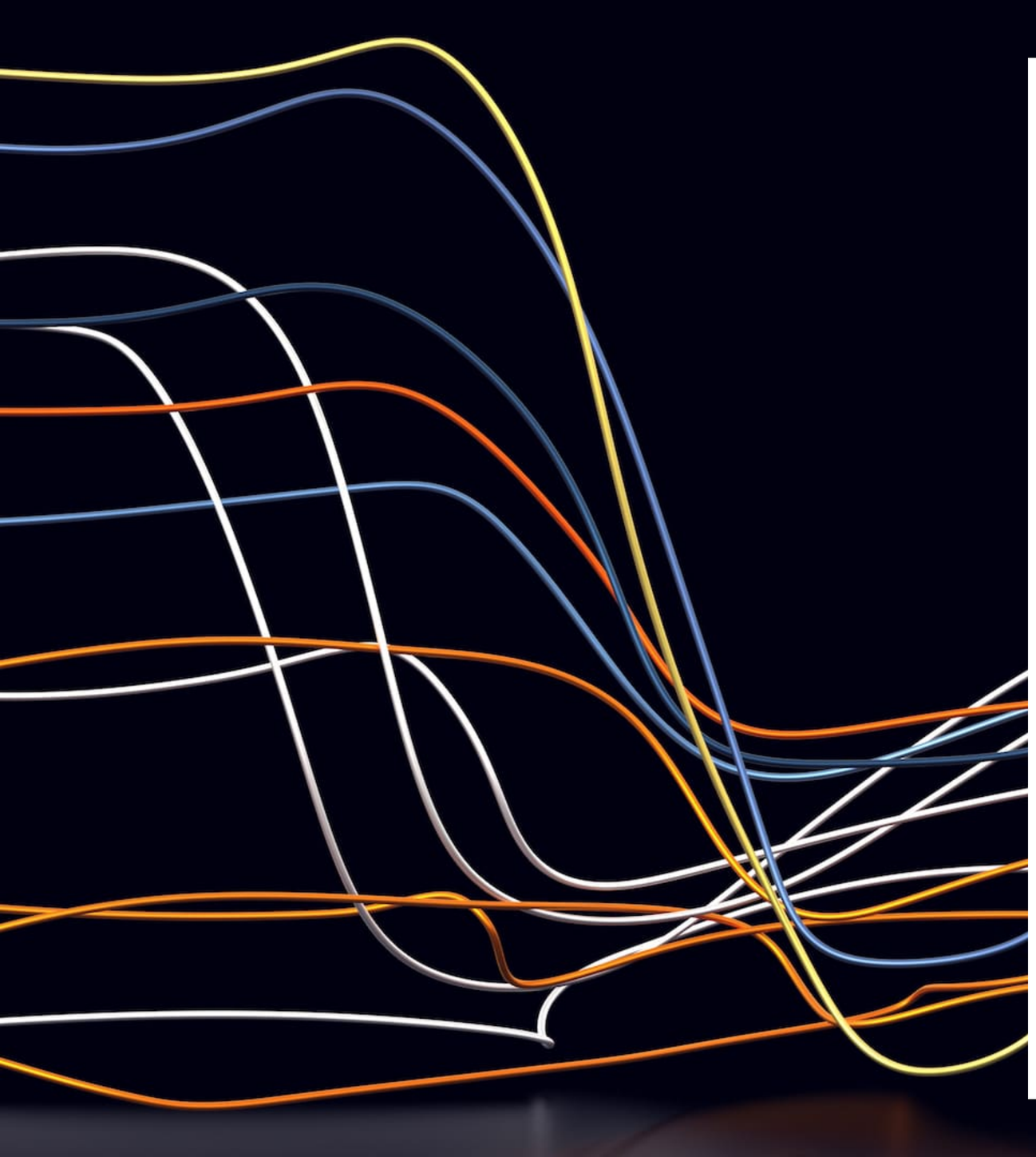


Technology Day 2024

"Domain-Driven Design? Das ist doch nur Bloat!"

— eine Geschichte voller Missverständnisse und schlecht formulierter Wünsche

INNOQ

FABIAN KRETZER
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 -> Clean-, Onion- und Konsorten, DDD (erst strategisch)

(De)Motivation

„Der Code ist zu komplex!“

„Warum die vielen Klassen?“

„Warum dieses ständige "gemappe" von Werten?“

„Warum gibt es mehrere Klassen die "Auftrag" heißen?“

„Können wir die Klasse "Kunde" nicht in eine shared Library verschieben? Die brauchen wir doch überall.“

„Das System ist viel zu komplex!“

„Warum haben wir 65 Microservices???“

Haben der Einsatz von DDD-Patterns und domänenzentrischer Architekturstile Systeme und Code unnötig komplex gemacht?

