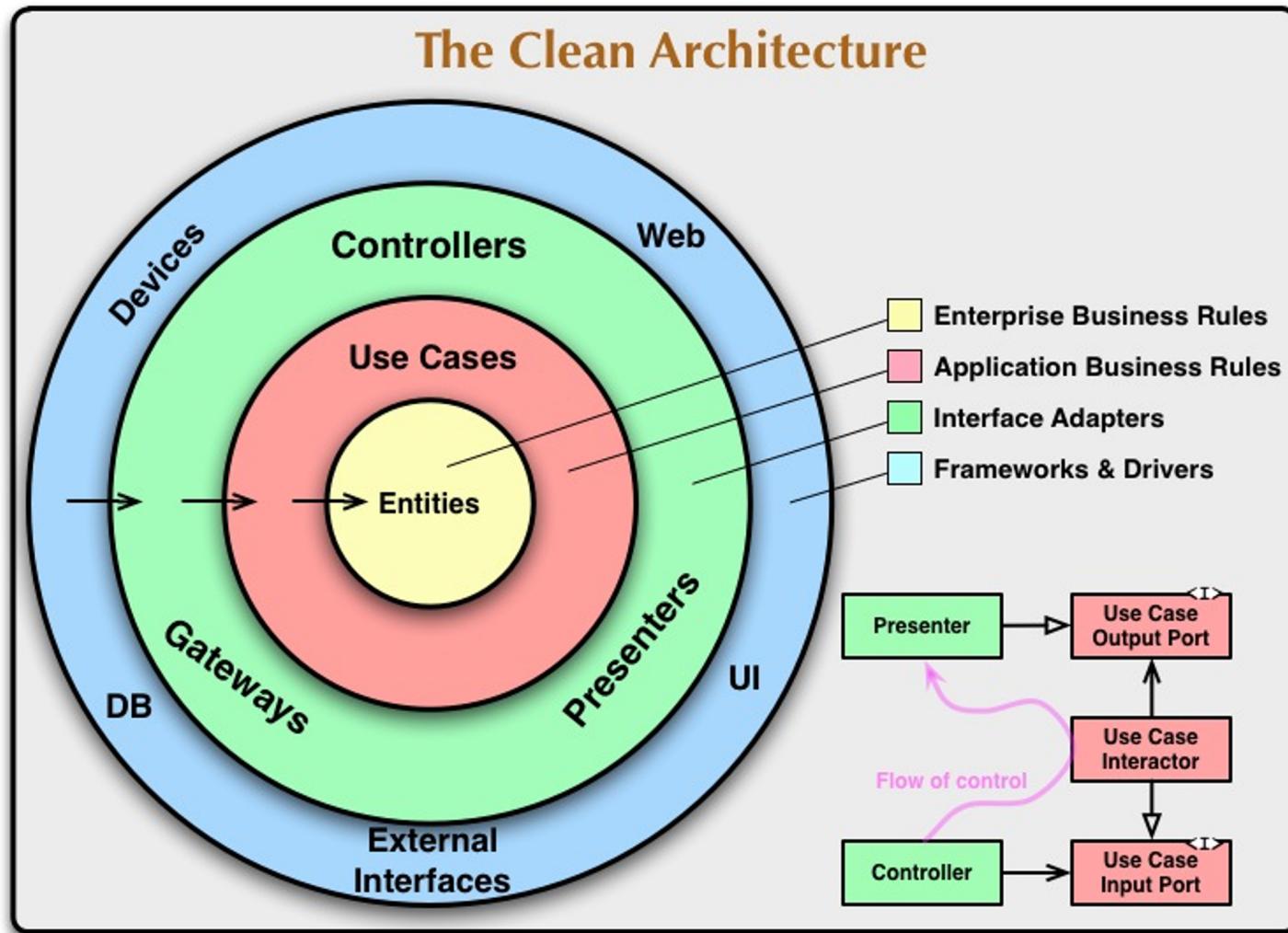
© Thomas Wolf, www.foto-tw.de (CC BY-SA 3.0 DE)

Eine Kafkaeske Geschichte



https://www.fischerverlage.de/buch/franz_kafka_der_process/9783596903566

Von innen nach aussen



<http://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>

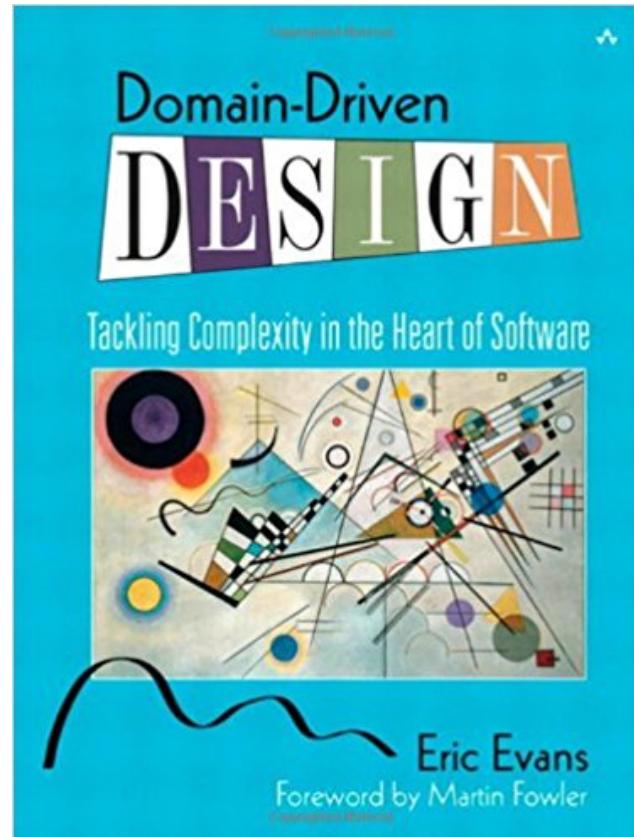
Nichts wirklich neues

- Hexagonal Architektur (Alistair Cockburn Mid'90s)
- Ports and Adapter (Alistair Cockburn 2005)
- Onion Architecture (Jeffrey Palermo 2008)
- ...
- Aber deswegen schlecht?

Zitate - Entities

- "Entities are generalizations that can be **used in many different applications**, ..." – Clean Architecture, S. 193
- "Entities encapsulate **enterprise-wide** Critical Business Rules. ... If you don't have an enterprise and are writing just a single application, then these entities are the business object of the application." – Clean Architecture, S. 204

