Data-Oriented Programming

Mike Sperber

@sperbsen@discuss.systems

@ctive group

Mike Sperber

- Oberchef
  Active Group
- Promotion
  Uni Tübingen 2001
- programmiere seit 1983
  erstes Buch 1986
- funktionale Programmierung
- Ausbildung

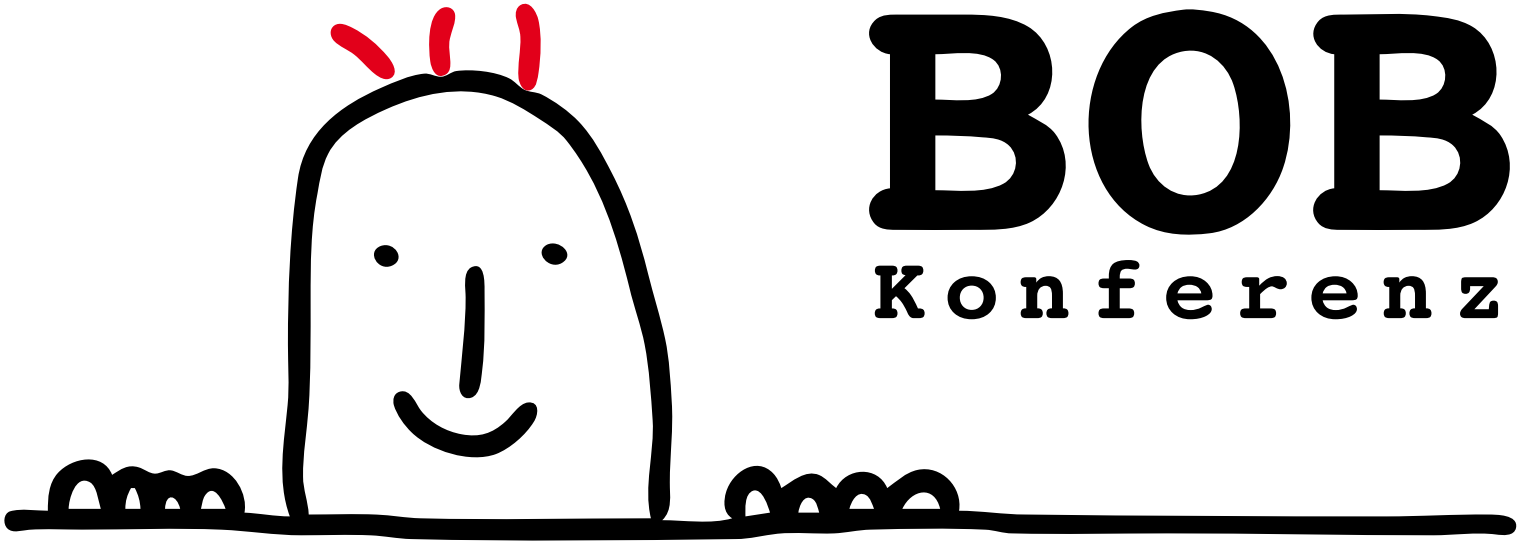
@active group

- Projektentwicklung
- verschiedene Branchen
- funktionale Programmierung
- Schulungen
- iSAQB Foundation
- iSAQB Advanced Level:
  Funktionale Softwarearchitektur
  Domänenspezifische Sprachen
  Formale Methoden

