Die Starre überwinden – Hin zu geschmeidigem Code

Was ist eigentlich das Problem bei der Softwareentwicklung?

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Was ist eigentlich das Problem bei der Softwareentwicklung?

Kommunikationsprobleme:

- Kein ausreichendes Verständnis für das zu lösende Problem
- Keine gemeinsame Sprache von Fachbereich und Entwicklern
- Business-Wissen ist nicht im Code erkennbar
- Es gibt keine Beschreibung der logischen Struktur der Software („Modell“)

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Was ist eigentlich das Problem bei der Softwareentwicklung?

Technische Probleme:

- Häufigstes Entwicklungspattern: Big Ball of Mud
  - ohne erkennbare Architektur
  - auch kleine Änderungen sind langwierig
  - Domäne und Technik sind miteinander vermischt
- Code erklärt nicht, wie er etwas tut

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Wie kann man die Komplexität in den Griff bekommen?

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Wie kann man die Komplexität in den Griff bekommen?

- **Wissenserwerb!**
  - Tiefes Verständnis der Fachlichkeit für alle (auch für die Entwickler)
  - Gemeinsame Sprache für Fachbereich und Entwickler
  - Verständnis für das Warum
  - Herausarbeiten der Kernfunktionalität

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Wie kann man die Komplexität in den Griff bekommen?

- **Wissenserwerb!**
  - Tiefes Verständnis der Fachlichkeit für alle (auch für die Entwickler)
  - Gemeinsame Sprache für Fachbereich und Entwickler
  - Verständnis für das Warum
  - Herausarbeiten der Kernfunktionalität
- **Modularisierung**
  - Zerlegen der Domäne in Teilbereiche
  - Erstellen eines Modells für jeden Teilbereich
  - Einbetten jedes Modells in seinen Kontext

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Wie kann man die Komplexität in den Griff bekommen?

- **Wissenserwerb!**
  - Tiefes Verständnis der Fachlichkeit für alle (auch für die Entwickler)
  - Gemeinsame Sprache für Fachbereich und Entwickler
  - Verständnis für das Warum
  - Herausarbeiten der Kernfunktionalität
- **Modularisierung**
  - Zerlegen der Domäne in Teilbereiche
  - Erstellen eines Modells für jeden Teilbereich
  - Einbetten jedes Modells in seinen Kontext
- **Saubere Implementierung**
  - Fokus auf die Fachlichkeit
  - Bewährte Basis: Solide Objektorientierung / Clean Code
  - Auch andere Patterns sind einsetzbar, sofern sie passen

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Wie kann man die Komplexität in den Griff bekommen?

- **Wissenserwerb!**
  - Tiefes Verständnis der Fachlichkeit für alle (auch für die Entwickler)
  - Gemeinsame Sprache für Fachbereich und Entwickler
  - Verständnis für das Warum
  - Herausarbeiten der Kernfunktionalität
- **Modularisierung**
  - Zerlegen der Domäne in Teilbereiche
  - Erstellen eines Modells für jeden Teilbereich
  - Einbetten jedes Modells in seinen Kontext
- **Saubere Implementierung**
  - Fokus auf die Fachlichkeit
  - Bewährte Basis: Solide Objektorientierung / Clean Code
  - Auch andere Patterns sind einsetzbar, sofern sie passen

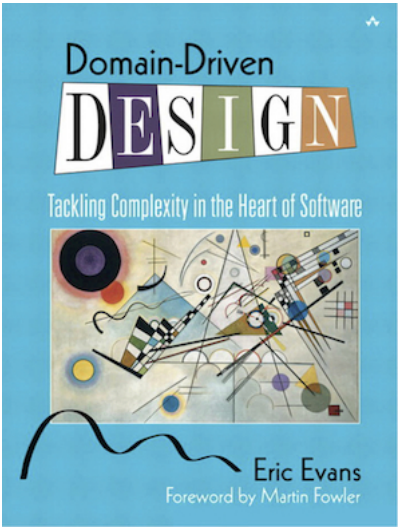
$\Rightarrow$ **Domain-Driven Design**

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Supple Design

Supple Design - Einführung

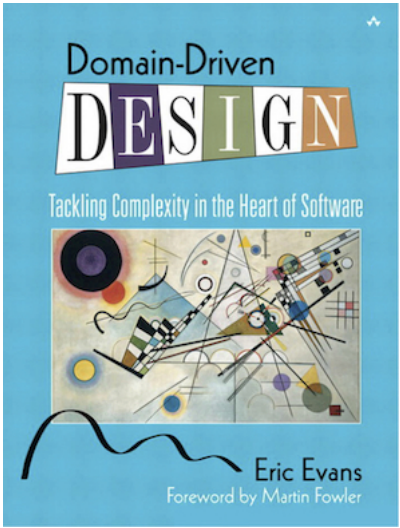
- Kapitel 10 in Eric Evans' Buch: Supple Design



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Supple Design - Einführung

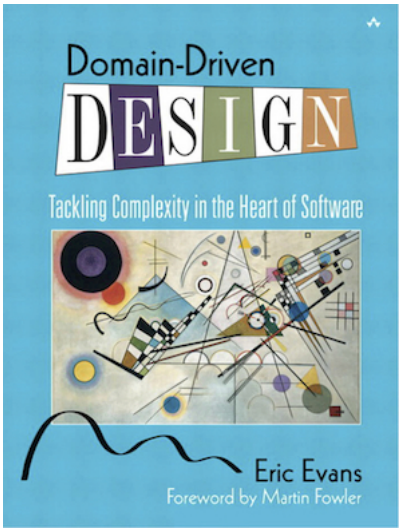
- Kapitel 10 in Eric Evans' Buch: Supple Design
  - Intention-Revealing Interfaces



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Supple Design - Einführung

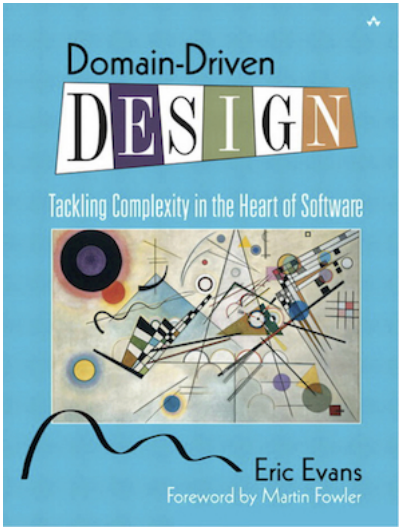
- Kapitel 10 in Eric Evans' Buch: Supple Design
  - Intention-Revealing Interfaces
  - Side-Effect-Free Functions



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Supple Design - Einführung

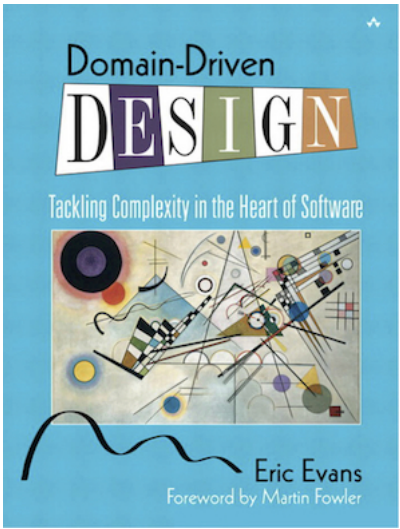
- Kapitel 10 in Eric Evans' Buch: Supple Design
  - Intention-Revealing Interfaces
  - Side-Effect-Free Functions
  - Assertions



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Supple Design - Einführung

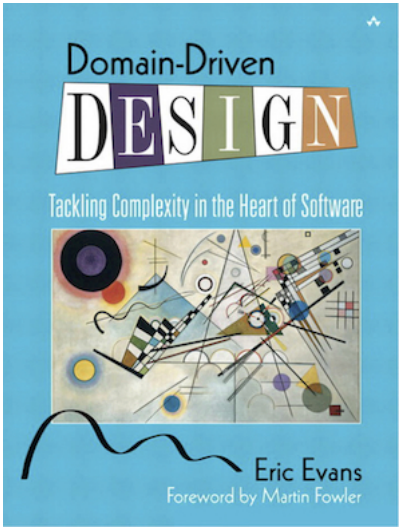
- Kapitel 10 in Eric Evans' Buch: Supple Design
  - Intention-Revealing Interfaces
  - Side-Effect-Free Functions
  - Assertions
  - Conceptual Contours