Die Idee kommt doch bekannt vor



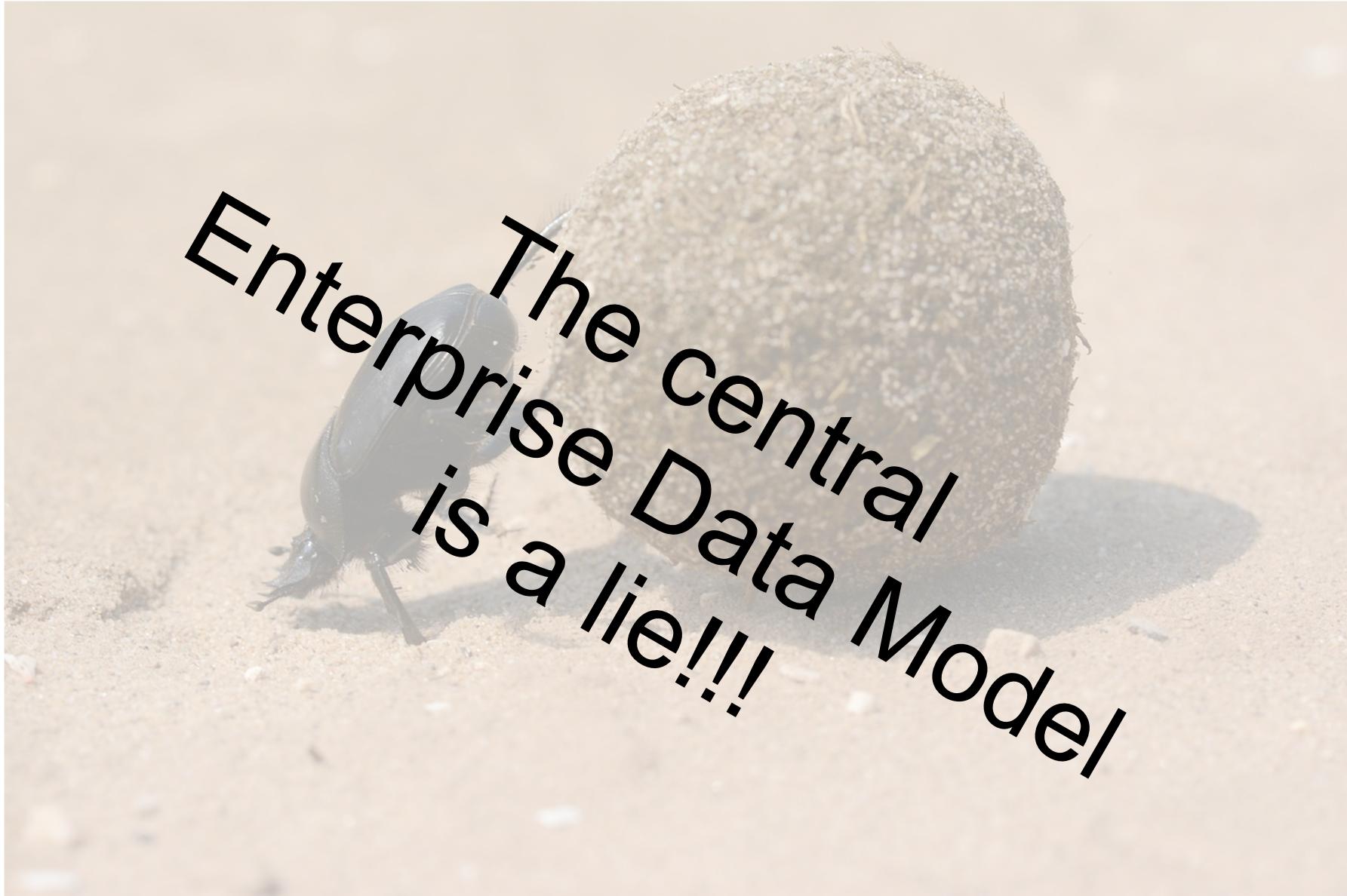
Problem Domäne und Domänen Modell

- Ein Domänen Modell auf abstraktem Level hilft deutlich beim Verständniss der Problemdomäne
 - Es gibt uns eine gemeinsame Sprache
 - Wir wissen wovon wir sprechen
- Dann lasst doch genau um dieses Modell unsere Architektur bauen!
 - Das Enterprise Data Model ... es lebt!
 - ... oder nicht?

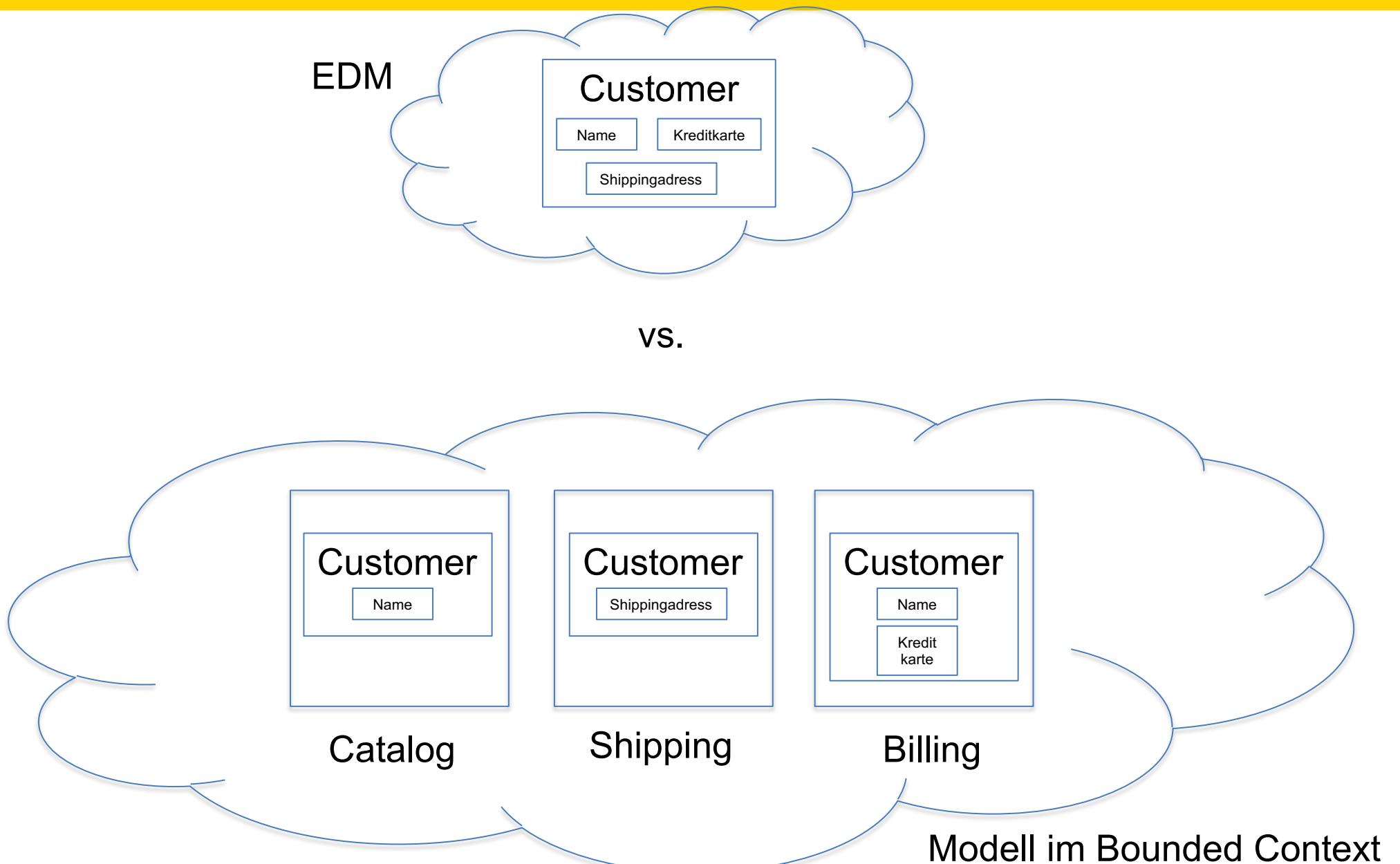
Wohin führen zentrale Datenmodelle?