www.active-group.de

funktionale-programmierung.de

@active group



BOB Konferenz

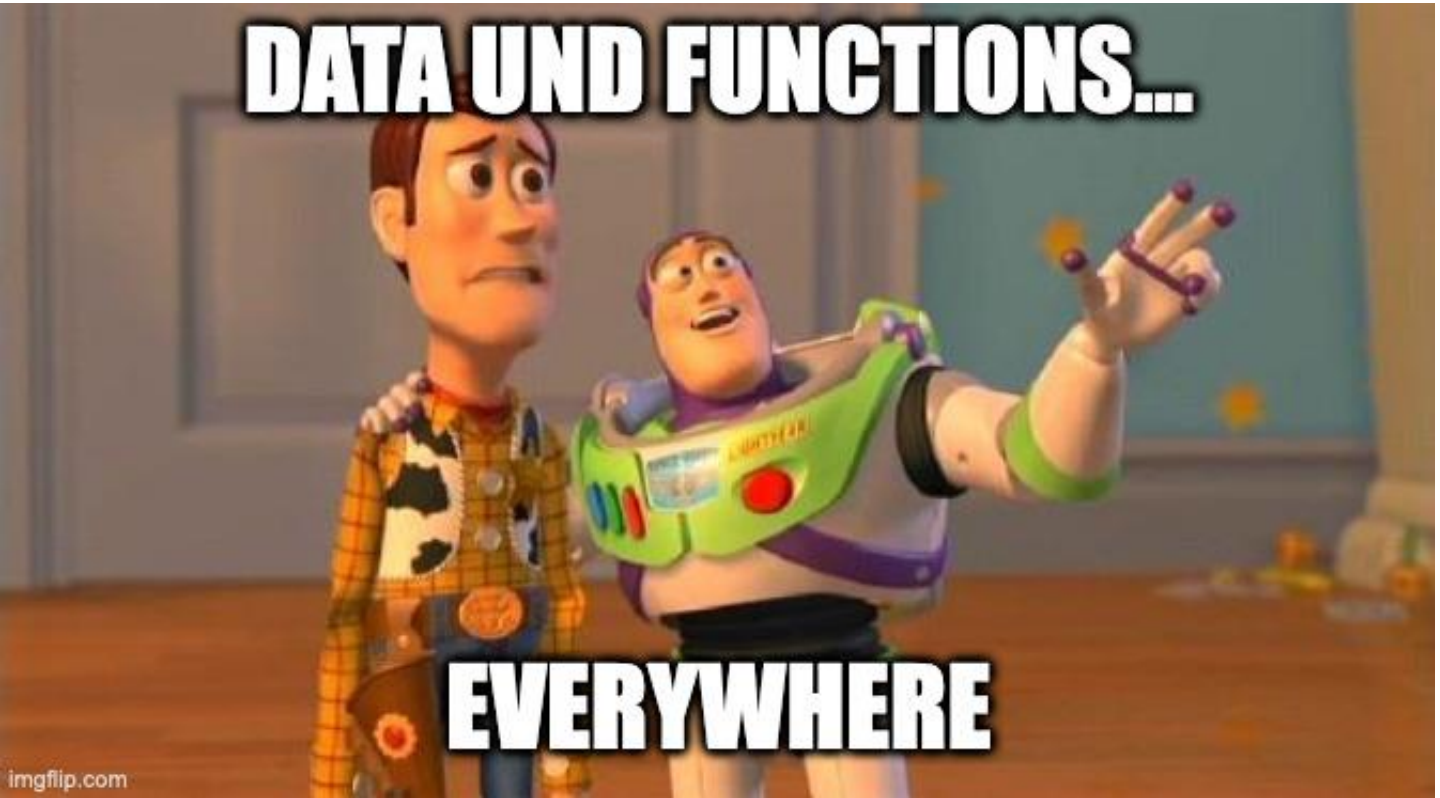
bobkonf.de

Februar? März?, 2027, Berlin

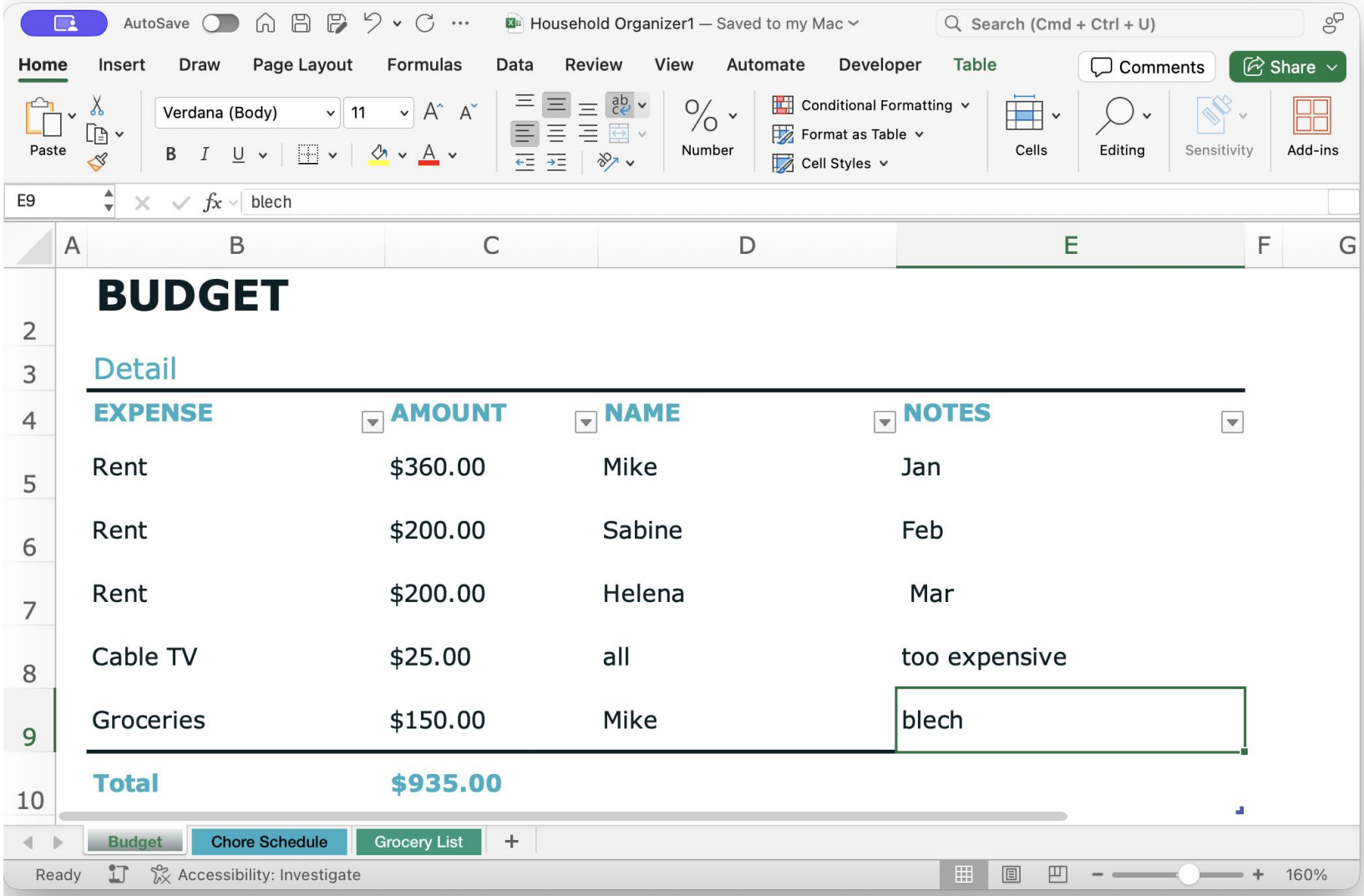
Funktionale Softwarearchitektur

- alles sind (unveränderliche) Daten
- funktionale Anforderungen zuerst

- Bottom-Up
- Abstraktion FTW
- Mathe FTW



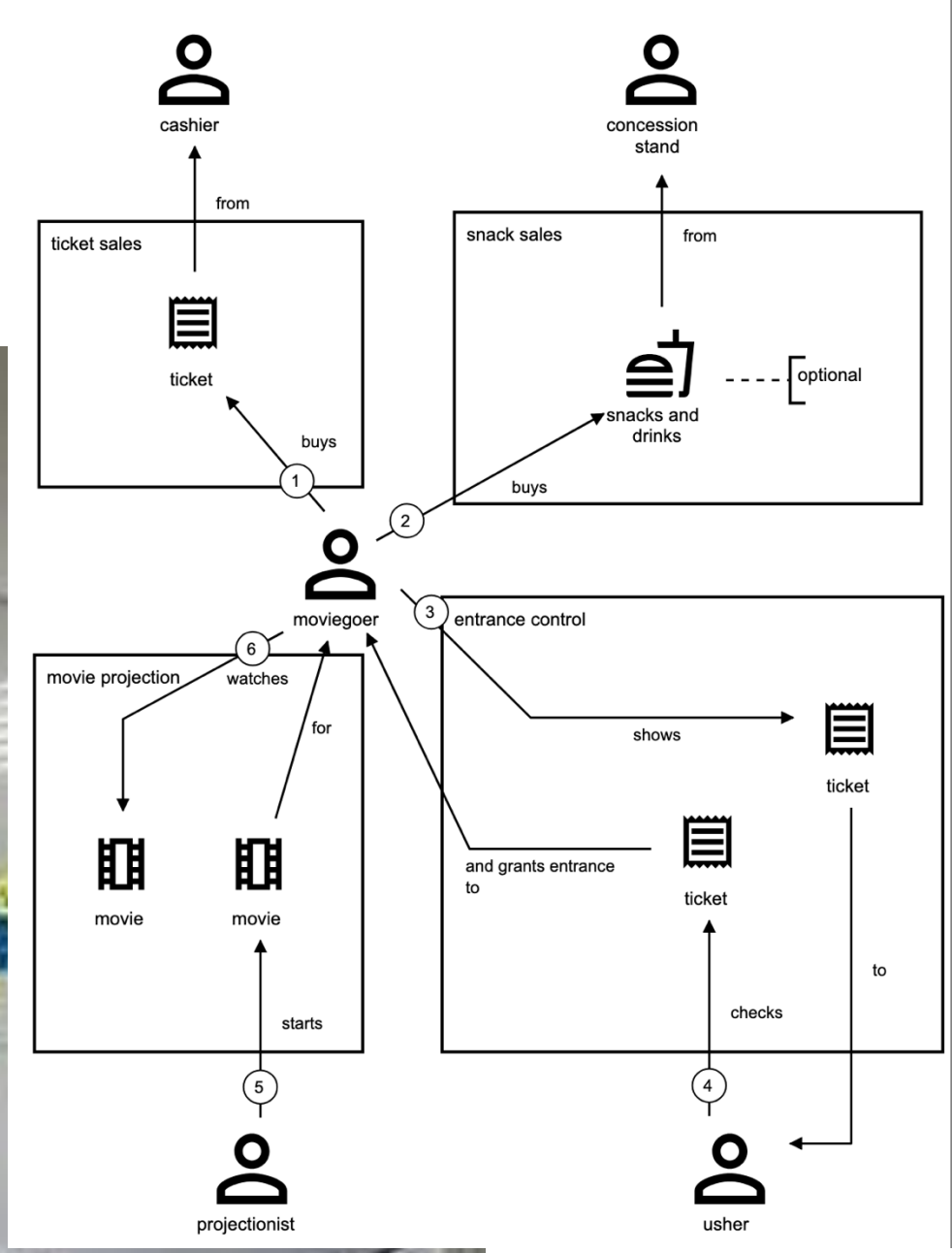
Daten in der freien Wildbahn



,

)

“Collaborative Modeling”



iSAQB Foundation 2025

LZ 01-07: Bedeutung von Daten und Datenmodellen (R2)

Softwarearchitekt:innen verstehen die Bedeutung von Daten und Datenmodellen (unabhängig von ihrer physischen Repräsentation) für die Architektur. Sie

- können Datenmodelle identifizieren, die maßgeblichen Einfluss auf die Architektur haben.
- können solche Datenmodelle systematisch entwerfen.
- verstehen den Unterschied zwischen Produkten und Summen in der Datenmodellierung.

Datenmodellierung

Ein Gesundheitsinformationssystem stellt u.a. Informationen über die Medikation eines Patienten bereit. Für unser Beispiel besteht eine Medikation aus dem Namen eines Medikaments und seiner Dosierung. Es gibt zwei verschiedene Arten von Dosierungen, abhängig davon, ob das Medikament oral über Tabletten oder intravenös über eine Infusion verabreicht wird.