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Supple Design - Einführung

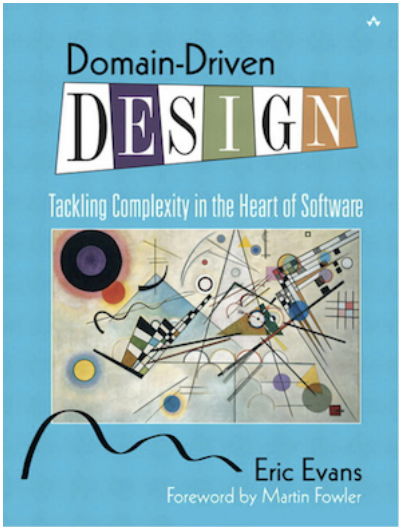
- Kapitel 10 in Eric Evans' Buch: Supple Design
  - Intention-Revealing Interfaces
  - Side-Effect-Free Functions
  - Assertions
  - Conceptual Contours
  - Standalone Classes



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

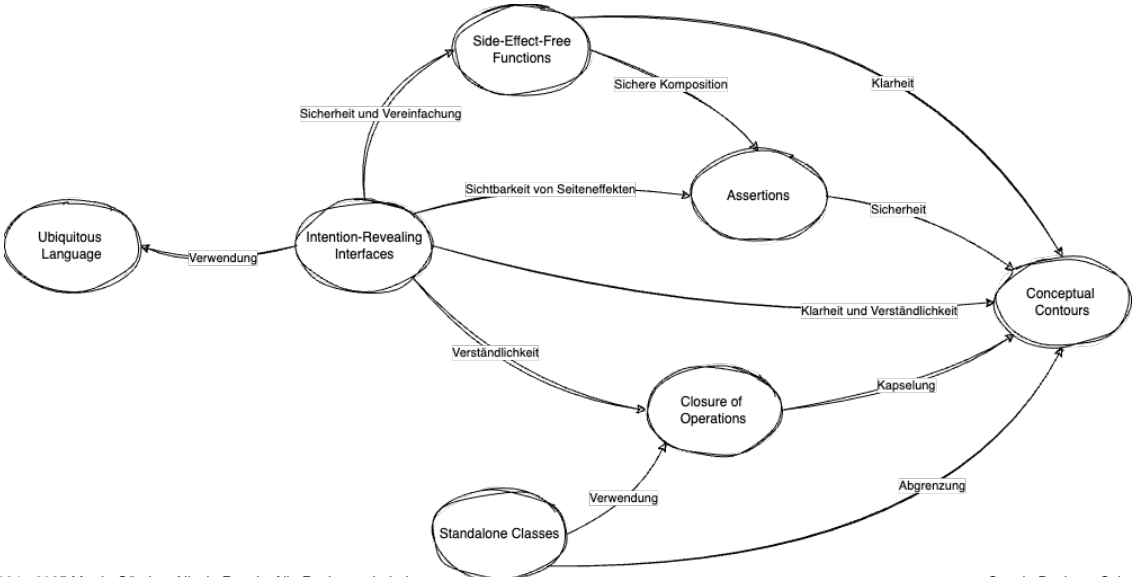
Supple Design - Einführung

- Kapitel 10 in Eric Evans' Buch: Supple Design
  - Intention-Revealing Interfaces
  - Side-Effect-Free Functions
  - Assertions
  - Conceptual Contours
  - Standalone Classes
  - Closure of Operations



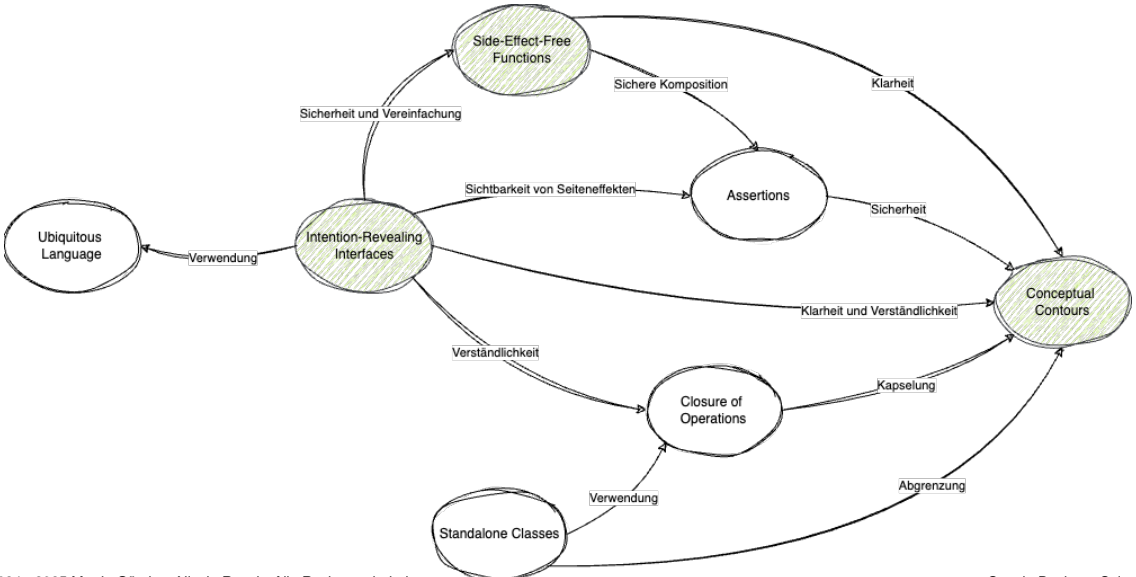
© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Übersicht



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Wir schauen heute an



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Unsere Domäne

Der Gast kann. . .

- . . . einen Raum anfragen
- . . . einen Raum buchen
- . . . bezahlen

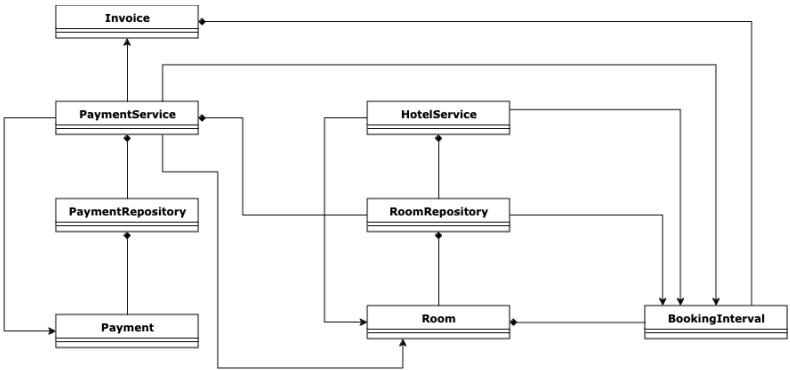
© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Das Hotel kann. . .

- . . . den Gast einchecken
- . . . den Gast auschecken
- . . . die vom Gast gezahlten Beträge prüfen
- . . . eine Rechnung erstellen

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Initiale Codestruktur



- Anämisches Modell
- Viel Logik in den Services

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Warum sehen viele Java-Enterprise-Anwendungen so aus?