Irgendwie sind EDMs wie Kuchen ;)



Der Bounded Context als Grenze der Wahrheit



Übung – Let's play cards

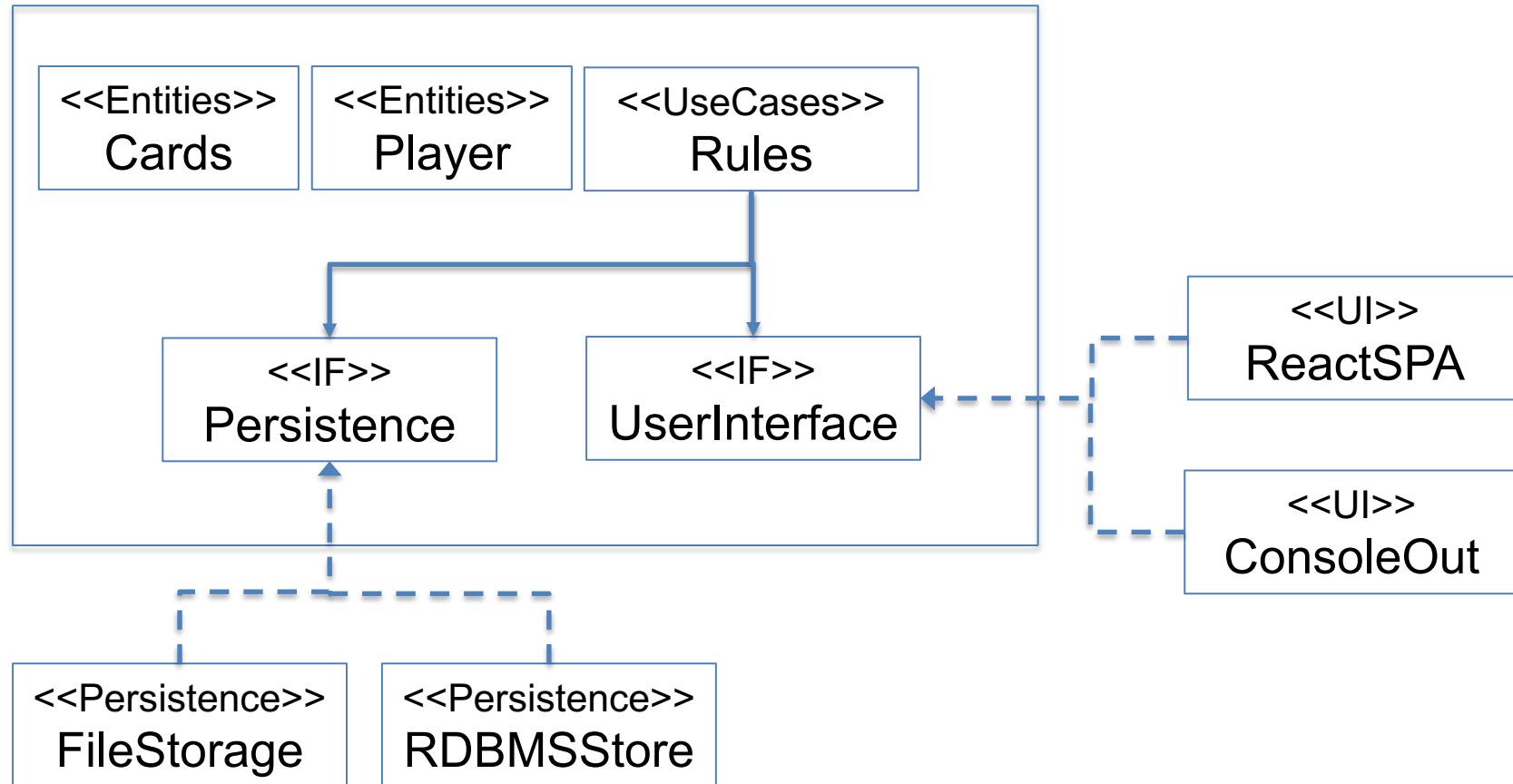


https://de.wikipedia.org/wiki/Dogs_Playing_Poker#/media/File:A_Friend_in_Need_1903_C.M.Coolidge.jpg

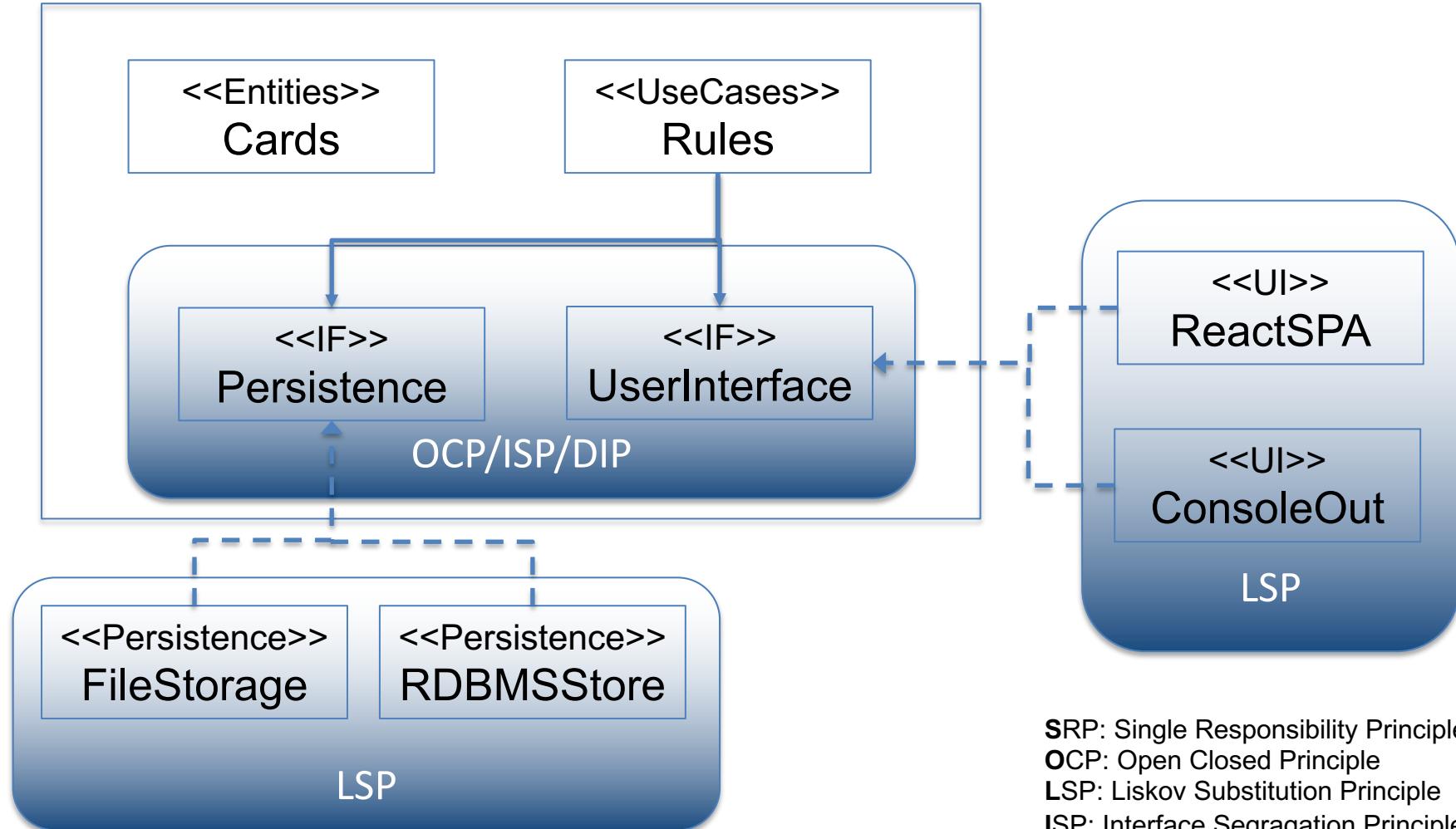
Übung – Die Anforderungen

- Es werden 52 Karten verdeckt ausgeteilt
- Jeder Spieler deckt erst eine, dann eine zweite Karte auf
- Sind beide Karten bzgl. Farbe und Zahl gleich, darf der Spieler sie behalten, ansonsten werden sie wieder verdeckt
- Der Spieler, der am Ende die meisten Pärchen entdeckt hat, hat gewonnen und bekommt einen Rating Punkt
- Das Rating der Spieler wird gespeichert
- Der Prototyp soll auf der Konsole laufen, das spätere Spiel im Web