Die Dosierung für Tabletten gibt die Anzahl der Tabletten an, die morgens, mittags und abends eingenommen werden sollen. Beispiel: *1-0-2* bedeutet, dass morgens eine Tablette, mittags keine Tablette und abends zwei Tabletten genommen werden sollen.

Die Dosierung für Infusionen gibt an, wie schnell die Infusion fließt (in Millilitern pro Minute) und wie lange die Infusion laufen soll (in Stunden). Beispiel: *1,5ml/min für 2h* bedeutet, dass die Infusion 2 Stunden lang mit einer Geschwindigkeit von 1,5ml pro Minute laufen soll.

https://funktionale-programmierung.de/2024/11/25/sums-products.html

Produkte und Summen

- *Medikation* besteht aus dem Namen der Droge und der Dosierung
- *Dosierung* ist entweder eine Dosierung für Tabletten oder eine für Infusionen
- *Dosierung für Tabletten* besteht aus jeweils einer Menge von Tabletten, die am Morgen, Mittag und Abend eingenommen werden sollen.
- *Dosierung für Infusionen* besteht aus der Geschwindigkeit (ml/min) und der Dauer (h).

Produkte und Summen

- *Medikation* **besteht aus** dem Namen der Droge **und** der Dosierung
- *Dosierung* ist **entweder** eine Dosierung für Tabletten **oder** eine für Infusionen
- *Dosierung für Tabletten* **besteht aus** jeweils einer Menge von Tabletten, die am Morgen, Mittag **und** Abend eingenommen werden sollen.
- *Dosierung für Infusionen* **besteht aus** der Geschwindigkeit (ml/min) **und** der Dauer (h).

Haskell

```
data Medication
  = Medication { drugName :: String,
                 dosage :: Dosage }
data Dosage
  = TabletDosage { morning :: Int,
                   midday :: Int,
                   evening :: Int }
  | InfusionDosage { speed :: Double,
                     duration :: Int }
```

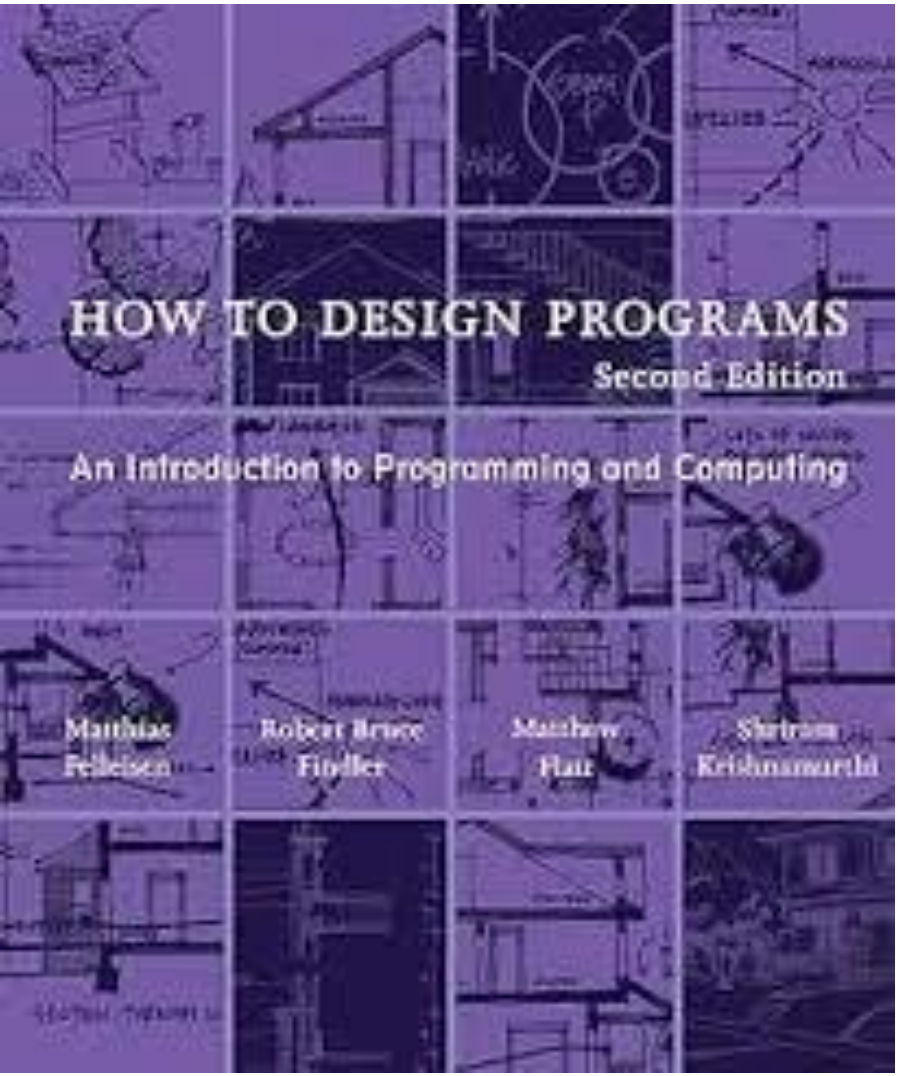
Java

```
public record Medication(String drugName,
                         Dosage dosage) {}

public sealed interface Dosage {
  record Tablet(int morning,
                int midday,
                int evening)
    implements Dosage {}

  record Infusion(double speed,
                  int duration)
    implements Dosage {}
}
```

