

Model Driven Development

einige

wichtige

Grundprinzipien

Johannes Scheier

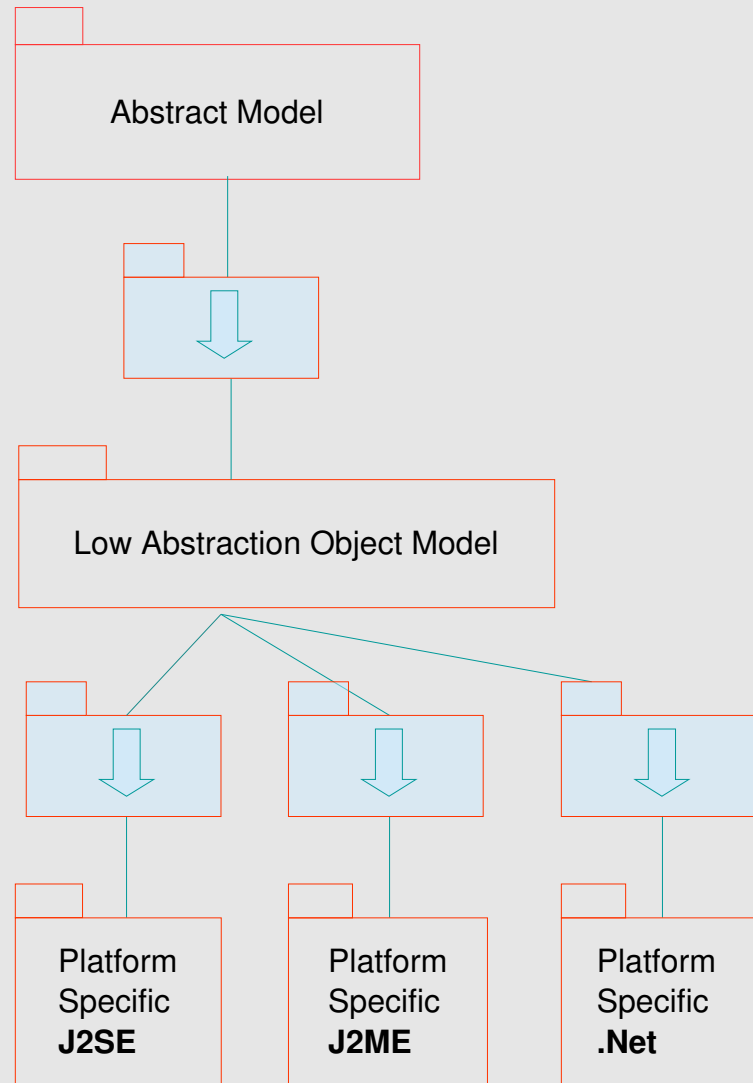
[j@scheier-software.ch](mailto:j@scheier-software.ch)

# Inhalt

- Was ist Model Driven Development (MDD)?
- Was verspricht MDD?
- Was ist Abstraktion?
- Die Grundprinzipien
- Tool-Demo

# Was ist MDD?

## Model Driven Development



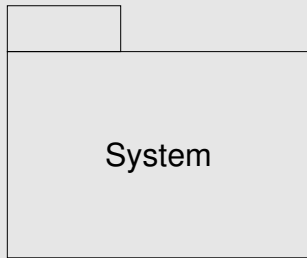
- MDD ist ein Konzept zur Entwicklung von Software
- Das System wird in einem abstrakten Modell beschrieben.
- Das abstrakte Modell wird durch automatische Transformation in Code transformiert.
- Die Transformation kann in mehreren Schritten erfolgen.
- Das Modell am Schluss der Transformationskette ist die Implementation des Systems.

# Was verspricht MDD?

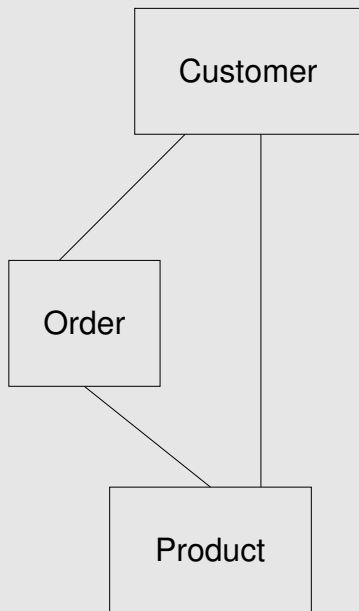


- Die Entwicklungszeit wird verkürzt
- Das Modell ist „zeitlos“
  - Es altert mit der Domäne und nicht mit der Technologie
- Bessere Dokumentation
  - Das Modell ist die bessere Dokumentation als der Code
  - besser lesbar
  - durch Generierung immer konsistent
- Das System ist besser anpassbar
- Konsequenterer Umsetzung einer Softwarearchitektur
- Architektur kann nachträglich geändert werden
- Plattformunabhängigkeit
- Bessere Aufgabenteilung im Projekt