Spoiler:

Diese Annahme könnte etwas mit dem Unterschied zwischen Kompliziertheit und Komplexität zu tun haben.

Kompakter, minimalistischer Code kann hohe Komplexität verbergen.

Geäußerte Kritik und Unmut sind meist schlecht formulierte Wünsche.

Geäußerte Kritik und Unmut sind meist schlecht formulierte Wünsche.

Nach?

Geäußerte Kritik und Unmut sind meist schlecht formulierte Wünsche.

Nach?

Kontext!

Welche Personen haben unter welchen Bedingungen welche Entscheidungen getroffen?

Perfektes gemeinsames (Domänen-)Verständnis

Perfektes gemeinsames (Domänen-)Verständnis

Wie erlange ich es?

Wie erhalte ich es?

Wie passe ich es an?

Perfektes gemeinsames (Domänen-)Verständnis

Wie erlange ich es?

Wie erhalte ich es?

Wie passe ich es an?

- **Spoiler: Nie**
- **Spoiler: Doku reicht nicht**
- **Domäne typsicher modellieren**

Perfektes gemeinsames (Domänen-)Verständnis

Wie erlange ich es?

Wie erhalte ich es?

Wie passe ich es an?

- **Spoiler: Nie**
- **Spoiler: Doku reicht nicht**
- **Domäne typsicher modellieren**

„If Auftragsnummer starts with 42, don't send Rechnung."

Mehrwert entsteht durch das explizite Modellieren der Domäne.

Dadurch wird Unterstützung der Menschen in ihren Prozesse ermöglicht und und die Software steht ihnen nicht durch versteckte Komplexität im Weg.

Gemeinsame Grundlagen

Komplex vs. Kompliziert

„Der Code ist zu komplex!“

„Warum gibt es mehrere Klassen die "Auftrag" heißen?“

Tackling Complexity

System zeigt ein emergentes, nicht vorhersehbares Verhalten durch Rückkopplungseffekte.^1

^1 https://www.zdf.de/wissen/frag-den-lesch/komplex-oder-kompliziert---was-macht-den-unterschied-100.html

Tackling Complexity

Heißt:

Creating Complicatedness?

Viele kleine, gut abgegrenzte Dinge mit klaren Schnittstellen werden zu einem komplizierten Ganzen.

Kompliziert heißt: nachvollziehbar.

Fachlich

- Reduzierung bidirektionaler Beziehungen

Technisch

- Verschieben in andere Layer (später mehr)

Menschlich?

Möchte ich das als Organisation?