Sources



htdp.org

deinprogramm.de

Duschprodukte

```
public record Soap(double pH) {}

public enum HairType {
  OILY, DRY, NORMAL, DANDRUFF
    }

public record Shampoo(HairType hairType) {}

public record ShowerGel(Soap soap,
                        Shampoo shampoo) {}
```

Summen FTW

```
public sealed interface ShowerProduct {
    record Soap(double pH)
     implements ShowerProduct {}
    record Shampoo(HairType hairType)
     implements ShowerProduct {}
    record Mixture(ShowerProduct product1,
                   ShowerProduct product2)
     implements ShowerProduct {}
}
```

Pattern-Matching

```
static double
    soapProportion(ShowerProduct product) {
      return switch (product) {
       case Soap(double pH) -> 1.0;
       case Shampoo(HairType hairType) -> 0.0;
       case Mixture(ShowerProduct product1,
                    ShowerProduct product2) ->
        (soapProportion(product1) +
         soapProportion(product2))
          /2.0;
      };
    }
```

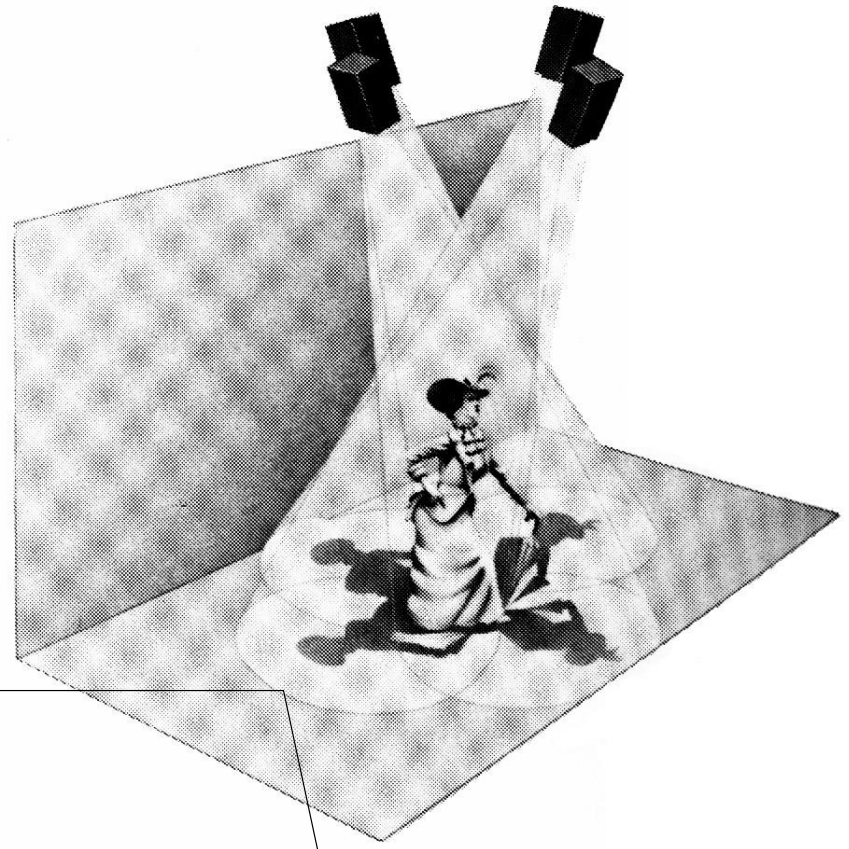
Abstrahieren