# Was ist ein abstraktes Systemmodell?



- Abstraktion des wirklichen Systems
- Beschreibt nur das Wesentliche
- Abstrahiert vom Unwesentlichen



- Modellelemente sind Abstraktionen von Teilen des wirklichen Systems

# Abstraktionsgrad

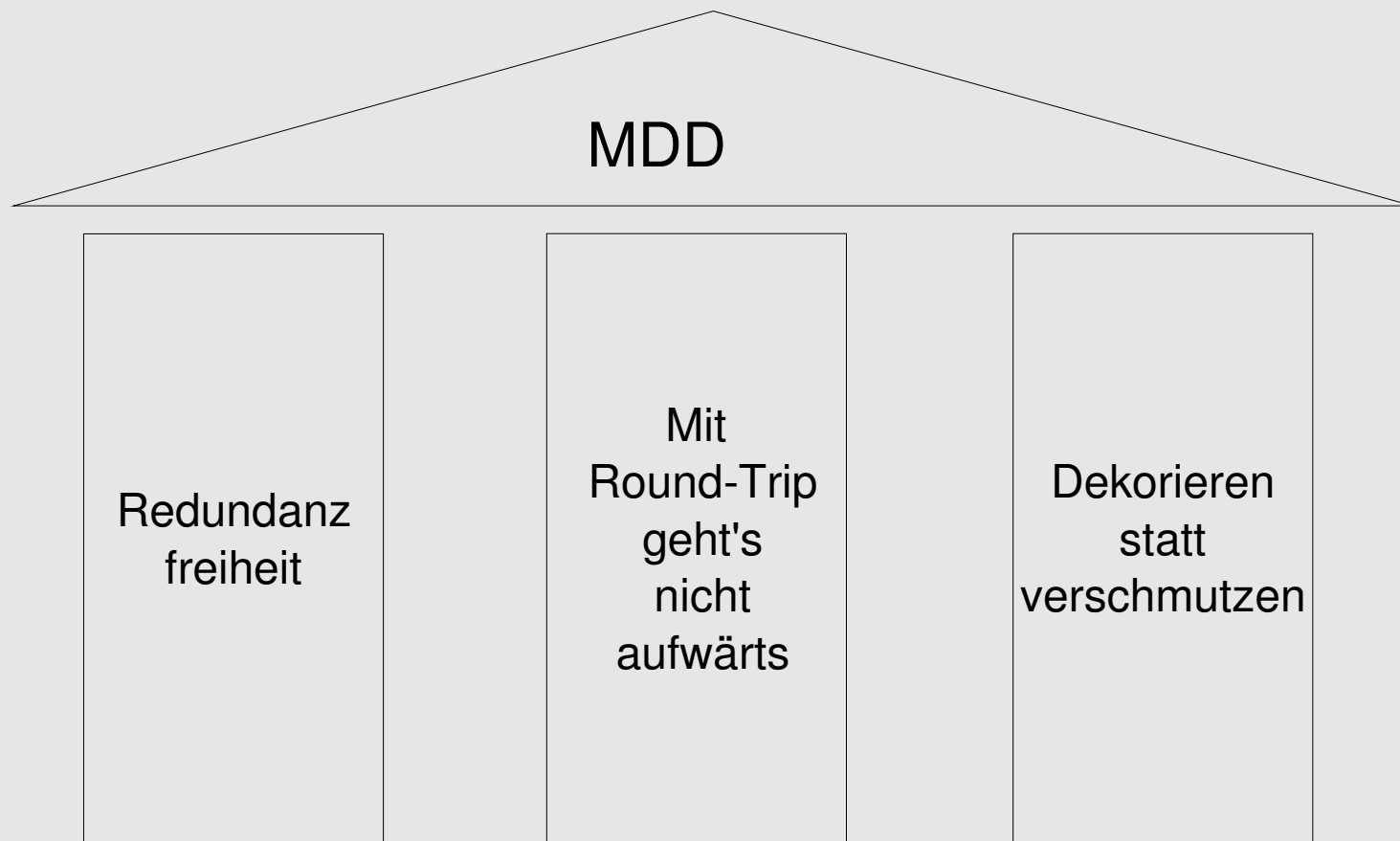
```
switch(event)
{
  case eventX:
    switch state
      case A:
        ....
  case eventY:
    switch state
      case A:
        ....
}
```