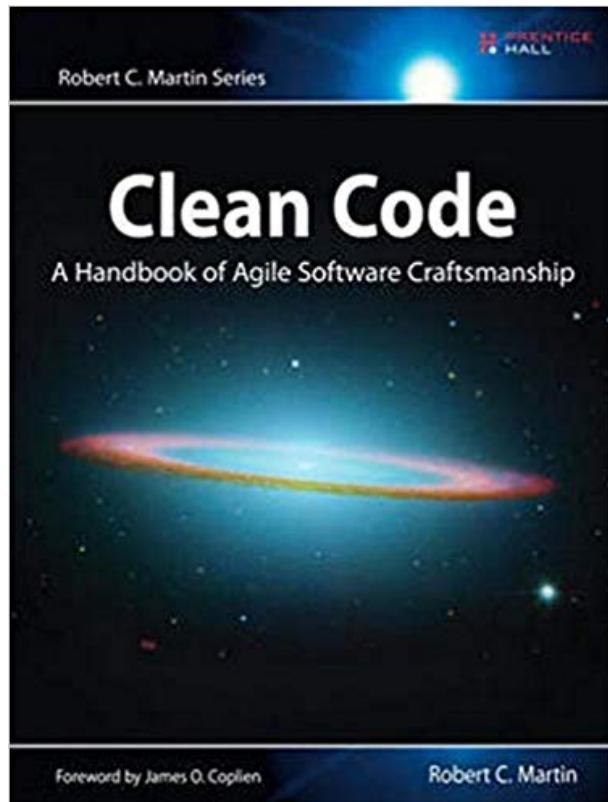
Eine Lösung



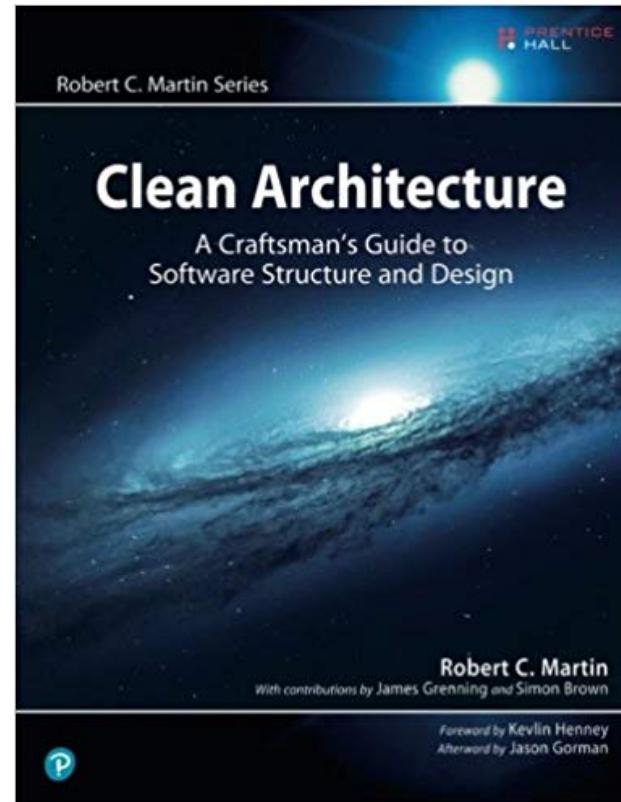
Eine Lösung – SOLID in Action



Design Prinzipien auch auf Architektur Niveau



= ??? =>



Übersicht: Komponenten

- **Component Cohesion:** Verhalten **innerhalb** einer Komponente
 - Reuse/Release Equivalence Principle (REP)
 - Common Closure Principle (CCP)
 - Common Reuse Principle (CRP)
- **Component Coupling:** Verhalten **zwischen** Komponenten
 - Acyclic Dependencies Principle (ADP)
 - Stable Dependencies Principles (SDP)
 - Stable Abstraction Principle (SAP)

Component Cohesion

- Welche Klassen gehören in eine Komponente?
 - REP: Reuse/Release Equivalency Principle
 - CCR: Common Closure Principle
 - CRP: Common Reuse Principle

Component Cohesion - REP

- **Reuse/Release Equivalency Principle (REP)**
 - Komponente soll aus zusammenhängenden Teilen bestehen
 - Problem: Was genau ist „zusammenhängend“?
 - Komponente soll wiederverwendbar sein (reuse)
 - Release
 - Versionierung
 - Dokumentation: Clients müssen selbst entscheiden können, ob und wann sie eine neue Version einer Komponente einbinden

Component Cohesion – CCP (1)

■ Common Closure Principle (CCP)

- Was das Single Responsibility Principle (SRP) für Klassen ist, ist das CCP für Komponenten.
- „*...the Common Closure Principle (CCP) says that a **component** should not have multiple reasons to change*“ (R. C. Martin)
- SRP und CCP in allgemeiner Form:
- „*Gather **together** those things that **change at the same times and for the same reasons**. Separate those things that **change at different times or for different reasons**.*“ (R. C. Martin)

Component Cohesion – CCP (2)

■ Common Closure Principle (CCP)

- CCP hängt auch mit dem Open/Closed Principle (OCP) zusammen:
 - Das **Closed** von OCP (Klassenebene) entspricht dem **Closure** des CCP auf Komponentenebene.

Component Cohesion – CRP

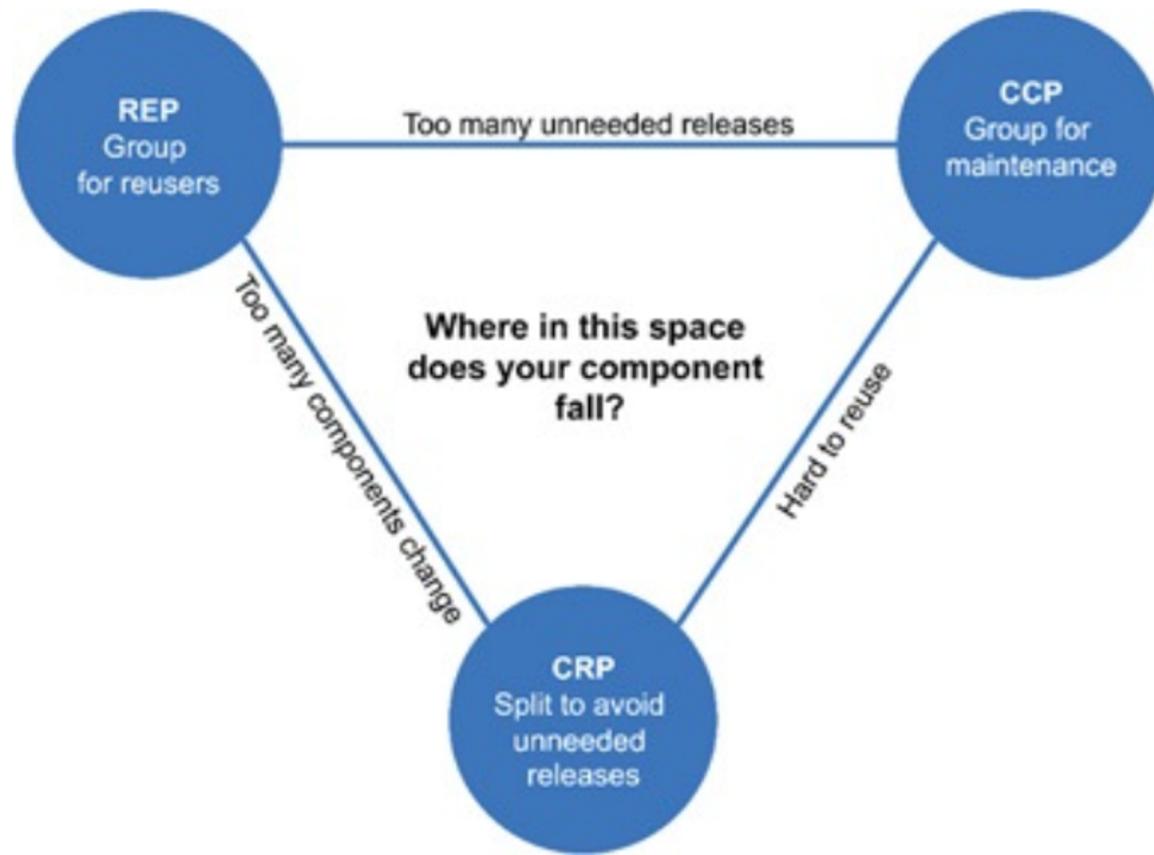
- **Common Reuse Principle (CRP)**
 - „***Don't force users of a component to depend on things they don't need.***“ (R. C. Martin)
 - Dies ist die Architekturen-Variante des Interface Segregation Principles (ISP).
 - Eine Komponente sollte keine Klassen beinhalten, die nicht benötigt werden.
 - Klassen, die sich oft zusammen ändern, sollten in eine Komponente gruppiert werden (hohe Kohäsion)

Component Cohesion – Tension Diagram (1)

- REP und CCP sind **inklusiv**. Ihre Anwendung führt dazu, dass Komponenten **größer** werden
- CRP ist **exklusiv** und führt zu **kleineren** Komponenten
- Das Spannungsfeld zwischen diesen 3 Prinzipien wird im Tension Diagram beschrieben

Component Cohesion – Tension Diagram (2)

- Die Kantenbeschriftungen beschreiben die Kosten, die bei Verletzung der gegenüberliegenden Ecke auftreten.