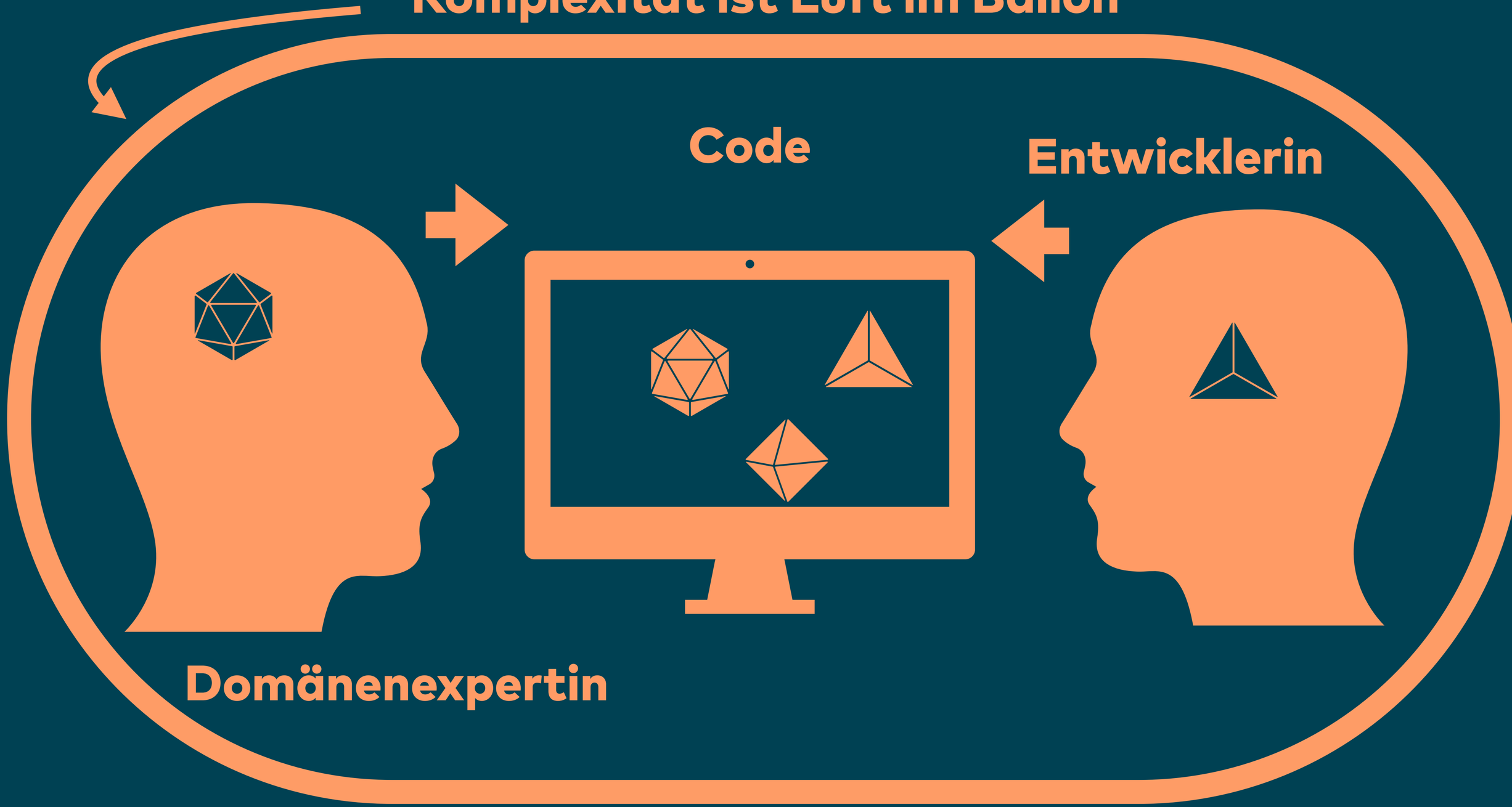
ICH will würde das nicht pauschal einschränken. Würde Kreativität und Innovation abwürgen.

Soziotechnisch!

https://www.innoq.com/de/articles/2022/03/die-angemessenheit-von-komplexitaet/

Komplexität ist Luft im Ballon

Komplexität ist Luft im Ballon



Refresher: Was steckt hinter den taktischen DDD-Patterns?

Schritt zurück: Da fehlt doch was?

„Das System ist viel zu komplex!“

Beim Strategic Design nicht dabei gewesen?

Mein Favorit: Eventstorming

Hoch interaktives Format!

Der eigentliche Wert sind nicht die Artefakte, sondern die Interaktion.