- Geringe Abstraktion
  - Modellelemente sind einfach
  - semantisch arm
  - viele Elemente werden kombiniert um etwas komplexes auszudrücken
- Hohe Abstraktion
  - Modellelemente sind semantisch reich
  - ein einziges Element verbirgt schon eine Menge Komplexität

# Die Grundprinzipien

- MDD ist nicht einfach Code-Generierung aus Modellen.
- Nur wer die nachfolgenden Grundprinzipien beachtet, kann von den Vorteilen von MDD profitieren



# Redundanzfreiheit im Modell

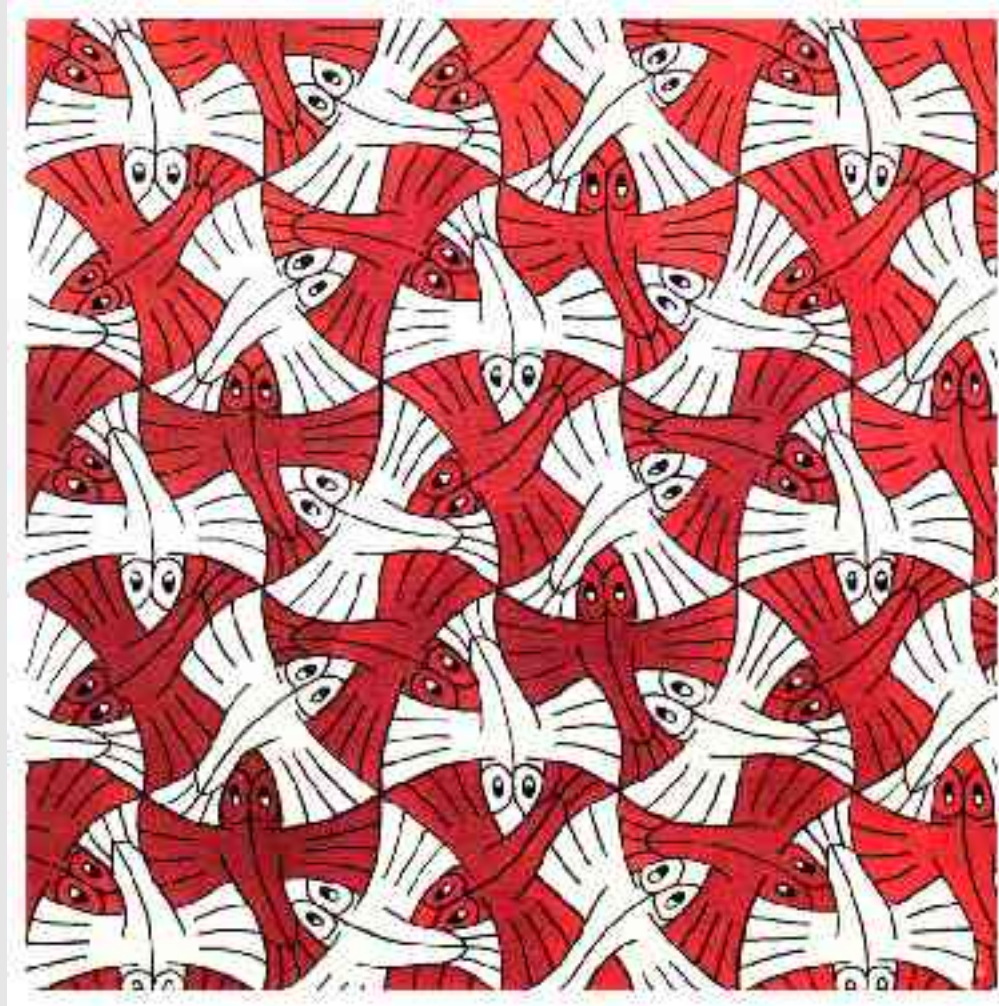
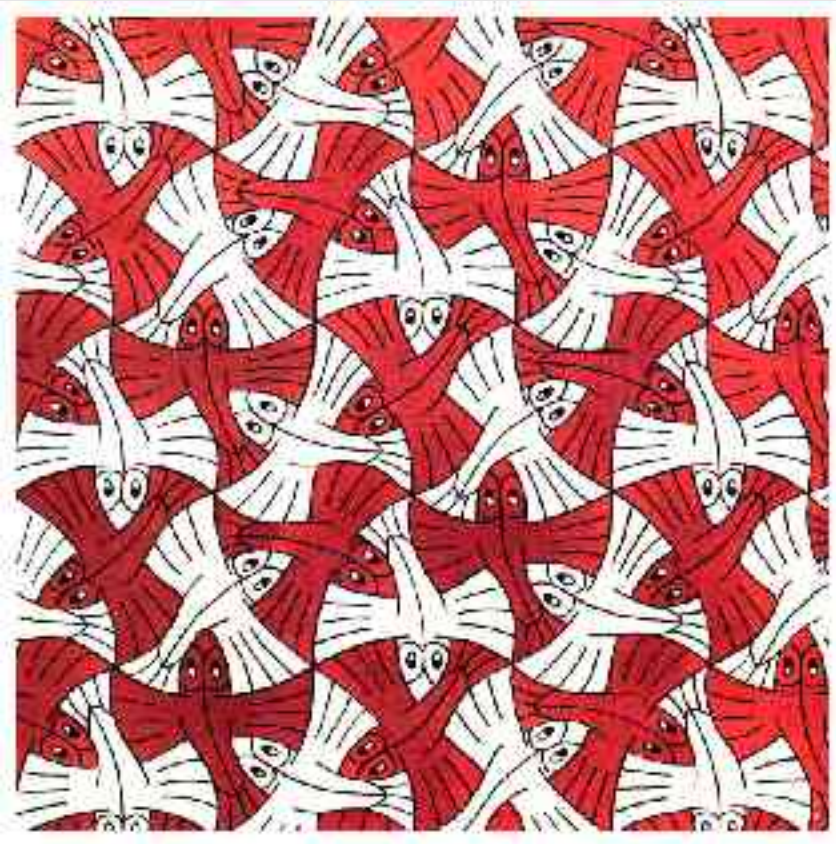