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Historische Wurzeln

- **Geschäftslogik in der Datenbank:**
  - Stored Procedures oder Java-Code direkt in der DB

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Historische Wurzeln

- **Geschäftslogik in der Datenbank:**
  - Stored Procedures oder Java-Code direkt in der DB
- **EJB-forcierte Schichtenarchitektur:**
  - Message Driven Beans, Session Beans, Entity Beans

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Historische Wurzeln

- **Geschäftslogik in der Datenbank:**
  - Stored Procedures oder Java-Code direkt in der DB
- **EJB-forcierte Schichtenarchitektur:**
  - Message Driven Beans, Session Beans, Entity Beans
- **SOA & stateless Services:**
  - Logik in zentralisierten Services statt in Domänenobjekten

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Langfristige Auswirkungen

- **Anämische Domänenmodelle:**
  - (Mutable) Entitäten enthalten nur Daten, keine Logik

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Langfristige Auswirkungen

- **Anämische Domänenmodelle:**
  - (Mutable) Entitäten enthalten nur Daten, keine Logik
- **Technische Struktur dominiert:**
  - Controller $\rightarrow$ Service $\rightarrow$ Repository als Standard

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Langfristige Auswirkungen

- **Anämische Domänenmodelle:**
  - (Mutable) Entitäten enthalten nur Daten, keine Logik
- **Technische Struktur dominiert:**
  - Controller $\rightarrow$ Service $\rightarrow$ Repository als Standard
- **CRUD-zentriertes Denken:**
  - Datenbankstrukturen bestimmen den Code

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Warum ist das auch heute noch so?

- **Gewohnheit & “Best” Practices:**
  - aus der EJB- und SOA-Ära

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Warum ist das auch heute noch so?

- **Gewohnheit & “Best” Practices:**
  - aus der EJB- und SOA-Ära
- **Schnelle Entwicklung vs. langfristige Wartbarkeit:**
  - CRUD ist einfach, aber lenkt Fokus weg vom Verhalten

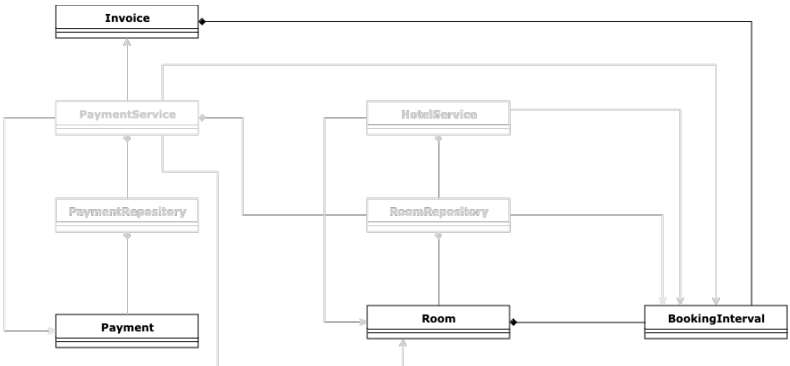
© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Warum ist das auch heute noch so?

- **Gewohnheit & “Best” Practices:**
  - aus der EJB- und SOA-Ära
- **Schnelle Entwicklung vs. langfristige Wartbarkeit:**
  - CRUD ist einfach, aber lenkt Fokus weg vom Verhalten
- **Fehlender Fokus auf Objektorientierung:**
  - Technische Strukturen dominieren, während die Fachlogik oft auf der Strecke bleibt

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Domänenmodellierung



- Domäne sehr schwach ausgeprägt
- Nicht wirklich erkennbar, worum es eigentlich geht

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Intention-Revealing Interfaces

Intention Revealing Interfaces - Motivation

- Wir nutzen Modularisierung, um Komplexität in den Griff zu bekommen.
- Module bieten Schnittstellen für deren Benutzung an.
- *„Wenn ein Entwickler die Implementierung einer Komponente berücksichtigen muss, um sie verwenden zu können, geht der Wert der Kapselung verloren.“* – Eric Evans
- An der Schnittstelle einer Komponente sollte erkennbar sein, **was** sie beabsichtigt zu tun - um das **wie** sollte sich der Benutzer keine Gedanken machen müssen.
- Das gilt sogar, wenn die gleiche Person Entwickler und Benutzer der Komponente ist.

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Beispiel HotelService - Vorher

```
HotelService:

   Double requestRoom(LocalDate, LocalDate)

   void bookRoom(LocalDate, LocalDate, String)

   List<String> checkIn(String, LocalDate)

   void checkOut(String, String, LocalDate)
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Beispiel PaymentService - Vorher

```
PaymentService:

   void payAmount(String, double)

   double remainingCredit(String)

   Invoice produceInvoice(String, LocalDate, List<String>)
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Intention Revealing Interfaces - Steps

- Die Schnittstelle besteht aus mehr als nur dem Methodennamen.
- Nutze domänenspezifische Typen
  - als Rückgabetyp und
  - als Typen der Parameter,
  - sodass mit dem Methodennamen zusammen
  - ein INTENTION-REVEALING INTERFACE entsteht.
- Die verwendeten Namen sollten mit der UBIQUITOUS LANGUAGE in Einklang stehen.
- Schreibe Tests unter Verwendung der Schnittstelle vor der Implementierung, um selbst die Perspektive eines Benutzers einzunehmen.

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Martin Fowler: Refactoring

Erwähnt schon 1999 den Code Smell

Primitive Obsession

Whenever a variable that is just a simple `string`, or an `int` simulates being a more abstract concept, which could be an object, we encounter a *Primitive Obsession* code smell. This lack of abstraction quickly becomes a problem whenever there is the need for any additional logic, and also because these variables easily spread wide and far in the codebase. This alleged verbal abstraction is just a "supposed" object, but it should have a real object instead.

Causation

Possibly a missing class to represent the concept in the first place. Mäntylä gives an example of representing money as primitive rather than creating a separate class [1], and so does Fowler, who states that many programmers are reluctant to create their own fundamental types. [2]. Higher-level abstraction knowledge is needed to clarify or simplify the code. [3

Problems

Hidden Intention

Primitive Obsession

Last Revision — April 19, 2022
2 Min Read

Also Known As

Obstruction
Bloaters

Occurrence
Data

Expanse
Between Classes

Related Smells
- **Type Embedded in Name** (co-exist)
- **Obscured Intent** (causes)
- **Lazy Element** (antagonistic)

History
Martin Fowler in book (1999):
"Refactoring: Improving the Design of Existing Code"

https://luzkan.github.io/smells/primitive-obsession

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Object Calisthenics

Schau, die machen das auch!

1. Only One Level Of Indentation Per Method
2. Don’t Use The ELSE Keyword
3. Wrap All Primitives And Strings
4. First Class Collections
5. One Dot Per Line
6. Don’t Abbreviate
7. Keep All Entities Small
8. No Classes With More Than Two Instance Variables
9. No Getters/Setters/Properties

- `https://williamdurand.fr/2013/06/03/object-calisthenics/`

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Beispiel HotelService - Nachher

Vorher:

```
HotelService:

   Double requestRoom(LocalDate, LocalDate)

   void bookRoom(LocalDate, LocalDate, String)

   List<String> checkIn(String, LocalDate)

   void checkOut(String, String, LocalDate)
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Beispiel HotelService - Nachher

Vorher:

```
HotelService:

   Double requestRoom(LocalDate, LocalDate)

   void bookRoom(LocalDate, LocalDate, String)

   List<String> checkIn(String, LocalDate)

   void checkOut(String, String, LocalDate)
```

Nachher:

```
HotelService:

   Amount requestRoom(ArrivalDate, DepartureDate)

   void bookRoom(ArrivalDate, DepartureDate, GuestName)

   List<RoomNumber> checkIn(GuestName, ArrivalDate)

   void checkOut(GuestName, RoomNumber, DepartureDate)
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Beispiel PaymentService - Nachher

Vorher:

```
PaymentService:

   void payAmount(String, double)

   double remainingCredit(String)

   Invoice produceInvoice(String, LocalDate, List<String>)
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Beispiel PaymentService - Nachher

Vorher:

```
PaymentService:

   void payAmount(String, double)

   double remainingCredit(String)