Quelle: „Clean Architecture“ R.C. Martin

Component Cohesion – Zusammenfassung

- Das Zusammenspiel zwischen
 - Reuse/Release Equivalence Principle (REP),
 - Common Closure Principle (CCP) und
 - Common Reuse Principle (CRP)
- zeigt, dass das Thema Kohäsion in der Software-Architektur komplexer ist als auf Klassenebene.
- Themen wie Wiederverwendbarkeit und „develop-ability“ müssen über die Zeit immer wieder gegeneinander abgewogen und neu bewertet werden.

Component Coupling

- **Component Coupling**
 - Die Beziehung **zwischen** Komponenten

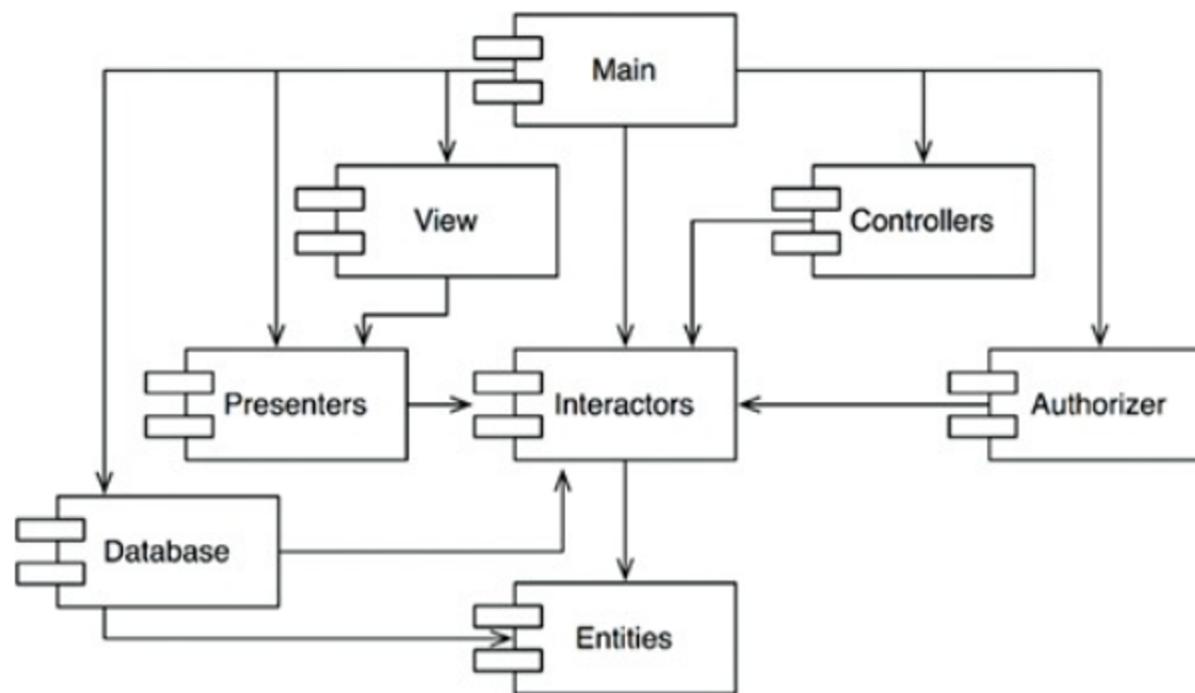
Component Coupling - ADP

■ Acyclic Dependencies Principle (ADP)

- „***Allow no cycles in the component dependency graph.***“ (R. C. Martin)
- Komponenten unabhängig voneinander entwickeln und deployen (mit Versionierung)
- Die Struktur der Abhängigkeiten muss gepflegt werden!

Component Coupling - ADP

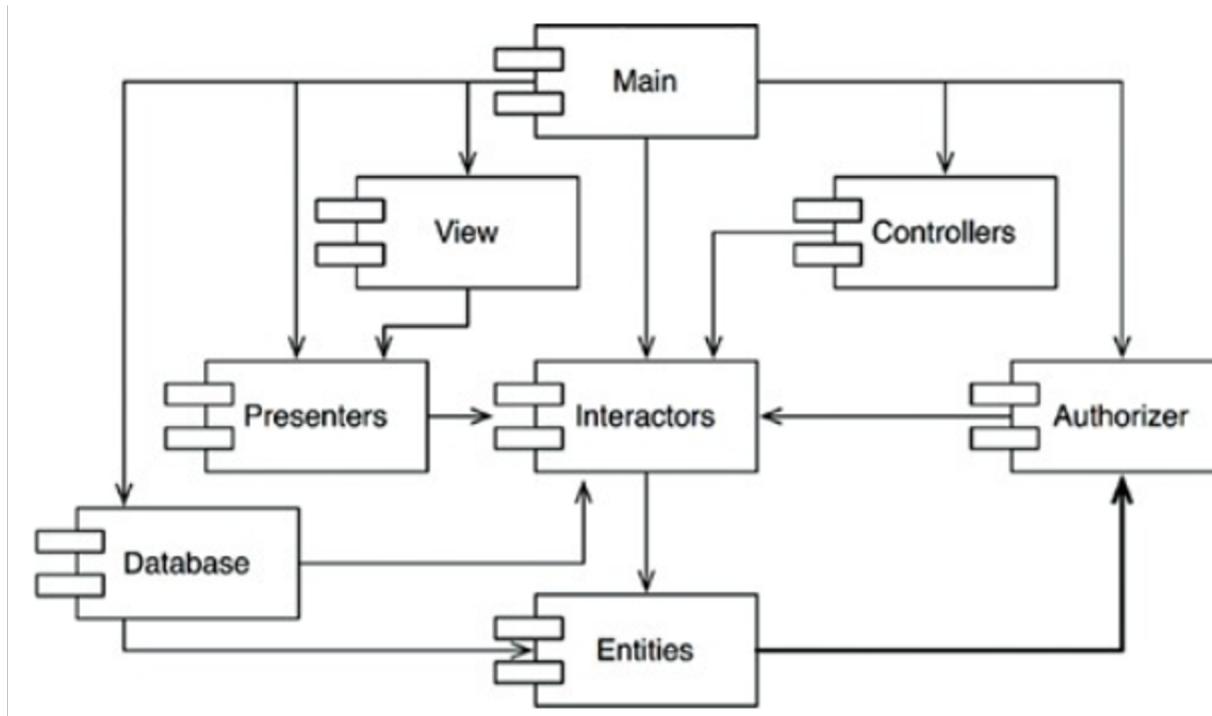
- Saubere Abhängigkeiten (keine Zyklen)



