Funktionale Programmierung vs. Domain-Driven-Design: Unterschiede erkennen und nutzen

Michael Sperber
@sperbsen@discuss.systems

- Individualsoftware
- branchenunabhängig
- funktionale Programmierung
- Schulungen, Coaching
- iSAQB-akkreditiert
  Funktionale Softwarearchitektur
  Domänenspezifische Sprachen

www.active-group.de

funktionale-programmierung.de

Berlin

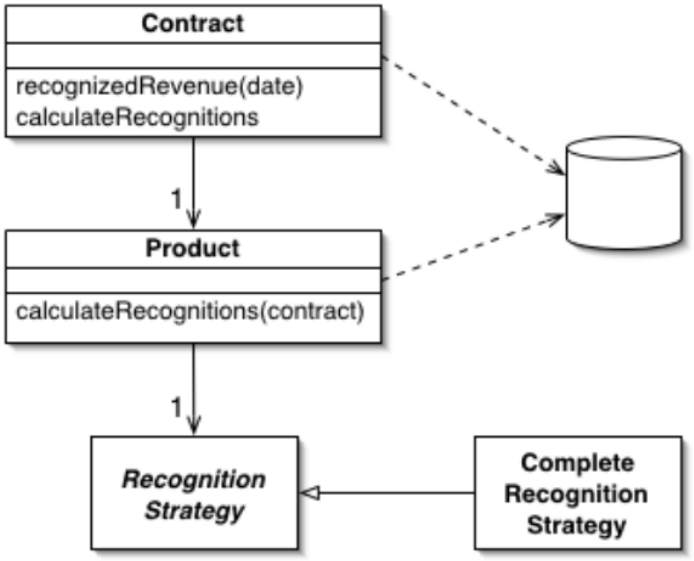
bobkonf.de/2025/

| P of EAA Catalog |

Domain Model

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.

Objekte und ihr Zustand

Quelle: https://commons.wikimedia.org/wiki/File:Gotisches_Haus_%28Brandenburg_an_der_Havel%29.jpg
(Creative Commons Attribution 3.0 Unported)

Alan Kay on OO

"Though OOP came from many motivations, two were central. [...] to find a more flexible version of assignment, and then to try to eliminate it altogether."

Alan Kay, *History of Smalltalk*
Communications of the ACM, 1996

Funktionale Programmierung

- (fast) alles unveränderlich
- höherstehende Abstraktionen
- Mathematik FTW
- Kombinatormodelle

Tiere auf dem texanischen Highway - OO

```
interface Animal {
  void runOver();
  void feed(Amount amount);
}
```

Quelle: Wikipedia
CC Attribution 3.0 Unported

Tiere auf dem texanischen Highway - FP

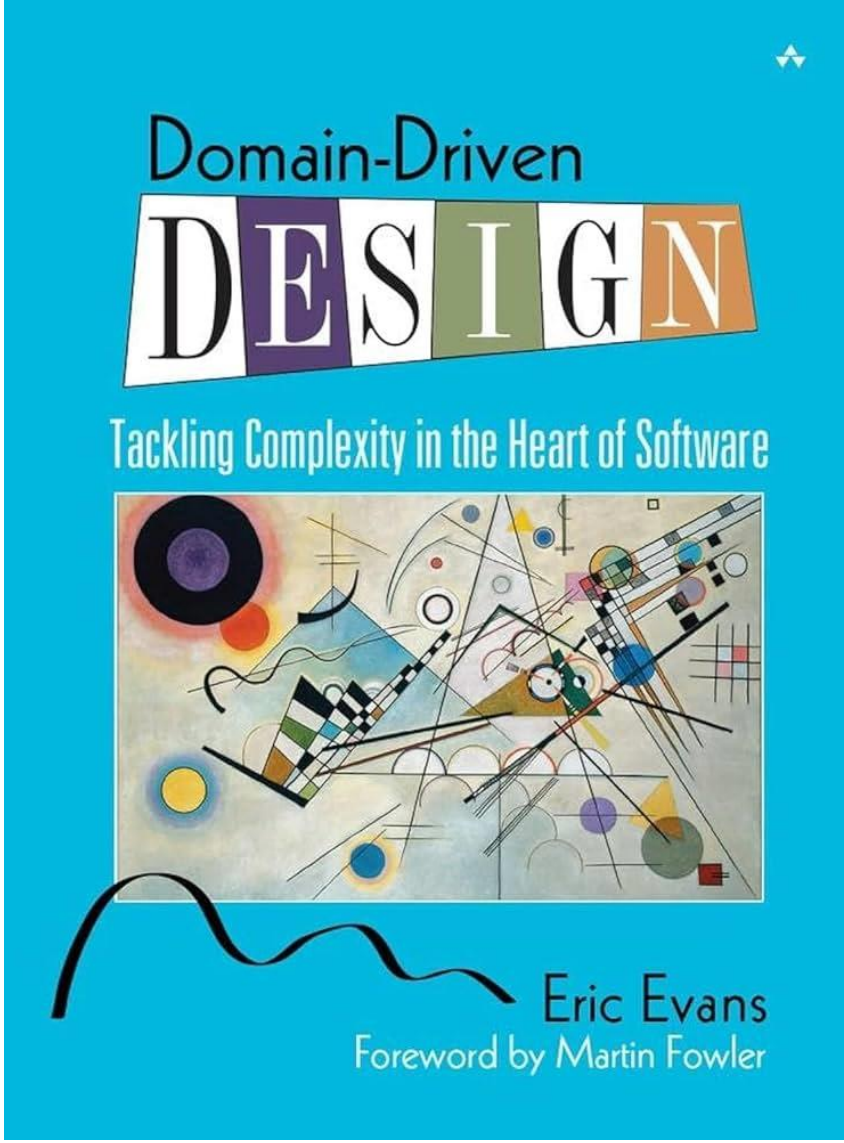
```
runOverAnimal :: Animal -> Animal

feedAnimal :: Weight -> Animal -> Animal
```