   Invoice produceInvoice(String, LocalDate, List<String>)
```

Nachher:

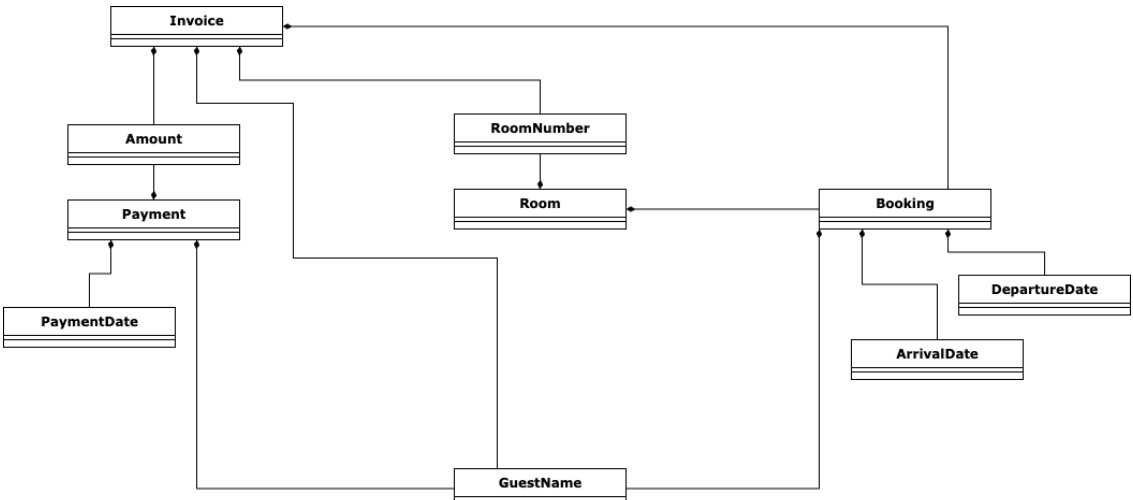
```
PaymentService:

   void payAmount(GuestName, Amount)

   Amount remainingCredit(GuestName)

   Invoice produceInvoice(GuestName, DepartureDate, List<RoomNumber>)
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Side-Effect Free Functions

Command-Query-Separation (CQS)

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Command-Query-Separation (CQS)

„Das CQS-Prinzip besagt, dass eine Methode entweder als Abfrage (*query*) oder als Kommando (*command, modifier oder mutator*) implementiert werden soll. Eine Abfrage muss hierbei Daten zurückliefern und darf keine Seiteneffekte auf dem beobachtbaren Zustand des Systems aufweisen, während ein Kommando beobachtbare Nebeneffekte aufweist und keine Daten zurückliefert.“ – Wikipedia

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Command-Query-Separation (CQS)

„Das CQS-Prinzip besagt, dass eine Methode entweder als Abfrage (*query*) oder als Kommando (*command, modifier oder mutator*) implementiert werden soll. Eine Abfrage muss hierbei Daten zurückliefern und darf keine Seiteneffekte auf dem beobachtbaren Zustand des Systems aufweisen, während ein Kommando beobachtbare Nebeneffekte aufweist und keine Daten zurückliefert.“ – Wikipedia

“Perform all queries and calculations in methods that cause no observable side effects.”

– Bertrand Meyer: Object-oriented Software Construction, 1988

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Command-Query-Separation (CQS)

„Das CQS-Prinzip besagt, dass eine Methode entweder als Abfrage (*query*) oder als Kommando (*command, modifier oder mutator*) implementiert werden soll. Eine Abfrage muss hierbei Daten zurückliefern und darf keine Seiteneffekte auf dem beobachtbaren Zustand des Systems aufweisen, während ein Kommando beobachtbare Nebeneffekte aufweist und keine Daten zurückliefert.“ – Wikipedia

“Perform all queries and calculations in methods that cause no observable side effects.”

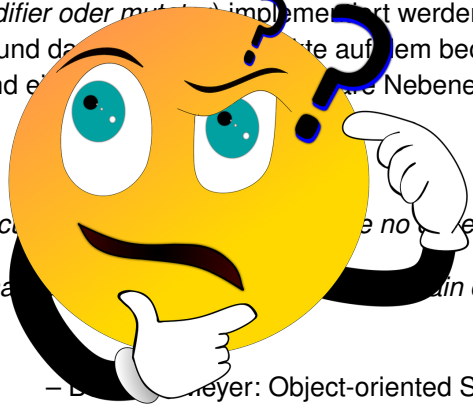
“Ensure that the methods that cause changes do not return domain data and are kept as simple as possible.”

– Bertrand Meyer: Object-oriented Software Construction, 1988

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Command-Query-Separation (CQS)

„Das CQS-Prinzip besagt, dass eine Methode entweder als Abfrage (*query*) oder als Kommando (*command, modifier oder mutator*) implementiert werden soll. Eine Abfrage muss hierbei Daten zurückliefern und darf keine Änderung auf dem beobachtbaren Zustand des Systems aufweisen, während ein Kommando keine Nebeneffekte aufweist und keine Daten zurückliefert.“ – Wikipedia



“Perform all queries and calculations [illegible] no observable side effects.”

“Ensure that the methods that [illegible] contain data and are kept as simple as possible.”

– [illegible]eyer: Object-oriented Software Construction, 1988

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Seiteneffekt

*Der Begriff **Seiteneffekt** bezeichnet jeden beobachtbaren Effekt, abgesehen vom Lesen der Argumentwerte und vom Zurückgeben eines Wertes.* – Wikipedia

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Seiteneffekt

*Der Begriff **Seiteneffekt** bezeichnet jeden beobachtbaren Effekt, abgesehen vom Lesen der Argumentwerte und vom Zurückgeben eines Wertes.* – Wikipedia

Beispiele:

- Modifizieren
  - nichtlokaler Variablen
  - Instanzvariablen (= Zustand der eigenen Klasse)
  - als Referenz übergebener Methodenargumente
- Werfen von Exceptions
- Ausführen von I/O
- Aufrufen von Funktionen mit Seiteneffekten

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Vorteile von Seiteneffektfreiheit

- Seiteneffektfreiheit ist notwendig für Referentielle Transparenz, d. h.
  - ein Ausdruck liefert mit denselben Argumenten stets dasselbe Ergebnis
- Verständnis für den Kontext und mögliche vergangene Ereignisse ist nicht nötig, d. h.
  - Seiteneffektfreiheit erleichtert das Nachdenken über ein Programm
  - Seiteneffektfreiheit erleichtert das Debuggen
  - Seiteneffektfreiheit erleichtert das Testen
  - Seiteneffektfreiheit erleichtert die formale Verifikation eines Programms

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Seiteneffekt: Exception

```
class HotelService {

    ...
    public RoomNumber bookRoom(ArrivalDate arrivalDate, DepartureDate departureDate, GuestName guestName) {

        Booking booking = new Booking(arrivalDate, departureDate, guestName);
        for (Room room : rooms.getRooms().values()) {
            if (room.roomIsFree(arrivalDate, departureDate)) {
                room.getBookings().add(booking);
                rooms.save(room);
                return room.roomNumber();
            }
        }
        throw new IllegalStateException("No rooms available on the given date(s)");
    }
    ...
}
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

It is up to you to decide what to do when an exception condition is encountered. Many object-oriented programming languages define mechanisms for programmers to declare exceptions and error conditions, signal their occurrence, and to write and associate exception-handling code that executes when signaled.

– Rebecca Wirfs-Brock, Alan McKean: Object Design, 2002.

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

It is up to you to decide what to do when an exception condition is encountered. Many object-oriented programming languages define mechanisms for programmers to declare exceptions and error conditions, signal their occurrence, and to write and associate exception-handling code that executes when signaled.

Alternatively, you could design an object to detect an exception condition, and, instead of raising an exception, it could return a result indicating that an exception occurred.

– Rebecca Wirfs-Brock, Alan McKean: Object Design, 2002.

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Keine Exception: Either

```
class HotelService {

    ...
    public Either<Error, RoomNumber> bookRoom(ArrivalDate arrivalDate, DepartureDate departureDate,
         GuestName guestName) {

        Booking booking = new Booking(arrivalDate, departureDate, guestName);
        for (Room room : rooms.getRooms().values()) {
            if (room.roomIsFree(arrivalDate, departureDate)) {
                room.getBookings().add(booking);
                rooms.save(room);
                return Either.ofResult(room.roomNumber());
            }
        }
        return Either.ofError(new Error("No rooms available on the given date(s)"));
    }

    ...
}
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Keine Exception: Either

```
class HotelService {

    ...
    public Either<Error, RoomNumber> bookRoom(ArrivalDate arrivalDate, DepartureDate departureDate,
         GuestName guestName) {

        Booking booking = new Booking(arrivalDate, departureDate, guestName);
        for (Room room : rooms.getRooms().values()) {
            if (room.roomIsFree(arrivalDate, departureDate)) {
                room.getBookings().add(booking);
                rooms.save(room);
                return Either.ofResult(room.roomNumber());
            }
        }
        return Either.ofError(new Error("No rooms available on the given date(s)"));
    }

    ...
}
```

Aber weiterhin Seiteneffekt durch Laden und Persistieren des Raums!