When in doubt, use a Canvas

Architecture Inception Canvas

https://canvas.arc42.org/architecture-inception-canvas

Architecture Communication Canvas

https://canvas.arc42.org/architecture-communication-canvas

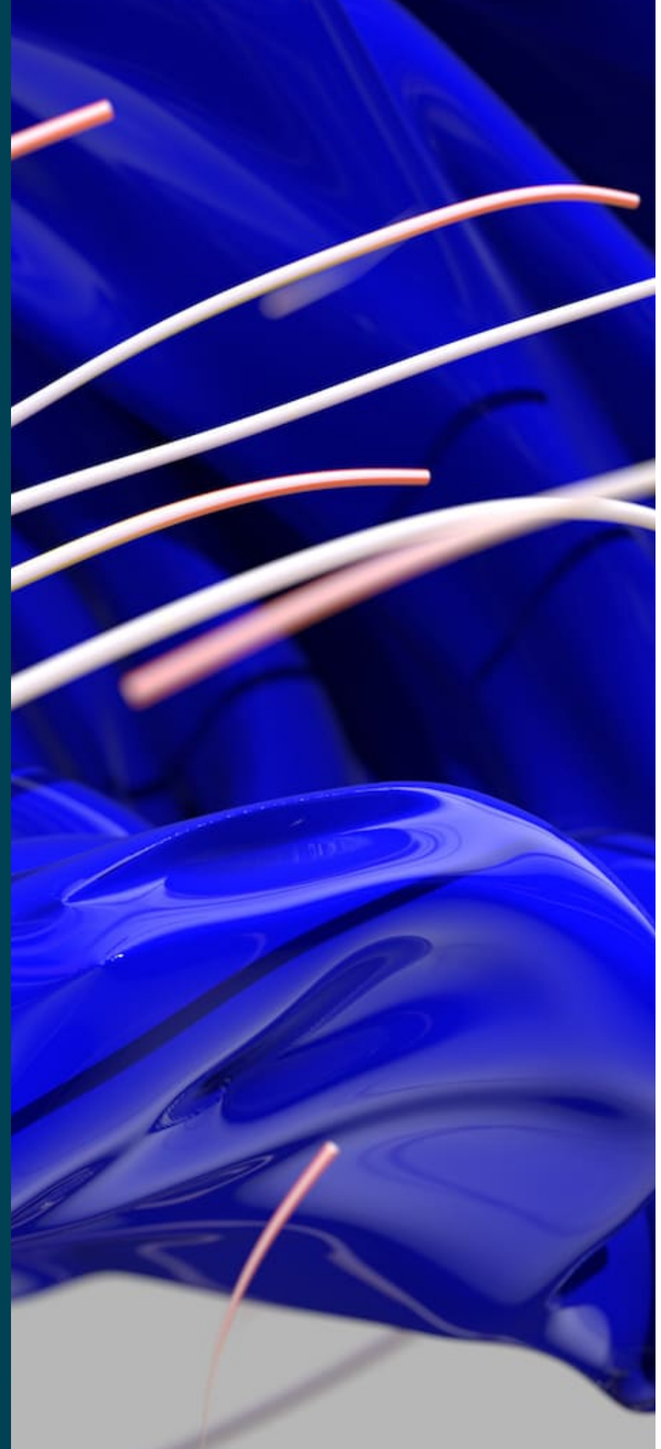
Reden, Reden und Reden

Die meisten Artefakte haben gemein: Es fehlt der Entscheidungsprozess, der zum Ergebnis führte. Kontext!

Jetzt aber: Was steckt hinter den taktischen DDD-Patterns?

„Die Klassenhierarchie ist falsch. In der Realität ist das ja ganz anders."

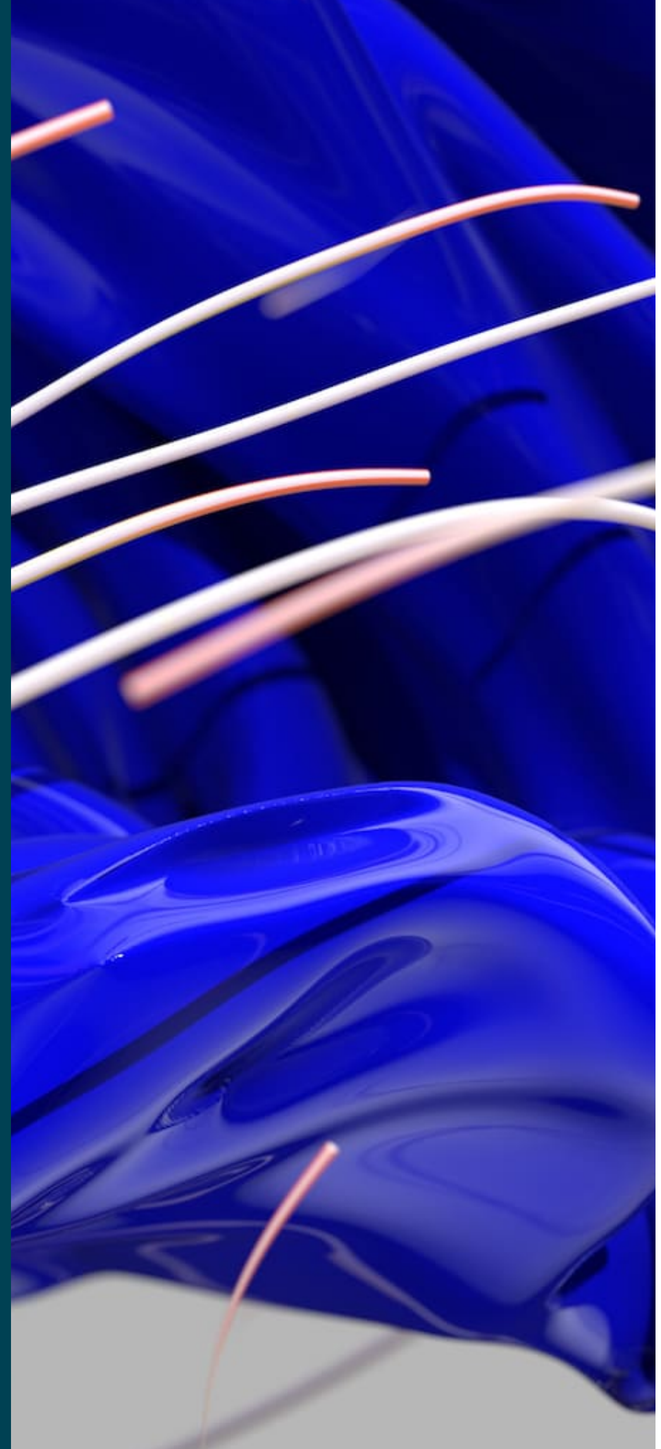
Von: „Habe ich recht zügig verstanden„ bis „Da knabbere ich heute noch dran“