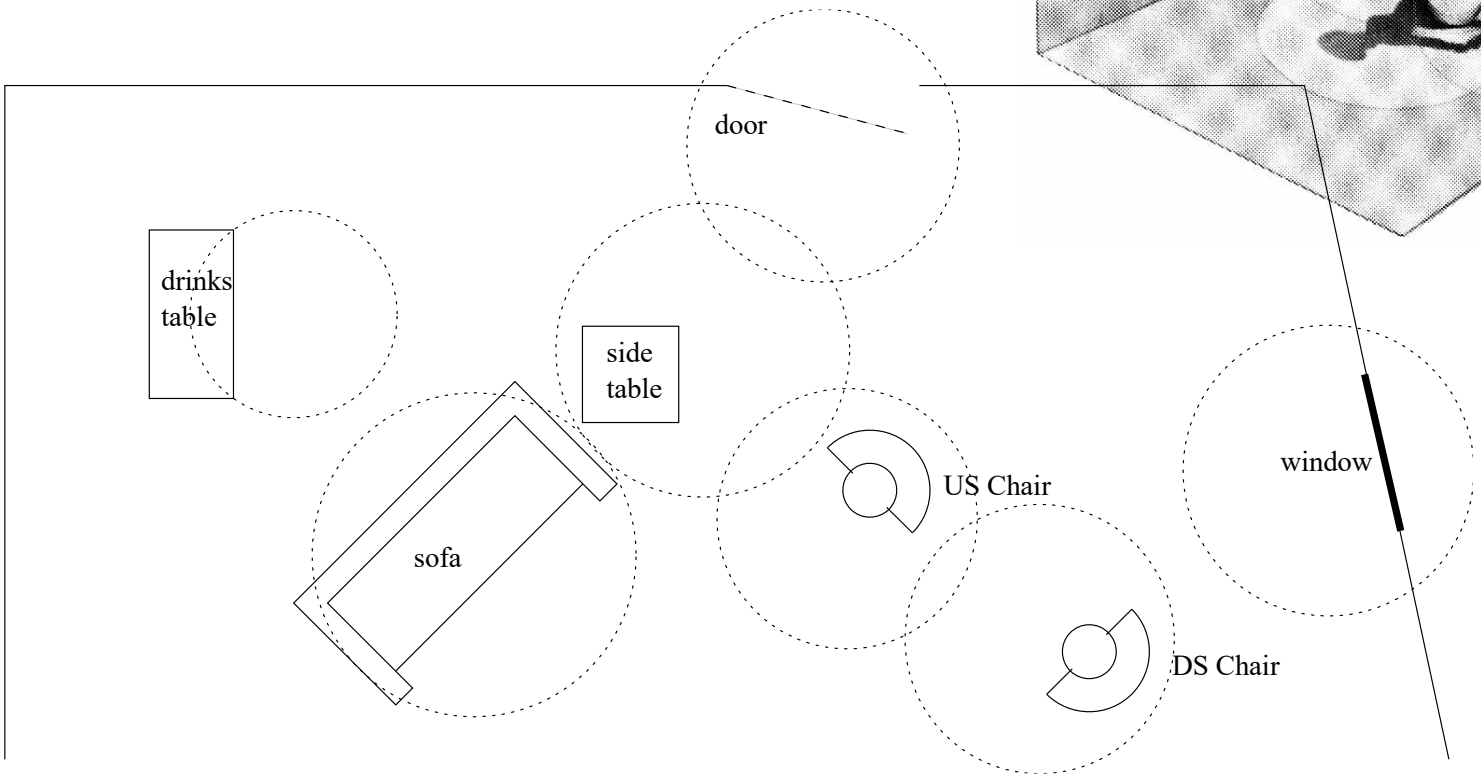
oder

nicht abstrahieren

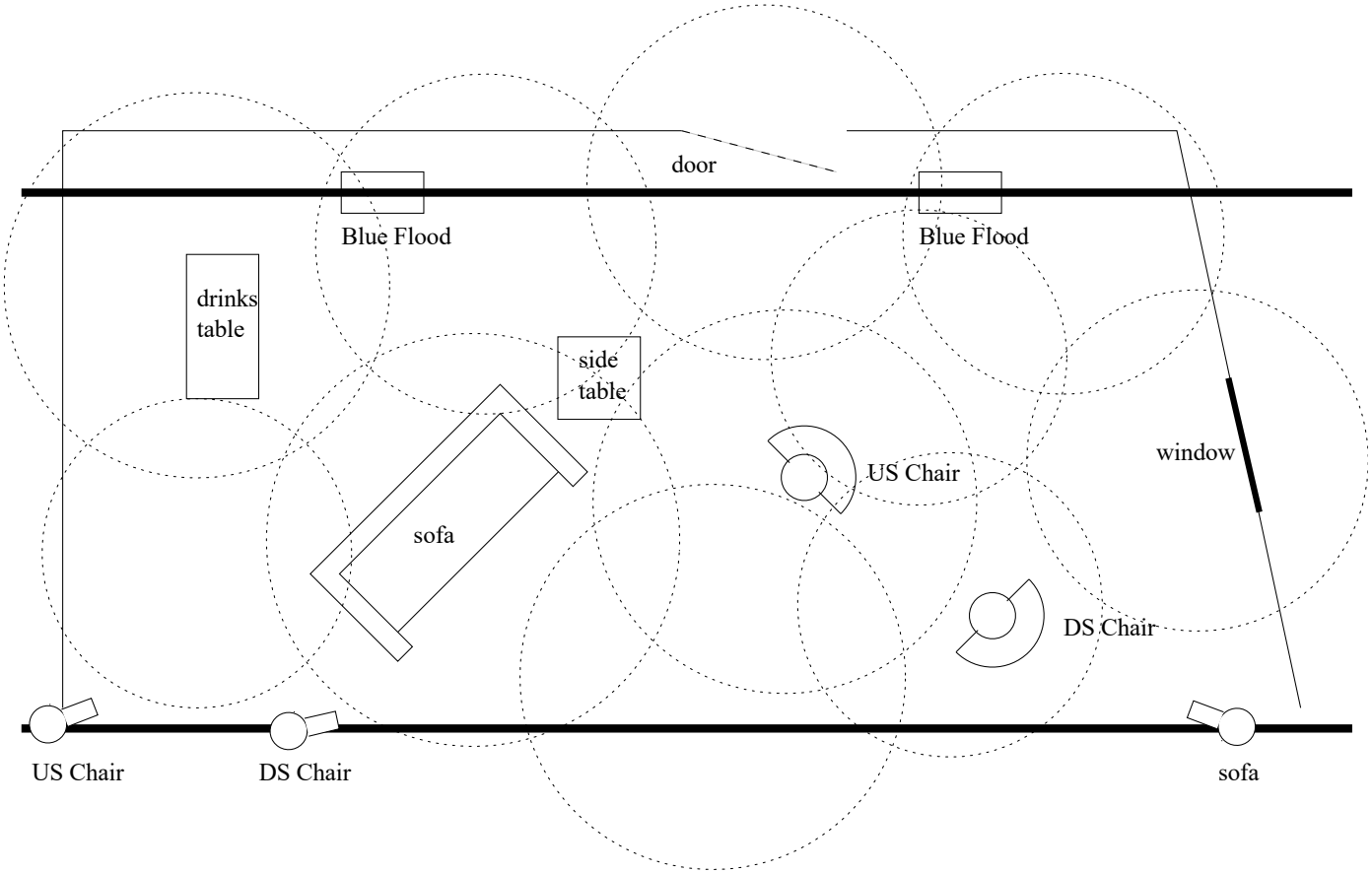
Bühnenbeleuchtung

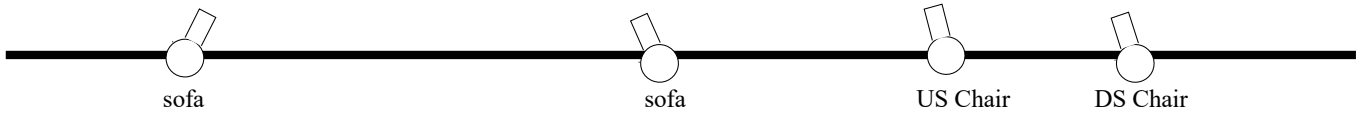
Lichtdesign



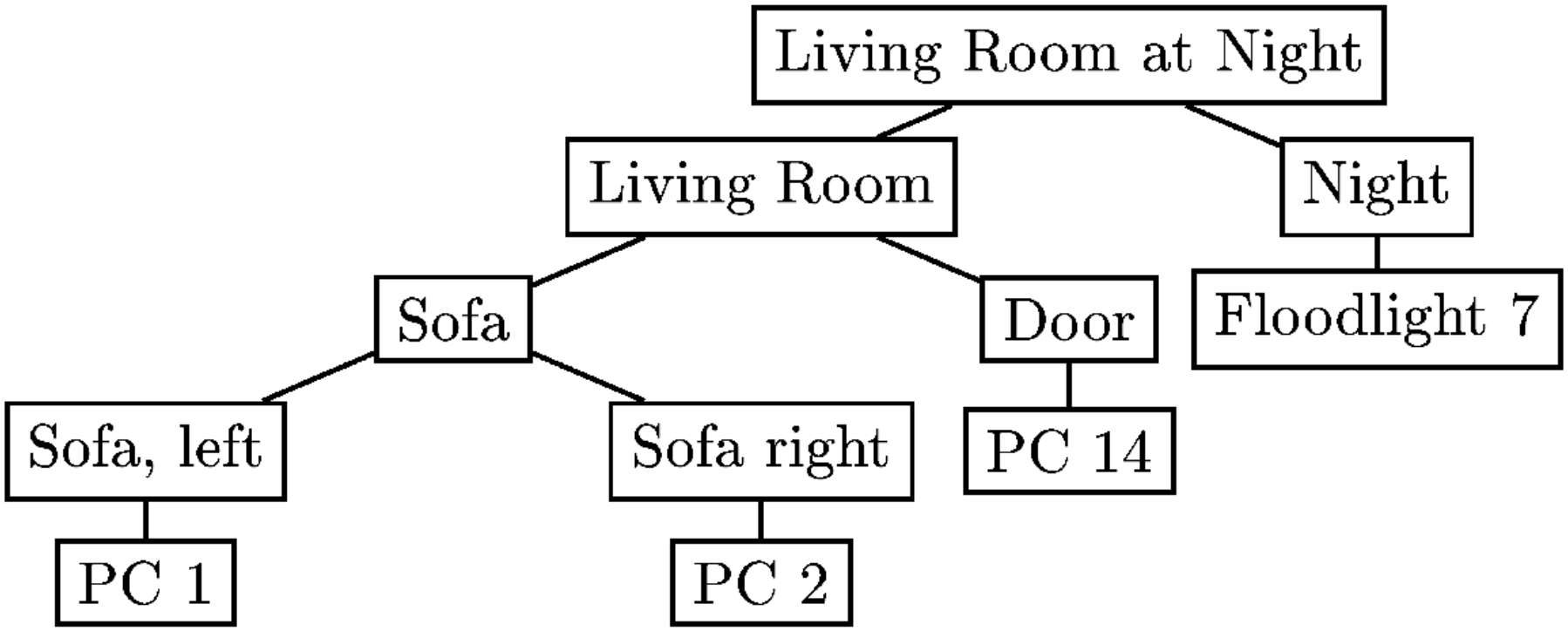


Spezifikation vs. Implementierung





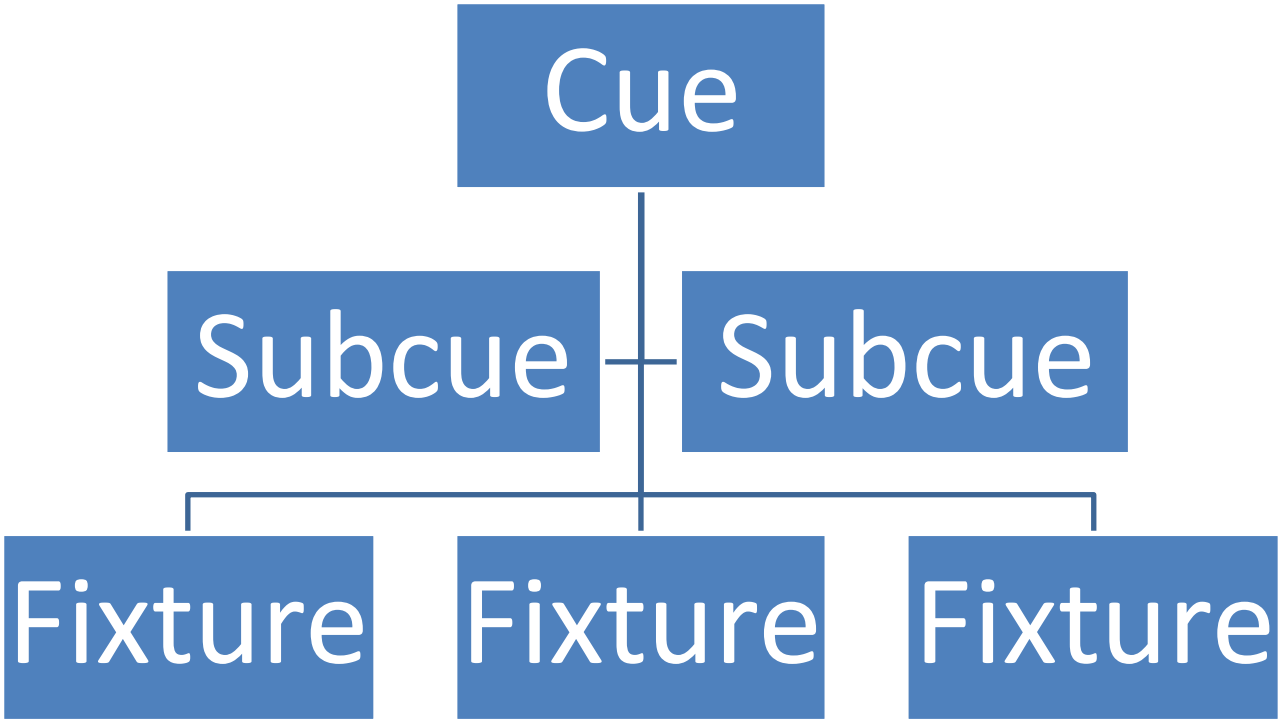
Hierarchische Struktur



Beleuchtungskontrollsysteme

- “Submaster”
- “relative Cues”
- “Presets”
- …

Stratifiziertes Datenmodell



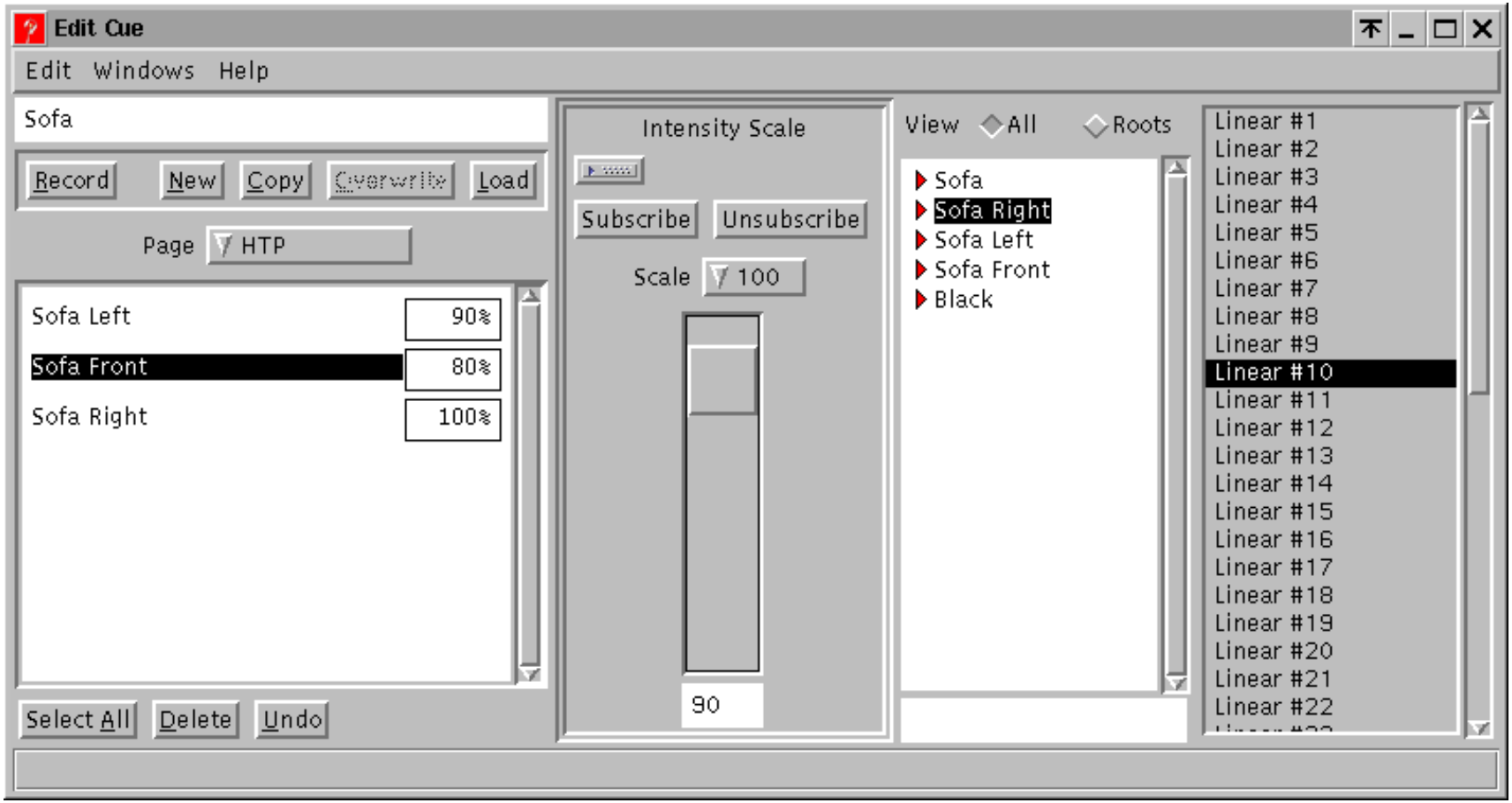
Kombinatormodell

Ein **Cue** ist ...

- ... ein **einfache** Lampe
- ... ein **skalierter** Cue
- ... eine **Kombination** von Cues

```
public sealed interface Cue {
  record Single(Fixture fixture)
    implements Cue {}
  record Scaled(double factor, Cue cue)
    implements Cue {}
  record Combination(List<Cue> cues)
    implements Cue {}
}
```

Lula



Vorteile

- **viel** einfacher zu benutzen
- **viel** schneller zu benutzen
- braucht keine große Konsole
- folgt Änderungen
- funktioniert auf Tournee
- langlebig

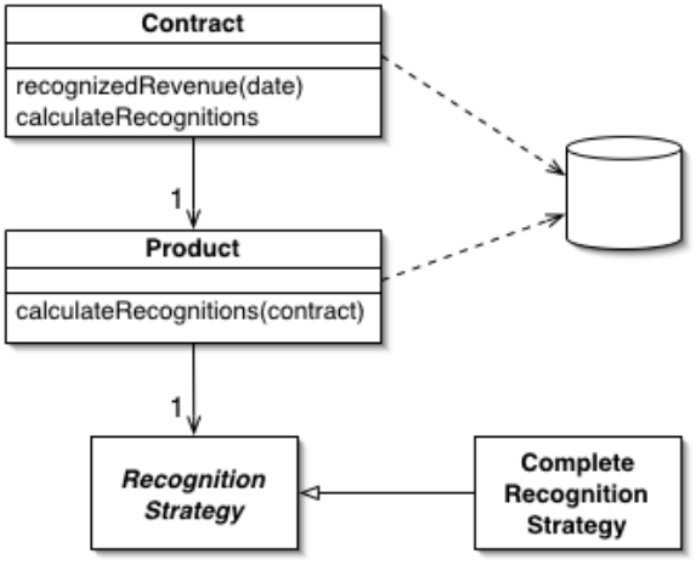
| P of EAA Catalog |

Domain Model

Semantik

An object model of the domain that incorporates both ~~behavior~~ and data.

For a full description see P of EAA page **116**



At its worst business logic can be very complex. Rules and logic describe many different cases and slants of behavior, and it's this complexity that objects were designed to work with. A Domain Model creates a web of interconnected objects, where each object represents some meaningful individual, whether as large as a corporation or as small as a single line on an order form.

Denotationelle Semantik

```
default double semantics(Fixture fixture) {
    return switch (this) {
      case Single(var singleFixture) -> {
        if (fixture.equals(singleFixture))
          yield 1.0;
        else
          yield 0.0;
        }
      case Scaled(var factor, var cue) ->
        cue.semantics(fixture)  * factor;
      case Combination(var cues) ->
        cues.stream()
      .mapToDouble(cue -> cue.semantics(fixture))
          .max()
          .orElse(0.0);
        };
    }
```