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Kein Laden / Persistieren

```
class HotelService {

    ...
    public Either<Error, Room> bookRoom(ArrivalDate arrivalDate, DepartureDate departureDate, GuestName
         guestName, Rooms allRooms) {

        Booking booking = new Booking(arrivalDate, departureDate, guestName);
        for (Room room : allRooms.rooms()) {
            if (room.roomIsFree(arrivalDate, departureDate)) {
                room.getBookings().add(booking);
                return Either.ofResult(room);
            }
        }
        return Either.ofError(new Error("No rooms available on the given date(s)"));
    }

    ...
}
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Kein Laden / Persistieren

```
class HotelService {

    ...
    public Either<Error, Room> bookRoom(ArrivalDate arrivalDate, DepartureDate departureDate, GuestName
         guestName, Rooms allRooms) {

        Booking booking = new Booking(arrivalDate, departureDate, guestName);
        for (Room room : allRooms.rooms()) {
            if (room.roomIsFree(arrivalDate, departureDate)) {
                room.getBookings().add(booking);
                return Either.ofResult(room);
            }
        }
        return Either.ofError(new Error("No rooms available on the given date(s)"));
    }

    ...
}
```

Aber weiterhin Seiteneffekt durch Verändern der Raumbuchungen!

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Endlich seiteneffektfrei

```
public Either<Error, Room> bookRoom(ArrivalDate arrivalDate, DepartureDate departureDate, GuestName
     guestName, Rooms allRooms) {

    Booking booking = new Booking(arrivalDate, departureDate, guestName);
    for (Room room : allRooms.rooms()) {
        if (room.roomIsFree(arrivalDate, departureDate)) {
            return Either.ofResult(room.add(booking));
        }
    }
    return Either.ofError(new Error("No rooms available on the given date(s)"));
}
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Endlich seiteneffektfrei

```
public Either<Error, Room> bookRoom(ArrivalDate arrivalDate, DepartureDate departureDate, GuestName
     guestName, Rooms allRooms) {

    Booking booking = new Booking(arrivalDate, departureDate, guestName);
    for (Room room : allRooms.rooms()) {
        if (room.roomIsFree(arrivalDate, departureDate)) {
            return Either.ofResult(room.add(booking));
        }
    }
    return Either.ofError(new Error("No rooms available on the given date(s)"));
}
```

Aber leider wird es einem nicht leicht gemacht...

```
public Room add(Booking booking) {
    List<Booking> newBookings = new ArrayList<>(bookings);
    newBookings.add(booking);
    return new Room(roomNumber, newBookings);
}
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Fokus zurück auf die Fachlichkeit

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Fokus zurück auf die Fachlichkeit

Wieder zurück zu den Intention-Revealing Interfaces:

```
public Either<Error, Room> bookRoom(ArrivalDate arrivalDate, DepartureDate departureDate, GuestName
     guestName, Rooms allRooms) {

    Booking booking = new Booking(arrivalDate, departureDate, guestName);
    ...
}
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Fokus zurück auf die Fachlichkeit

Wieder zurück zu den Intenti… Interfaces:

```
public Either<Error, Room>…rivalDate, DepartureDate departureDate, GuestName
     guestName, Rooms allR…

    Booking booking = new …                    de…rtureDate, guestName);
    ...
}
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Fokus zurück auf die Fachlichkeit

Wieder zurück zu den Intention-Revealing Interfaces:

```
public Either<Error, Room> bookRoom(ArrivalDate arrivalDate, DepartureDate departureDate, GuestName
     guestName, Rooms allRooms) {

    Booking booking = new Booking(arrivalDate, departureDate, guestName);
    ...
}
```

Wäre es nicht besser...

```
public Either<Error, Room> bookRoom(Booking booking, Rooms allRooms) {
    ...
}
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Fokus zurück auf die Fachlichkeit

Wieder zurück zu den Intention-Revealing Interfaces:

```
public Either<Error, Room> bookRoom(ArrivalDate arrivalDate, DepartureDate departureDate, GuestName
     guestName, Rooms allRooms) {

    Booking booking = new Booking(arrivalDate, departureDate, guestName);
    ...
}
```

Wäre es nicht besser...

```
public Either<Error, Room> bookRoom(Booking booking, Rooms allRooms) {
    ...
}
```

Aber eigentlich ist das ja noch kein Booking:

```
public Either<Error, Room> bookRoom(BookingRequest bookingRequest, Rooms allRooms) {
    ...
}
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Das Endergebnis

```
class HotelService {

    ...
    public Either<Error, Room> bookRoom(BookingRequest bookingRequest, Rooms allRooms) {

        for (Room room : allRooms.rooms()) {
            if (room.roomIsFree(bookingRequest.arrivalDate(), bookingRequest.departureDate())) {
                return Either.ofResult(room.add(bookingRequest));
            }
        }
        return Either.ofError(new Error("No rooms available on the given date(s)"));
    }