DATA UND FUNCTIONS...
EVERYWHERE

DDD

- “supple design”
- “deep model”
- “expressive design”
- “support change”
- “make composition safe”



Realität

Id	Art	Name	Kunde	Datum 1	Beitrag	Prop1 (NULL)	Prop2 (NULL)	Prop3 (NULL)	Prop4 (NULL)	Prop5 (NULL)	Prop5 (NULL)	Prop6 (NULL)	Prop7 (NULL)	Prop8 (NULL)

Offensichtliche Unterschiede

OOP

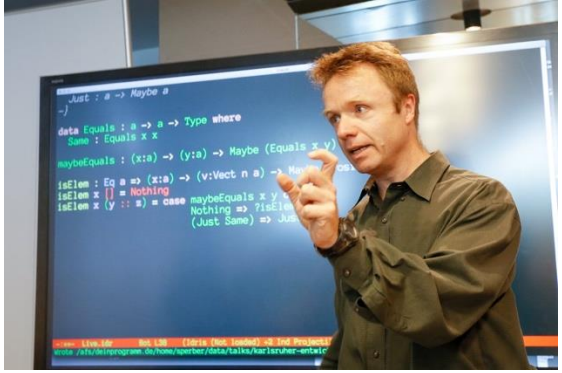
- Klassen, Methoden, Objekte
- Zustand durch Mutation
- abstrakte Klassen
- nominaleTypsysteme
- Turm von Konzepten
- erst erzeugen, dann validieren

FP

- Funktionen, Daten
- unveränderliche Daten
- algebraische Datentxpen
- strukturelle Typsysteme
- Higher-Order-Abstraktionen
- ”make illegal states unrepresentable”

Offensichtliche Synergien

- Programmiersprachen
  - Generics
  - Lambda-Ausdrücke
  - Streams
  - Records
  - Switch-Ausdrücke
  - Pattern-Matching
  - “lightweight concurrency”
  - Kotlin
- value objects FTW
- ”Immutable DDD”
- FP + Strategisches DDD
- ausdrucksstärkere Namen in Haskell



- Collaborative Modeling
  Prozesse
- Strategic Design
  Ubiquitous Language
- BDD
  Signaturen von Operationen
- TDD
  Testfälle & Funktionen

- Konstruktions–anleitungen
  Datenanalyse
- Konstruktions–anleitungen
  Funktionen
- Abstraktion
  Typen & Kombinator-Modelle
- Korrektheit & Konsistenz
  Mathematik

Closure of Operations

"Where it fits, define an operation whose return type is the same as the type of its argument(s). If the implementer has state that is used in the computation, then the implementer is effectively an argument of the operation, so the argument(s) and return value should be of the same type as the implementer. Such an operation is closed under the set of instances of that type. A closed operation provides a high-level interface without introducing any dependency on other concepts."

Evans, *Domain-Driven Design*

Kombinatormodelle

- Einkaufskörbe
- Diagramme
- Musik
- Animationen
- Produktionsrouten
- Versicherungsverträge
- Finanzprodukte
- ...

Auswahl

- Cargo-Beispiel aus dem Evans-Buch
- Kombinatormodell für Steuern
- Finanzverträge

- Programmiersprache?

Links

- A combinator library for taxes
  https://tinyurl.com/mryubyyf
- Gitpod für Haskell
  https://github.com/active-group/gitpod-haskell

Blaues Buch: Cargo-Beispiel