- Quelle: „Clean Architecture“ R. C. Martin

Component Coupling - ADP

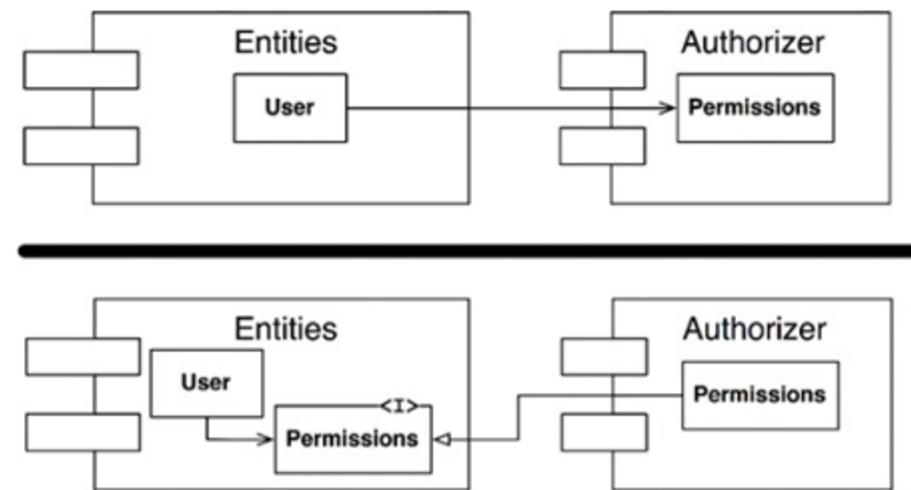
- Achtung: Zyklische Abhangigkeit!



- Quelle: „Clean Architecture“ R. C. Martin

Component Coupling - ADP

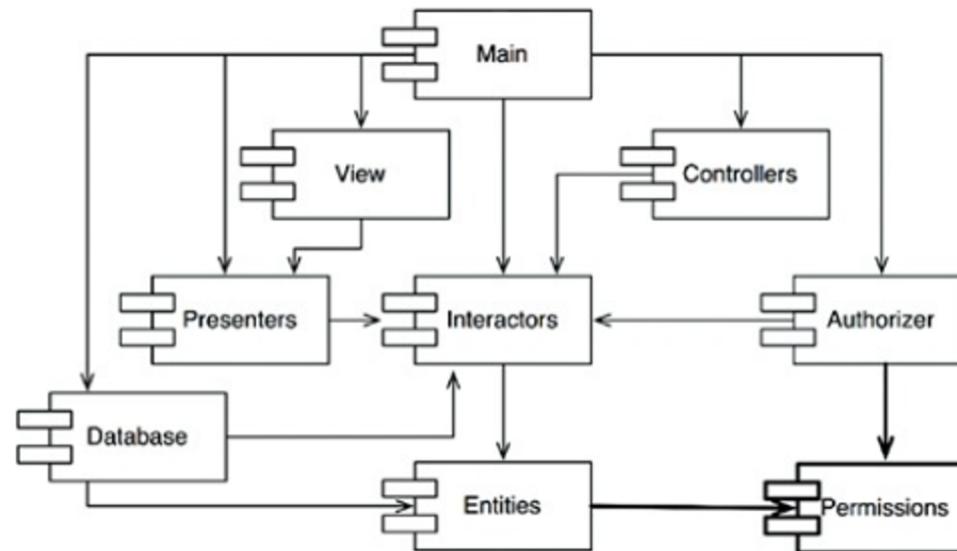
- **Breaking the Cycle (1/2): Dependency Inversion Principle (DIP)**



- Quelle: „Clean Architecture“ R. C. Martin

Component Coupling - ADP

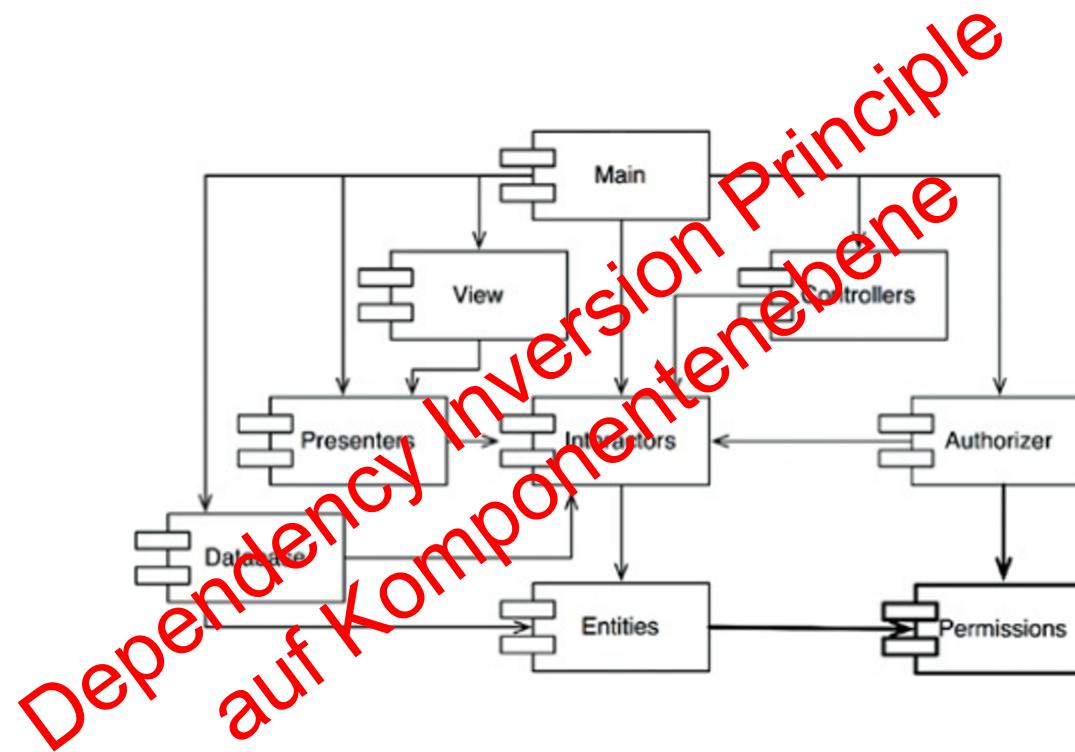
- **Breaking the Cycle (2/2): Komponente mit Interfaces extrahieren**



- Quelle: „Clean Architecture“ R. C. Martin

Component Coupling - ADP

- **Breaking the Cycle (2/2): Komponente mit Interfaces extrahieren**



- Quelle: „Clean Architecture“ R. C. Martin

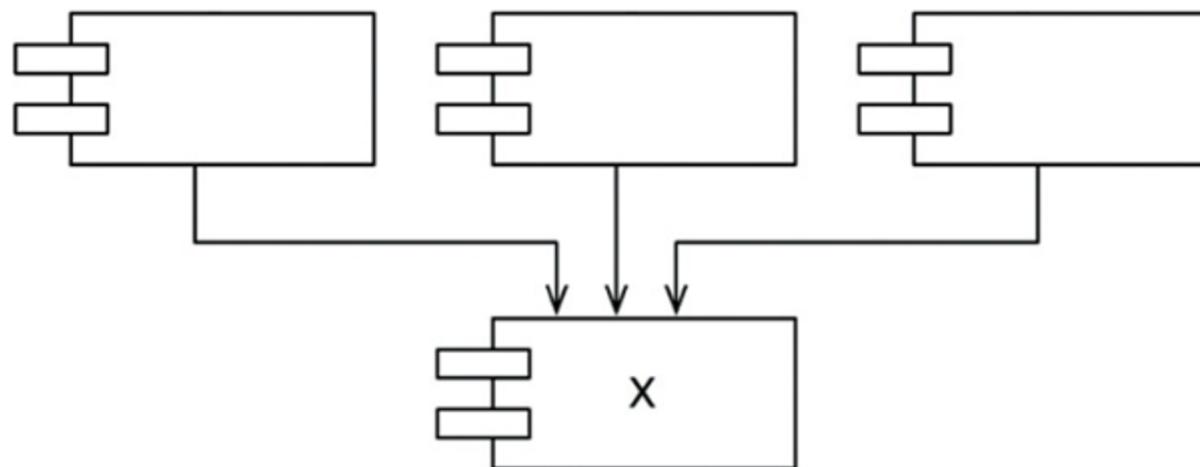
■ Stable Dependencies Principle (SDP)

- „*Depend in the direction of stability.*“ (R. C. Martin)
- Nicht alle Komponenten können stabil sein. Wir entwickeln auch ganz bewusst Komponenten, die sich häufig ändern (siehe Common Closure Principle CCP und Open/Closed Principle OCP).
- Sich häufig ändernde Komponenten sollten keine Abhängigkeiten auf Komponenten haben, die sich auch häufig ändern.