    ...
}
```

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Conceptual Contours

Conceptual Contours ist ein bisschen wie...

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Von der Realität zum Modell

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Von der Realität zum Modell

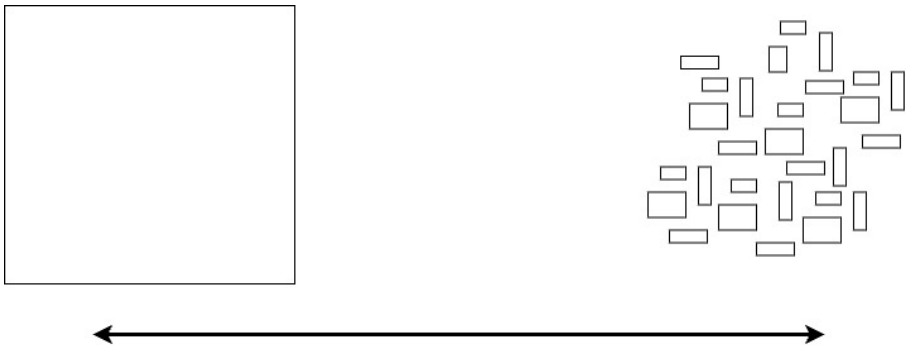
© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Von der Realität zum Modell

- Was ist Teil dieses Modells?
- Was ist explizit nicht Teil des Modells?
- Welche Granularität hat das Modell?

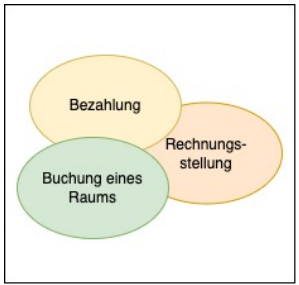
© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

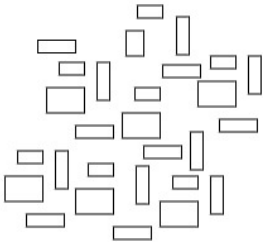
Granularität von Modellen



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

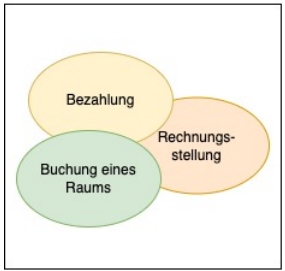
Granularität von Modellen

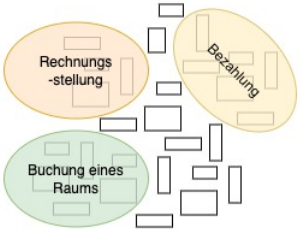




© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

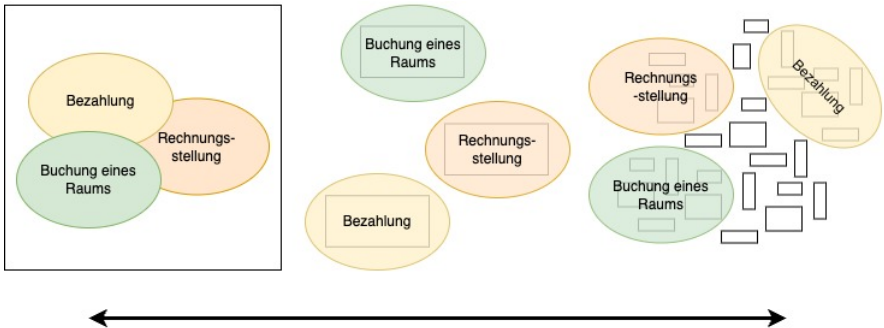
Granularität von Modellen





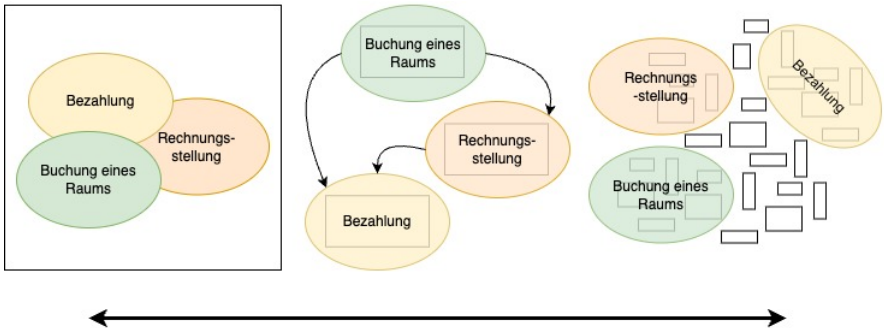
© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Granularität von Modellen



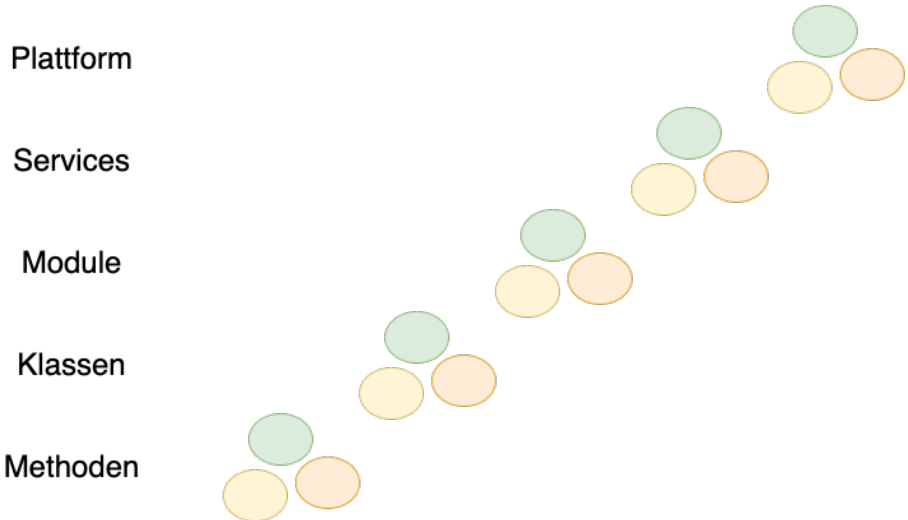
© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Granularität von Modellen



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Granularität von Modellen - auf allen Ebenen



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Zu grob modelliert?

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Zu fein modelliert?

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Conceptual Contours - Was nun?

- Es gibt kein Patentrezept, aber...

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Conceptual Contours - Was nun?

- Es gibt kein Patentrezept, aber...
  - ...jede Fachdomäne enthält logisch konsistente Aspekte.

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Conceptual Contours - Was nun?

- Es gibt kein Patentrezept, aber...
  - ...jede Fachdomäne enthält logisch konsistente Aspekte.
  - ...wenn wir diese grundlegenden, konsistenten Bereiche finden und modellieren, ist die Wahrscheinlichkeit größer, dass sie auch mit anderen Bereichen übereinstimmen, die wir vielleicht erst später entdecken.

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Conceptual Contours - Was nun?

- Es gibt kein Patentrezept, aber...
  - ...jede Fachdomäne enthält logisch konsistente Aspekte.
  - ...wenn wir diese grundlegenden, konsistenten Bereiche finden und modellieren, ist die Wahrscheinlichkeit größer, dass sie auch mit anderen Bereichen übereinstimmen, die wir vielleicht erst später entdecken.
  - ...wenn wir feststellen, dass eine neue Anforderung mit dem aktuellen Modell schwer umzusetzen ist, sollten wir ein tiefgreifendes Refactoring machen, sodass wir zu einem Modell kommen, mit dem die Änderung einfach umzusetzen ist (und hoffentlich ebenso folgende Änderungen).

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Conceptual Contours - Was nun?

- Es gibt kein Patentrezept, aber...
  - ...jede Fachdomäne enthält logisch konsistente Aspekte.
  - ...wenn wir diese grundlegenden, konsistenten Bereiche finden und modellieren, ist die Wahrscheinlichkeit größer, dass sie auch mit anderen Bereichen übereinstimmen, die wir vielleicht erst später entdecken.
  - ...wenn wir feststellen, dass eine neue Anforderung mit dem aktuellen Modell schwer umzusetzen ist, sollten wir ein tiefgreifendes Refactoring machen, sodass wir zu einem Modell kommen, mit dem die Änderung einfach umzusetzen ist (und hoffentlich ebenso folgende Änderungen).
- Wiederholtes Refactoring führt zu Suppleness

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Conceptual Contours - Was nun?

- Es gibt kein Patentrezept, aber...
  - ...jede Fachdomäne enthält logisch konsistente Aspekte.
  - ...wenn wir diese grundlegenden, konsistenten Bereiche finden und modellieren, ist die Wahrscheinlichkeit größer, dass sie auch mit anderen Bereichen übereinstimmen, die wir vielleicht erst später entdecken.
  - ...wenn wir feststellen, dass eine neue Anforderung mit dem aktuellen Modell schwer umzusetzen ist, sollten wir ein tiefgreifendes Refactoring machen, sodass wir zu einem Modell kommen, mit dem die Änderung einfach umzusetzen ist (und hoffentlich ebenso folgende Änderungen).
- Wiederholtes Refactoring führt zu Suppleness
  - Die Conceptual Contours des Modells nähern sich den Konturen der realen Fachdomäne an

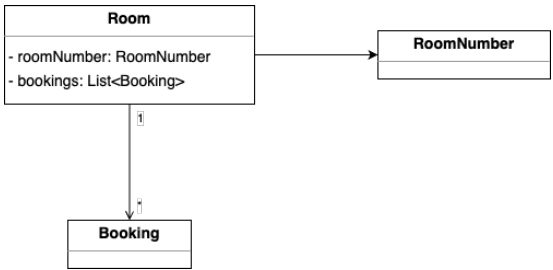
© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Conceptual Contours - Was nun?

- Es gibt kein Patentrezept, aber...
  - ...jede Fachdomäne enthält logisch konsistente Aspekte.
  - ...wenn wir diese grundlegenden, konsistenten Bereiche finden und modellieren, ist die Wahrscheinlichkeit größer, dass sie auch mit anderen Bereichen übereinstimmen, die wir vielleicht erst später entdecken.