Building Blocks

1. Value Objects
2. Entities
3. Repositories
4. Factories
5. Services
6. Modules
7. Aggregates

Von: „Habe ich recht zügig verstanden„ bis „Da knabbere ich heute noch dran“



Building Blocks

1. Value Objects
2. Entities
3. Repositories
4. Factories
5. Services
6. Modules
7. Aggregates

Das einfachste mögliche Modell meiner Domäne, was mich unterstützt mein Ziel zu erreichen.

„A model is a lie that helps you see the truth“

Howard Skipper

"All models are wrong but some are useful"

George Box

Ein perfektes Modell der Realität wäre die Realität.

Und die Realität ist unendlich komplex.

Ein perfektes Modell der Realität wäre die Realität.

Und die Realität ist unendlich komplex.

Modelle sind toll. Lasst uns viele kleine haben, die wir gut verstehen.

Die Cleane Zwiebel und das DDD

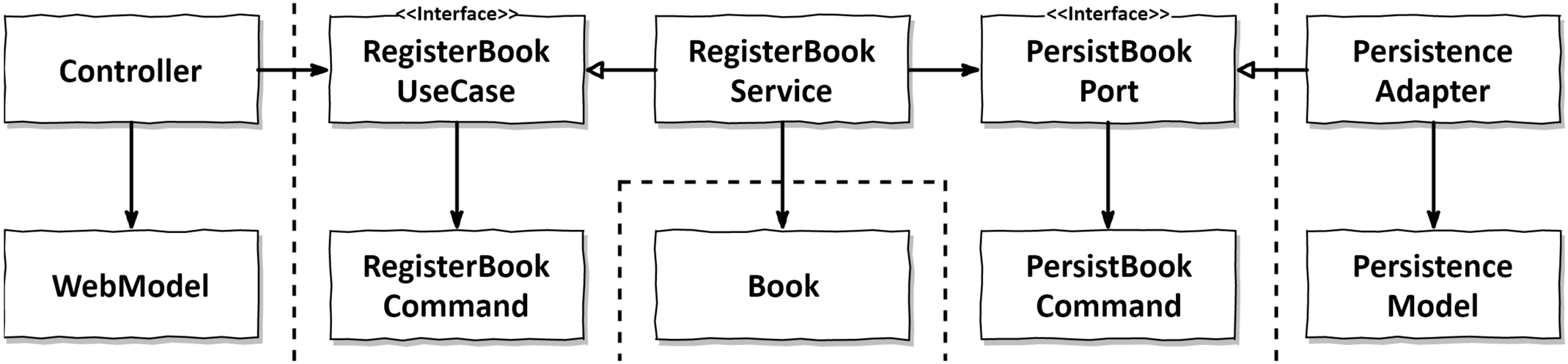
„Warum dieses ständige hin und her "gemappe" von Werten?"

Clean Architecture passt einfach gut

Domäne im Zentrum, Layered Architecture, Divide & Conquer, Domäne von Infrastruktur getrennt