Component Coupling - SDP

■ Was ist Stabilität?

- Komponente ist **stabil**, wenn
 - viele andere Komponenten auf sie verweisen
 - die Komponente selbst keine Abhängigkeiten hat

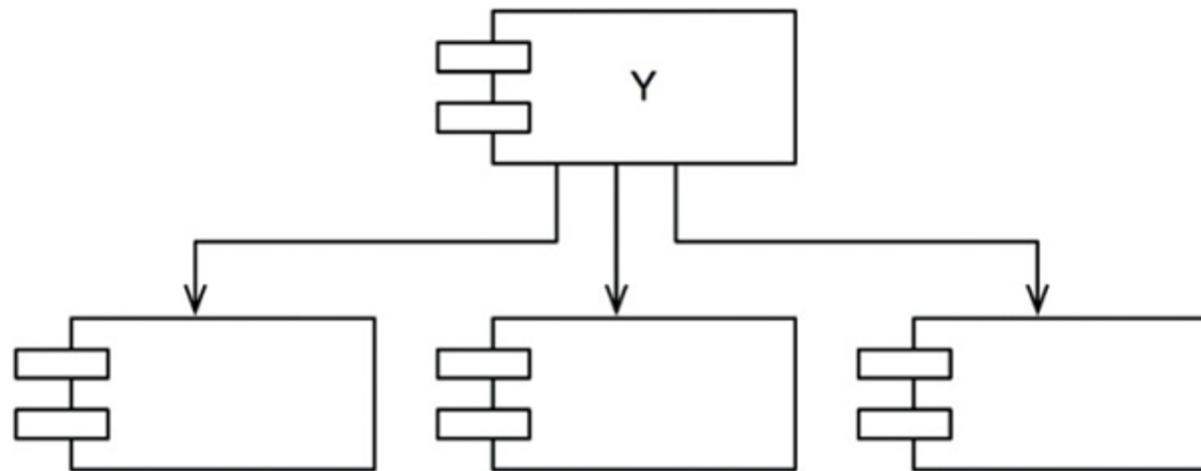


- Quelle: „Clean Architecture“ R. C. Martin

Component Coupling - SDP

■ Was ist Stabilität?

- Komponente ist **instabil**, wenn
 - keine andere Komponenten auf sie verweisen
 - die Komponente viele Abhängigkeiten hat



- Quelle: „Clean Architecture“ R. C. Martin

Component Coupling – SDP (Metrik)

■ Wie misst man Stabilität?

■ I – Fan-out / (Fan-in + Fan-out)

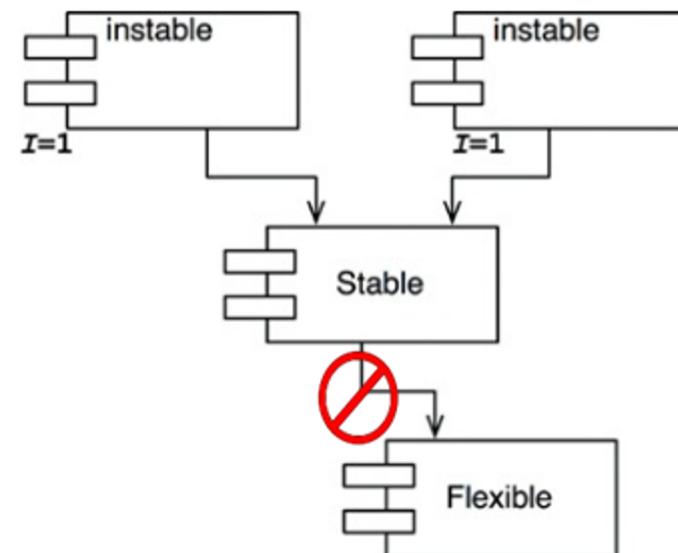
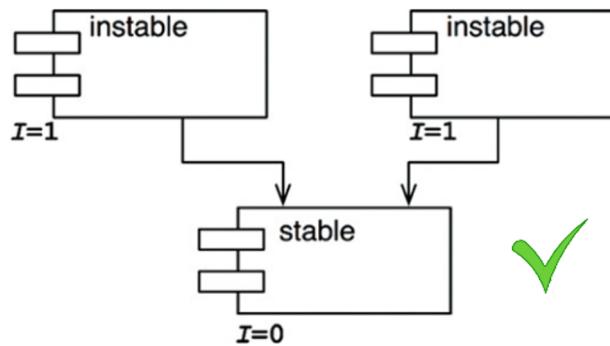
- I: Instability (zwischen 0 und 1)
- Fan-in: Anzahl Komponenten, die Abhängigkeit auf diese Komponente haben
- Fan-out: Anzahl an ausgehenden Abhängigkeiten

■ Kurzfassung

- **I = 1:** keine Komponente hat Abhängigkeit auf diese Komponente
- **I = 0:** andere Komponenten haben eine Abhängigkeit auf diese Komponente, und diese Komponente selbst hat keine Abhängigkeiten

Component Coupling – SDP (Metrik)

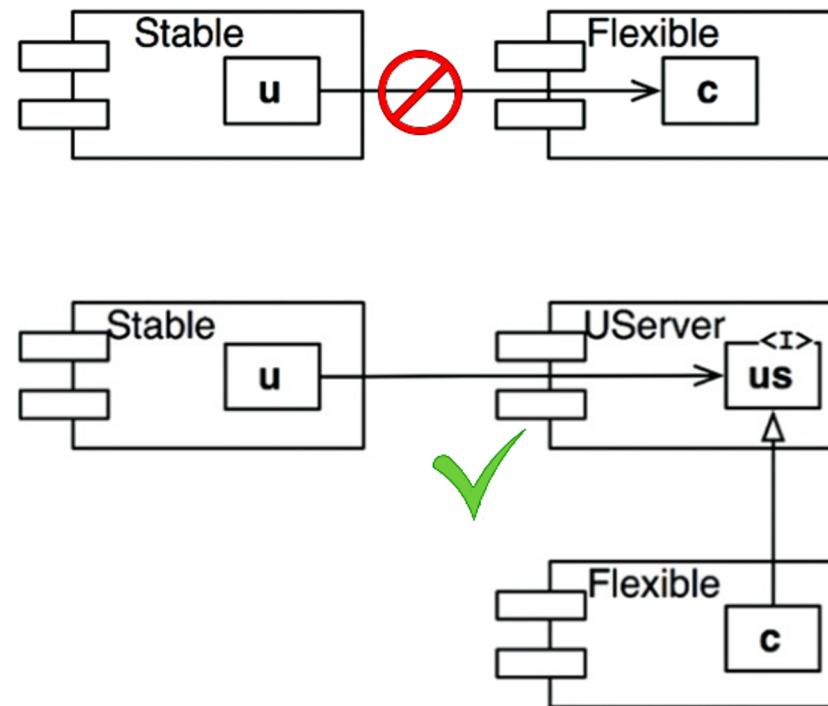
- Faustregel: I soll in Richtung der Abhangigkeit kleiner werden



- Quelle: „Clean Architecture“ R. C. Martin

Component Coupling – SDP

- Wie löst man das Problem?
 - DIP mit eigener Komponente für Interfaces (gängiges Pattern!)