  - ...wenn wir feststellen, dass eine neue Anforderung mit dem aktuellen Modell schwer umzusetzen ist, sollten wir ein tiefgreifendes Refactoring machen, sodass wir zu einem Modell kommen, mit dem die Änderung einfach umzusetzen ist (und hoffentlich ebenso folgende Änderungen).
- Wiederholtes Refactoring führt zu Suppleness
  - Die Conceptual Contours des Modells nähern sich den Konturen der realen Fachdomäne an
- *„...zügle dein technisches Denken und frage dich, ob die Änderungen zu deiner Sicht auf die Fachdomäne passen.“* – Eric Evans

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Modell 1



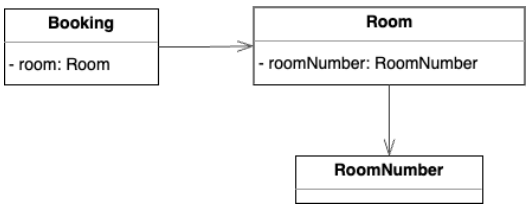
© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Realitätscheck: Modell 1

- Sollte sich der Raum ändern, wenn er gebucht wird?

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Modell 2



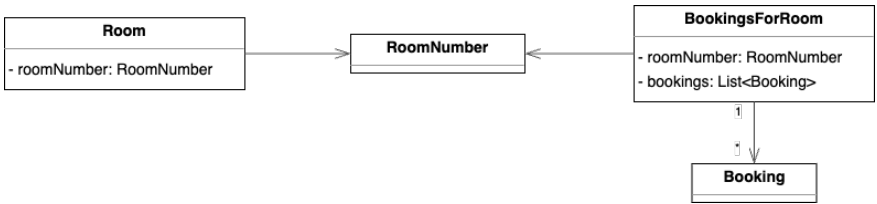
© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Realitätscheck: Modell 2

- Muss die Buchung den kompletten Raum mit **all** seinen Eigenschaften kennen?

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Modell 3



- Wie gut ist das Modell?
- BookingsForRoom - ist das wirklich UbiquitousLanguage?

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Realitätscheck: Modell 3

- Muss man bei einer Buchung wirklich all die verschiedenen Ordner durchgehen, die jeweils mit einer Raumnummer beschriftet sind?

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Vergleich der Modelle

- Komplexität der Umsetzung von UseCases

UseCase	**Modell 1** **R $\rightarrow$ B**	**Modell 2** **B $\rightarrow$ R**	**Modell 3** **BfRs $\rightarrow$ R,B**
Raum anfragen	gering	hoch	gering
Raum buchen	mittel	hoch	mittel
Gast einchecken	hoch	gering	mittel
Gast auschecken	hoch	gering	hoch

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Vergleich der Modelle

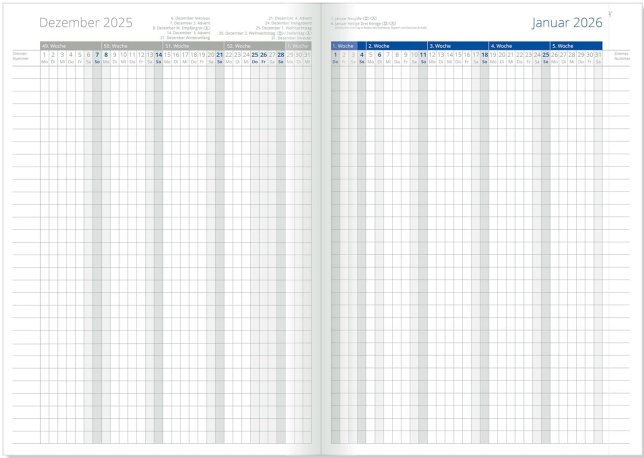
- Komplexität der Umsetzung von neuen UseCases

Neue UseCases	**Modell 1** $\mathbf{R \rightarrow B}$	**Modell 2** $\mathbf{B \rightarrow R}$	**Modell 3** $\mathbf{BfRs \rightarrow R,B}$
Stammzimmer	???	???	???
Betriebsferien	???	???	???
Raumausstattung	???	???	???
weiteres Hotel	???	???	???
...	...	...	...

- In allen drei Modellen sind die neuen UseCases schwer umsetzbar, da...
  - die Belegung eines Raumes implizit durch die Existenz einer Buchung ausgedrückt wird und
  - die Belegungzeiträume eines Raumes nur implizit in den Buchungen bekannt sind.
- Hier entsteht Legacy Software ;)

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

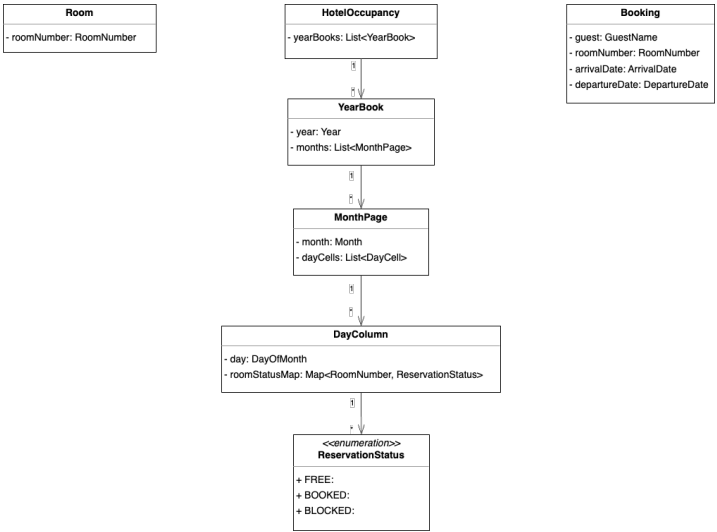
Eine mögliche Lösung - ein Belegungsplan



- https://www.amazon.de/Zimmerbelegungskalender-Zimmerbelegungsplan-Belegungskalender-Ferienvermietungen-Gästevermietungen/dp/B08P5WHRRR

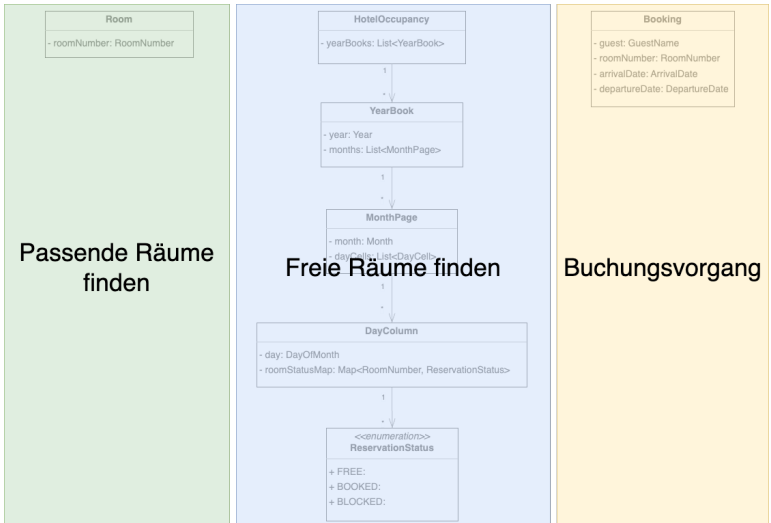
© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Eine mögliche Lösung - ein Belegungsplan



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Eine mögliche Lösung - ein Belegungsplan



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Belegungsplan

- Vorteile
  - Verknüpfung von Räumen, Buchungen und Aufenthaltszeitraum ist explizit im Belegungsplan modelliert.
  - Diese explizite Verknüpfung ermöglicht mehr Zustände als nur FREI oder GEBUCHT, z.B. blockierte Räume für Reparaturen oder Betriebsferien.
  - Durch Hinzufügen eines weiteren Belegungsplans lässt sich einfach ein zweites Hotel hinzufügen.
  - Das Suchen von Zimmern im Belegungsplan lässt sich einfach auf eine Teilliste von Raumnummern einschränken.
  - Das Filtern der Räume, beispielsweise das Lieblingszimmer eines Stammgastes oder bzgl. der Ausstattung, findet komplett im Raummodell statt.

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Belegungsplan

- Vorteile
  - Verknüpfung von Räumen, Buchungen und Aufenthaltszeitraum ist explizit im Belegungsplan modelliert.
  - Diese explizite Verknüpfung ermöglicht mehr Zustände als nur FREI oder GEBUCHT, z.B. blockierte Räume für Reparaturen oder Betriebsferien.
  - Durch Hinzufügen eines weiteren Belegungsplans lässt sich einfach ein zweites Hotel hinzufügen.
  - Das Suchen von Zimmern im Belegungsplan lässt sich einfach auf eine Teilliste von Raumnummern einschränken.
  - Das Filtern der Räume, beispielsweise das Lieblingszimmer eines Stammgastes oder bzgl. der Ausstattung, findet komplett im Raummodell statt.
- Nachteile
  - Initial vergleichsweise hohe Komplexität zum Aufsetzen des „Spielfelds“.
  - Änderungen an Buchungen müssen mit dem Belegungsplan synchronisiert werden.

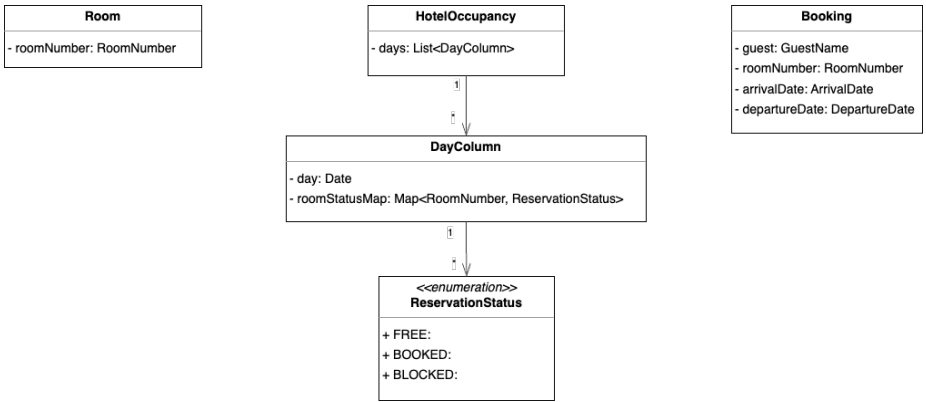
© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Belegungsplan

- Vorteile
  - Verknüpfung von Räumen, Buchungen und Aufenthaltszeitraum ist explizit im Belegungsplan modelliert.
  - Diese explizite Verknüpfung ermöglicht mehr Zustände als nur FREI oder GEBUCHT, z.B. blockierte Räume für Reparaturen oder Betriebsferien.
  - Durch Hinzufügen eines weiteren Belegungsplans lässt sich einfach ein zweites Hotel hinzufügen.
  - Das Suchen von Zimmern im Belegungsplan lässt sich einfach auf eine Teilliste von Raumnummern einschränken.