Mappingstrategien

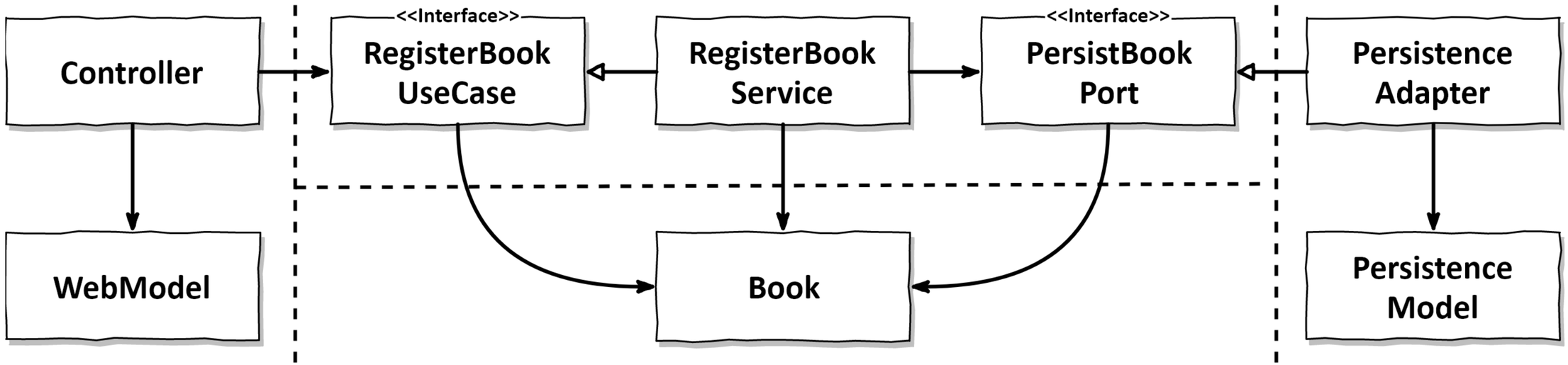
Vielleicht war da wer übermotiviert?



(c) Tom Hombergs - https://speakerdeck.com/thombergs/o-2019

Mappingstrategien

Es geht auch einfach(er)



Start simple

YAGNI

Das kleinstmögliche Modell der Domäne ins Zentrum stellen.

Technische Infrastruktur abtrennen.

Divide and Conquer.

In die Praxis

Teamstruktur & Umfeld

„Warum haben wir 65 Microservices???“

Modularisierungsansätze

Qualitätsziele

Bei Bestandssystemen ohne Doku:

„Reverse Qualitätstaktik“

Quality Reverse Engineering

Welche Patterns finde ich?

1. Auf welche Qualitätsziele zahlen sie ein?
2. War das YAGNI oder bewusst?
3. Ggf. anpassen

Siehe Markus Harrer - https://leanpub.com/qualitaetstaktiken

Endlich (!) (Java) Coden

„Warum die vielen Klassen?“

Nicht gegen Frameworks arbeiten

Technische (Spring, Hibernate)

und menschliche (Vorwissen)

Nicht gegen Frameworks arbeiten

JPA- & Spring-Annotations im Domain Layer 😱

Keine Schande

Und wie modelliere ich jetzt diese ominöse Domäne?

Und wie modelliere ich jetzt diese ominöse Domäne?

Funktional!

Functional domain modeling

- Scott Wlaschin hats am besten beschrieben…
- … „Leider“ in F#
- Grundgedanken lassen sich übertragen
- Value Objects^3 -> Java Records
- Moderne Java Features decken das ab:
- Sealed Class Hierarchies und
- Switch-Expressions 😍
- Kurz: Nutz alles was das Typsystem hergibt für die Modellierung