1. Abstraktion

2. Abstraktion

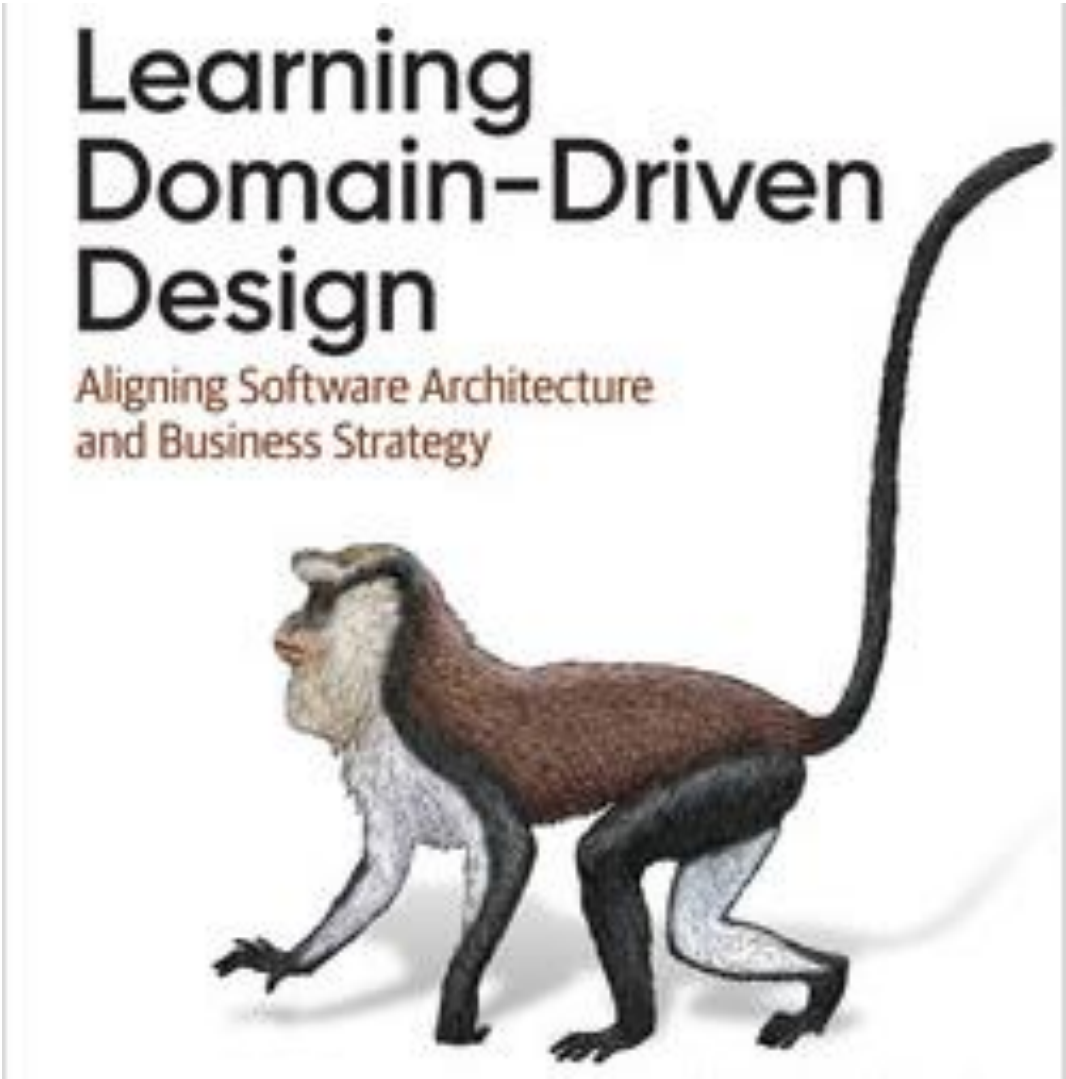
3. Abstraktion

Paul Hudak

@ctive group

Ubiquitous Language

"... business domain-related terms only."

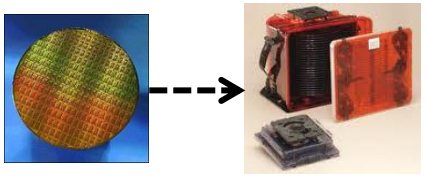


Design-Prozess

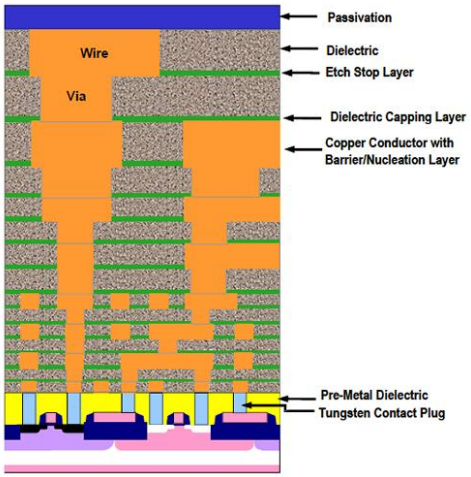
- **Datenmodellierung**
  mit Summen und Produkten
  (*Konstruktionsanleitungen*)
- wende **Abstraktion** an
- entwerfe **Kombinators**
- macje **Semantik** explizit

Semiconductor Manufacturing Process

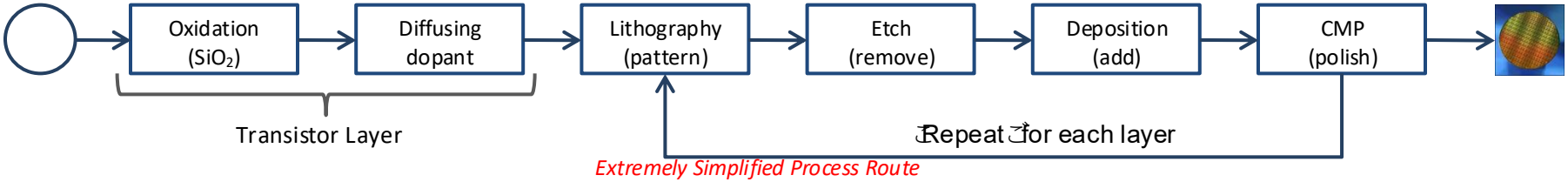
- The product Wafer contains many chips (devices)
  - Wafers are contained in FOUPs for movement through the factory



- The chip has multiple layers to create the functioning device
  - The first few layers are the transistors
  - The remaining layers are the interconnection 「wiring」



- The manufacturing process moves the wafers through multiple equipment to fabricate each layer



- Each step in the route has a recipe that is unique for the given product.

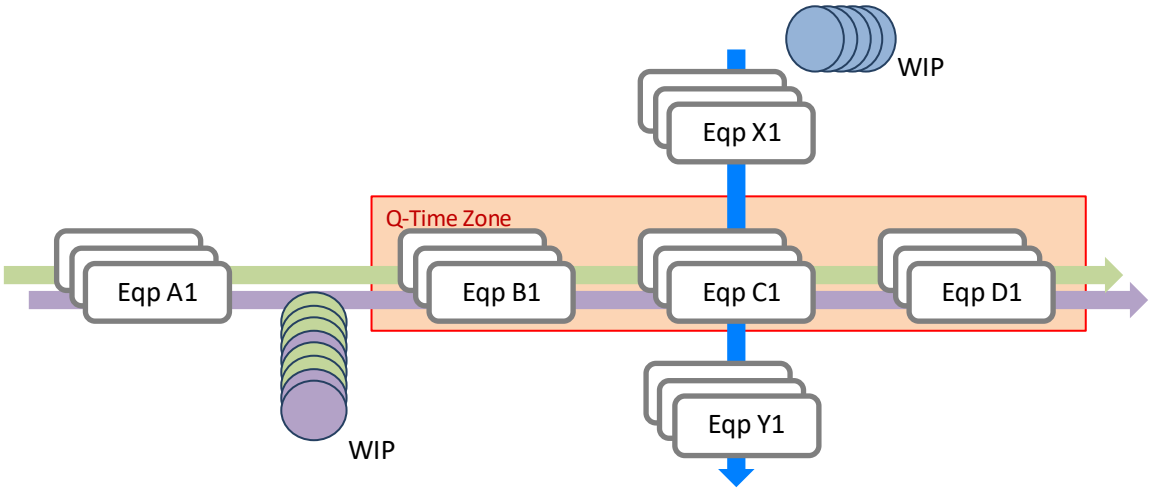
Representation für Routen

```
public enum Operation {
   TRACKIN, PROCESS, TRACKOUT }

public record
   Route(List<Operation> ops) {}

Route1 r1 = new Route1(List.of(TRACKIN, PROCESS,
                  PROCESS, TRACKOUT));
```

Q-Time Zones



Per Route:

- 1000 Operations
- 50 separate Q-Time Zones

Q-Time-Zonen

```
public record
    Route(List<RouteElement> elements)
     {}

public sealed interface RouteElement {
    record RouteOp(Operation op)
     implements RouteElement {}
    record RouteQTZone(Duration duration,
                       List<Operation> ops)
     implements RouteElement {}
}
```

Q-Time-Zonen

```
public record
    Route(List<RouteElement> elements)
     {}

public sealed interface RouteElement {
    record RouteOp(Operation op)
     implements RouteElement {}
    record RouteQTZone(Duration duration,
                       Route route)
     implements RouteElement {}
}
```

Lebe wild und gefährlich

- Produkte **und** Summen
- Immutability FTW
- Kombinatoren
- Abstraktion FTW
- Semantik für kontextübergreifende Konsistenz

iSAQB® ACCREDITED Training

https://www.active-group.de/schulung/