  - Das Filtern der Räume, beispielsweise das Lieblingszimmer eines Stammgastes oder bzgl. der Ausstattung, findet komplett im Raummodell statt.
- Nachteile
  - Initial vergleichsweise hohe Komplexität zum Aufsetzen des „Spielfelds“.
  - Änderungen an Buchungen müssen mit dem Belegungsplan synchronisiert werden.
- Ist es das beste Modell?

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Warum noch Jahresbücher und Monatsblätter?



© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Fazit

评论集
诗歌集
梦上床头
卢晓圣 著
香港文學報社出版公司
美国管理革命通鉴
中华养生粥谱
神奇的食醋
GAMBLING AND COMMERCIAL GAMING
Eadington & Cornelius
CHINESE BUSINESS ETIQUETTE
SCOTT D. SELIGMAN
现行证券期货法规汇编
法律出版社
毛泽东选集
水浒传
Exhibition IPR Protection Guiding Cases
RANDOM HOUSE DICTIONARY
JOURNAL OF PHYSIOLOGY
VOL. 181
1965
THE SOUTH BEACH DIET
PETROLEUM FLUIDS
活出人生的滋味 女人40岁以后的品味生活
中外大学校长论坛文集
女人要狠 地位才稳
人体造型与人体形象
世界上下五千年
中国P2P借贷服务行业白皮书2014
爱丽丝漫游奇境
彼得大帝
博学天下
一千零一夜
安徒生童话
YOGA 瑜伽全程轻松练
中国证券业竞争战略研究
小美女28天美容养颜经
习仲勋与群众路线
四季养花一本通
论语
弟子规
增广贤文
THE BIRDS OF PREY

“this kind of design is **difficult**. You can’t just look at an enormous system and say, ‘Let’s make this supple.’ You have to **choose targets**.”

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

“this kind of design is **difficult**. You can’t just look at an enormous system and say, ‘Let’s make this supple.’ You have to **choose targets**.”

“You just can’t tackle the whole design at once. **Pick away at it**. Some aspects of the system will suggest approaches to you, and they can be **factored out** and worked over. You may see a part of the model that can be viewed as specialized math; **separate that**. Your application enforces complex rules restricting state changes; **pull this out** into a **separate model** or simple framework that lets you declare the rules. With each such step, not only is the new module **clean**, but also the part left behind is **smaller** and **clearer**.”

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

“this kind of design is **difficult**. You can’t just look at an enormous system and say, ‘Let’s make this supple.’ You have to **choose targets**.”

“You just can’t tackle the whole design at once. **Pick away at it**. Some aspects of the system will suggest approaches to you, and they can be **factored out** and worked over. You may see a part of the model that can be viewed as specialized math; **separate that**. Your application enforces complex rules restricting state changes; **pull this out** into a **separate model** or simple framework that lets you declare the rules. With each such step, not only is the new module **clean**, but also the part left behind is **smaller** and **clearer**.”

“Supple design has a **profound effect** on the ability of software to **cope with change and complexity**. [. . .] it often hinges on quite **detailed** modeling and design decisions. The impact can **go beyond** a specific modeling and design problem.”

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Vielen Dank!

info@nicole-rauch.de
https://www.nicole-rauch.de

Software Craftsmanship
Domain-Driven Design
Specification by Example
React & Redux · TypeScript
Funktionale Programmierung

Martin Günther

mg@martinguenther-consulting.de
https://martinguenther-consulting.de/

Softwarearchitektur JVM
Domain-Driven Design
Camunda & Axon
Systemstabilisierung und -modernisierung
Spring Boot 3 Migrationen

Credits

Einführung: Matt Sharpe (unchanged)

`https://www.flickr.com/photos/mdsharpe/5075947453`

Supple Design: Leo

`https://pixabay.com/photos/leather-leather-gloves-gloves-4799259/`

Domain: Valeriia Bugaiova

`https://unsplash.com/photos/a-large-swimming-pool-surrounded-by-palm-trees-_pPHgeHz1uk`

IntentionRevealing: Buddhist, Ema - Good luck

`https://pixabay.com/photos/buddhist-ema-good-luck-intentions-5921037/`

SideEffectFree: guvo59 - Hand, tablets, capsules

`https://pixabay.com/photos/hand-tablets-capsules-health-3632914/`

Assertions: marsjo - Happens, Handshake, People

`https://pixabay.com/photos/happens-handshake-people-contract-4044426/`

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.

Credits

ConceptualContours: Peggy_Marco: Ongoing, Investigations, Crime scene

https://pixabay.com/photos/ongoing-investigations-crime-scene-1661910/

StandaloneClasses: Max Böhme - lonely tree

https://unsplash.com/photos/leafless-tree-on-snow-covered-ground-eajvTAkDovM

ClosureOfOperations: Sandy Millar - Road Closed sign on street

https://unsplash.com/photos/a-road-closed-sign-sitting-on-the-side-of-a-road-4MoIqT9BEOQ

Conclusion: Eric Prouzet - A Colorful Display of Vintage Books at a Local Used Bookstore in the Afternoon

https://unsplash.com/photos/assorted-title-book-lot-1aYp7IFkHRM

Smiley: Conmongt - Cartoon, Smiley, Fragen.

https://pixabay.com/de/illustrations/cartoon-smiley-fragen-r%C3%A4tseln-3082809/

© 2024 - 2025 Martin Günther, Nicole Rauch. Alle Rechte vorbehalten.