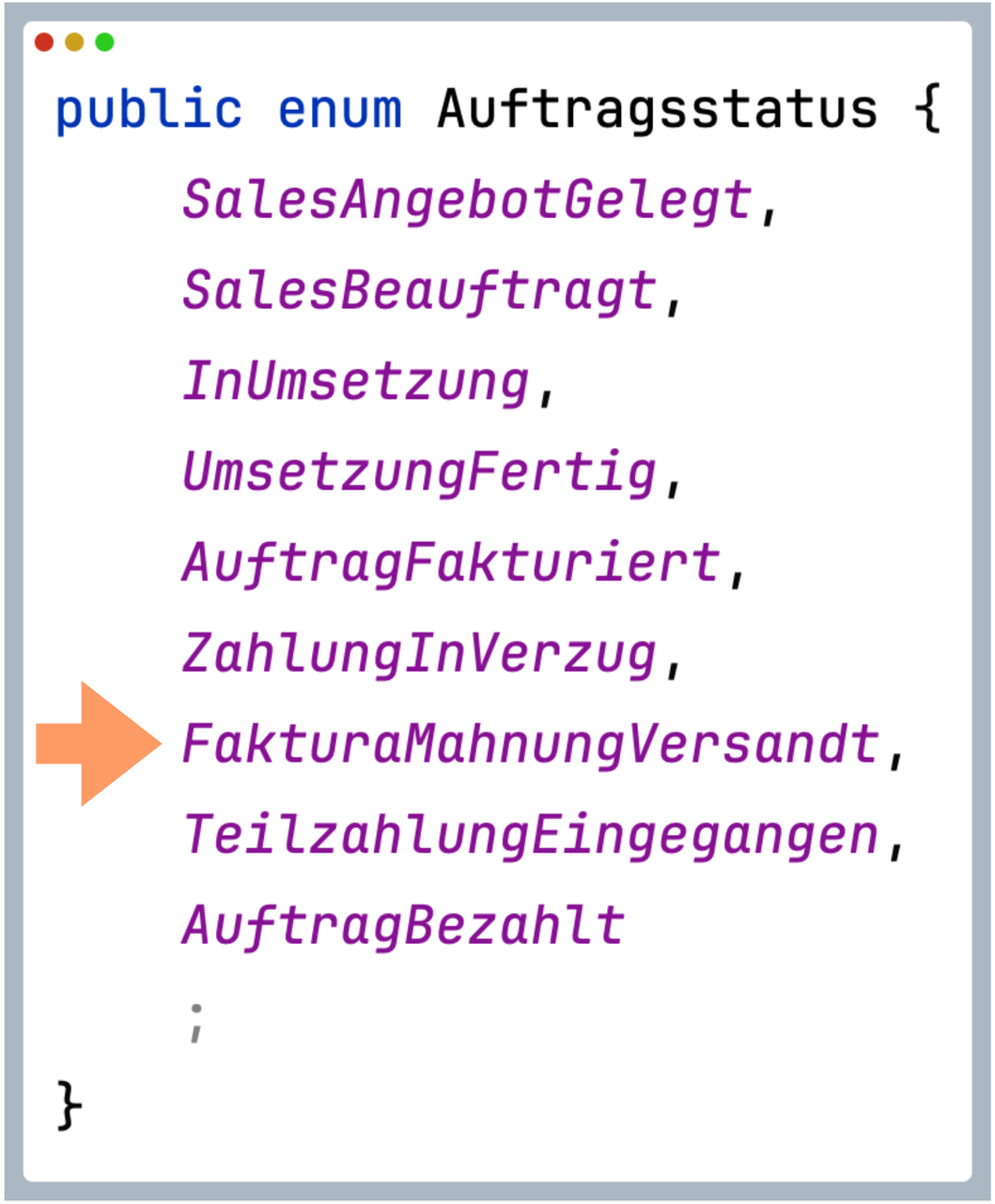
„Können wir die Klasse "Kunde" nicht in eine shared Library verschieben? Die brauchen wir doch überall.“

„Warum die vielen Klassen?“

Beispiel: Enum Auftragsstatus

```
public enum Auftragsstatus {
    SalesAngebotGelegt,
    SalesBeauftragt,
    InUmsetzung,
    UmsetzungFertig,
    AuftragFakturiert,
    ZahlungInVerzug,
    //FakturaMahnungVersandt,
    TeilzahlungEingegangen,
    AuftragBezahlt
    ;
}
```

```java
public class BenachrichtigungService {
    private KundenBenachrichtigung kundenBenachrichtigung;
    private TeamBenachrichtigung teamBenachrichtigung;
    private SalesBenachrichtigung salesBenachrichtigung;
    private FakturaBenachrichtigung fakturaBenachrichtigung;
    Results benachrichtigen(Auftragsstatus auftragsstatus) {
        return switch (auftragsstatus) {
            case SalesAngebotGelegt -> kundenBenachrichtigung.send();
            case SalesBeauftragt -> teamBenachrichtigung.send().with(salesBenachrichtigung.send());
            case InUmsetzung -> kundenBenachrichtigung.send();
            case UmsetzungFertig -> salesBenachrichtigung.send();
            case AuftragFakturiert -> kundenBenachrichtigung.send();
            case ZahlungInVerzug -> fakturaBenachrichtigung.send();
            case TeilzahlungEingegangen -> fakturaBenachrichtigung.send();
            case AuftragBezahlt -> salesBenachrichtigung.send().with(fakturaBenachrichtigung.send());
        };
    }
}
```