- Quelle: „Clean Architecture“ R. C. Martin

Component Coupling – SAP

■ Stable Abstraction Principle (SAP)

- Das Stable Abstraction Principle beschreibt das Verhältnis von Abstraktion zu Stabilität.
- Eine stabile Komponente sollte abstrakter sein als eine instabile Komponente.
- Die Kombination aus SAP und SDP ergeben zusammen das Dependency Inversion Principle (DIP) auf Komponentenebene.

Component Coupling – SAP (Metrik)

■ Abstraktion messen

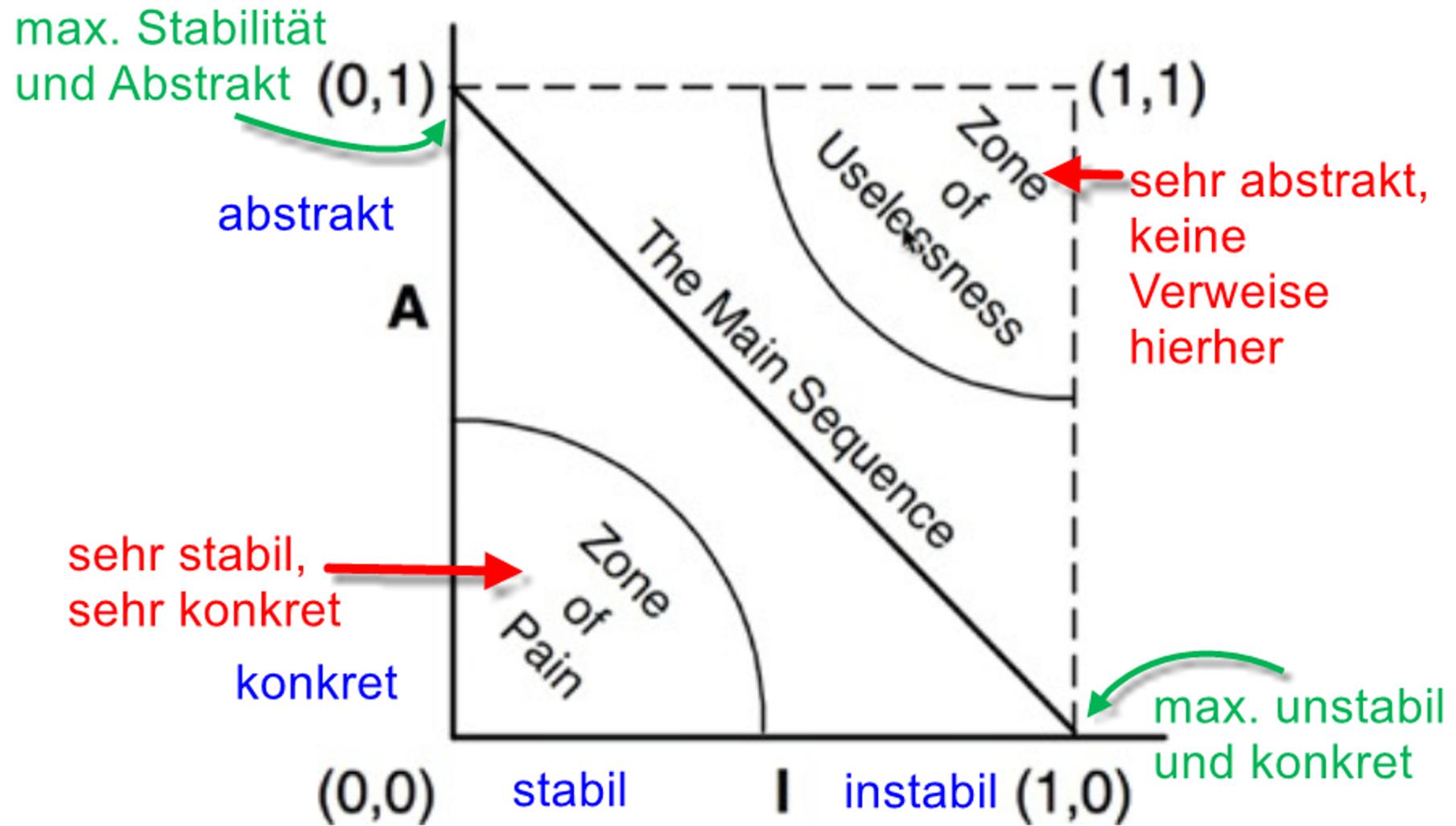
$$\blacksquare A = Na / Nc$$

- A: Abstraktion (zw. 0 und 1)
- Na: Anzahl an abstrakten Klassen in einer Komponente
- Nc: Anzahl aller Klassen einer Komponente

■ Kurzfassung

- **A = 0: Komponente ohne abstrakte Klassen/Interfaces**
- **A = 1: Komponente mit ausschließlich abstrakten Klassen**

Component Coupling – Metrik



Quelle: „Clean Architecture“ R. C. Martin

Nicht alles unreflektiert anwenden



Zusammenfassung



"doopsuiker en kaartje" by loufi is licensed under CC BY 2.0

Clean Architecture the good parts

- Screaming Architecture
 - Fachlichkeit im Zentrum
- Hexagonal/Onion/Clean Architecture
 - Technik als Detail
- SOLID Prinzipien auf Architektur Level

Clean Architecture meets Microservices

- Ideengeber für die Micro-Architektur
- Makro-Architektur ist i.d.R. Technik
 - Clean Architecture bietet Ansätze diese zu entkoppeln

Wie weit kann ich Technik wirklich abstrahieren

- Wie Technik die Möglichkeiten der Fachlichkeit doch einschränkt
 - „Das können wir bauen, aber das wird langsam werden“
 - Der „Gewinn“ muss dabei wieder auf Fachlichkeit einzahlen und nicht „nur“ die neue Hype Technologie ermöglichen
- Vorbereiten aber nicht zwingend durchziehen

Fazit

- Um schnell auf Änderungen reagieren zu können, müssen diese möglichst lokal gehalten werden
 - Trennung von Technik und Fachlichkeit
 - Komponenten Cohäsion beachten (SRP)
- Technik ändert sich schnell, Fachlichkeit selten
 - Fachlichkeit ins Zentrum stellen
 - Technik als Detail
- Nicht alles als „bare Münze“ nehmen
 - Gute als Ideengeber, aber selber denken ist weiterhin erlaubt
- Nicht ins Akademische übersteigern!

Zum Schluss doch noch Code...

```
<!-- Navbar -->
<div class="w3-top">
  <div class="w3-bar w3-black w3-hide-large w3-hide-small w3-right" href="javascript:void(0)" title="Toggle Navigation Menu"><i class="fa fa-bars"></i> HOME</a>
    <a href="#" class="w3-bar-item w3-button w3-padding-large w3-hide-small" href="#band" class="w3-bar-item w3-button w3-padding-large w3-hide-small">BAND</a>
    <a href="#" class="w3-bar-item w3-button w3-padding-large w3-hide-small" href="#" class="w3-bar-item w3-button w3-padding-large w3-hide-small">TOUR</a>
    <a href="#" class="w3-bar-item w3-button w3-padding-large w3-hide-small" href="#" class="w3-bar-item w3-button w3-padding-large w3-hide-small">CONTACT</a>
    <div class="w3-dropdown-hover w3-hide-small">
      <button class="w3-padding-large w3-button" title="More">MORE <i class="fa fa-caret-down"></i></button>
      <div class="w3-dropdown-content w3-bar-block w3-card-4">
        <a href="#" class="w3-bar-item w3-button">Merchandise</a>
        <a href="#" class="w3-bar-item w3-button">Extras</a>
        <a href="#" class="w3-bar-item w3-button">Media</a>
      </div>
    </div>
    <a href="javascript:void(0)" class="w3-bar-item w3-button w3-hide-small w3-right" href="#" class="w3-bar-item w3-button w3-padding-large w3-hide-small">LOGO</a>
  </div>
</div>
```

Fragen?

Vielen Dank!

werner.eberling@mathema.de

(@wer_eb)

hristiyan.pehlivanov@mathema.de

MATHEMA

https://github.com/wern/CleanArchitecture_AS20