public enum Auftragsstatus {
SalesAngebotGelegt,
SalesBeauftragt,
InUmsetzung,
UmsetzungFertig,
AuftragFakturiert,
ZahlungInVerzug,
FakturaMahnungVersandt,
TeilzahlungEingegangen,
AuftragBezahlt
;
}

```
public class BenachrichtigungService {
    private KundenBenachrichtigung kundenBenachrichtigung;
    private TeamBenachrichtigung teamBenachrichtigung;
    private SalesBenachrichtigung salesBenachrichtigung;
    private FakturaBenachrichtigung fakturaBenachrichtigung;
    Results benachrichtigen(Auftragsstatus auftragsstatus) {
        return switch (auftragsstatus) {
            case SalesAngebotGelegt -> kundenBenachrichtigung.send();
            case SalesBeauftragt -> teamBenachrichtigung.send().with(salesBenachrichtigung.send());
            case InUmsetzung -> kundenBenachrichtigung.send();
            case UmsetzungFertig -> salesBenachrichtigung.send();
            case AuftragFakturiert -> kundenBenachrichtigung.send();
            case ZahlungInVerzug -> fakturaBenachrichtigung.send();
            case TeilzahlungEingegangen -> fakturaBenachrichtigung.send();
            case AuftragBezahlt -> salesBenachrichtigung.send().with(fakturaBenachrichtigung.send());
        };
    }
}
```

```
public enum SalesAuftragsstatus {
    SalesAngebotGelegt,
    SalesBeauftragt,
    InUmsetzung,
    AuftragBezahlt
    ;
}
```

```
public enum FakturaAuftragsstatus {
    UmsetzungFertig,
    AuftragFakturiert,
    ZahlungInVerzug,
    FakturaMahnungVersandt,
    TeilzahlungEingegangen,
    AuftragBezahlt;
}
```

Beispiel: String23

```java
public record Name(String23 name) {}

public record String23(String string23) {
    public String23 {
        if (string23.length() > 23) {
            throw new IllegalArgumentException("String23 should have length 23");
        }
    }
}
```

Beispiel: Validated E-Mail

```
public record Email(String emailAddress) {
}
```

```
public record VerifiedEmail(String emailAddress) {
}
```

```java
public interface SendImportantInformation {
    void send(VerifiedEmail email);
}
```

```java
/**
 * Inspired by Scott Wlaschin
 */
public interface EmailVerifier {
    VerifiedEmail verify(Email email);
}
```

Aggregate und Transaktionsgrenzen

Beispielprojekt

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

Talk von Thomas Ploch:

The One Question To Haunt Everyone: What is a DDD Aggregate?

https://www.youtube.com/watch?v=zlFqjD2LKlE

Wenn das Typsystem an die Grenzen kommt

Archunit

EmailVerifier -> Einzige gültige Quelle für VerifiedEmail

Tests

Je besser das Modell, desto weniger Tests

(Tests sind Dokumentation der Domäne)

Reden, Reden und Reden

Mitnehmen!

Sicherstellen der Domänenlogik „nach vorne verschieben“!

Typsystem ausreizen, um Fachlichkeit zu modellieren

Screaming Architecture

Robert C. Martin

Der Code soll die Fachlichkeit herausschreien und nicht die Konzepte von Frameworks

https://blog.cleancoder.com/uncle-bob/2011/09/30/Screaming-Architecture.html

Das habe ich mitgenommen

Von:

Wenig Code -> Wenig Aufwand bei Änderungen

Zu:

Expliziter Code -> Weniger Überraschungen NACH Änderungen

Strategischer Designprozess muss lebendig gehalten werden

Personen lassen sich von Begrifflichkeit triggern und beißen sich an Details fest, während die Grundgedanken in den Hintergrund treten.

Es geht nicht um wenig Code, sondern um explizite Fachlichkeit.

Die Realität ist kompliziert und vermeintlich einfacher Code reduziert das nicht auf magische Art und Weise.

Weitere Links und Mentions

- Eric Evans: https://www.amazon.de/Domain-Driven-Design-Tackling-Complexity-Software/dp/0321125215
- Tom Hombergs: Get Your Hands Dirty on Clean Architecture (Jetzt auch auf Deutsch!)
- Scott Wlaschin: Domain Modeling Made Functional https://www.youtube.com/watch?v=2JB1_e5wZmU
- Dirk und Björn - Unsere Pro-Contra-Diskussionen haben meine Sicht geschärft
- https://alex-ber.medium.com/javas-record-and-sealed-classes-as-categorical-product-and-sum-types-part-iii-03f16544e703
- https://blog.cleancoder.com/uncle-bob/2011/09/30/Screaming-Architecture.html
- Thomas Ploch: What is a DDD Aggregate? https://www.youtube.com/watch?v=zlFqjD2LKlE

Feedback? Contact!

Fabian Kretzer

fabian.kretzer@innoq.com

innoq.social/@fabian

INNOQ

www.innoq.com

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