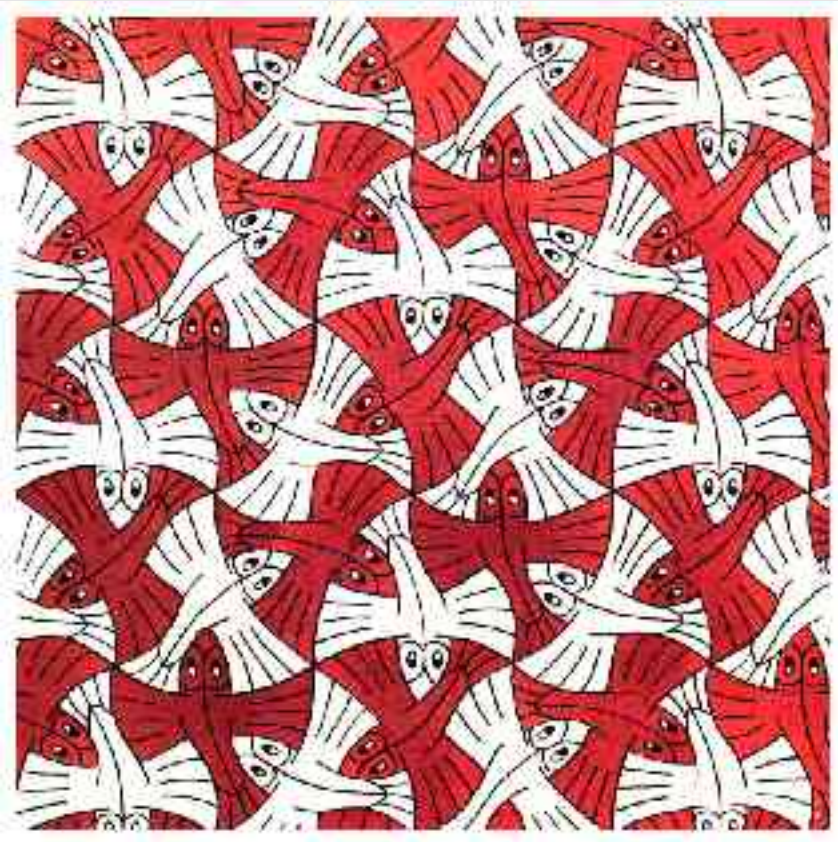
Bild von Maurits Cornelis Escher, niederländischer Künstler, von 1898 bis 1972

# Was ist Redundanz?



- Latein: redundare -> im Überfluss vorhanden sein
- Informationstheorie:
  - das mehrfache Vorhandensein ein und derselben Information
  - oder das umfangreiche Beschreiben einer Information, die auch kürzer dargestellt werden kann.

# Nachteile von redundanten Modellen für MDD



- Mehr Aufwand
  - Änderungen müssen an mehreren Stellen nachgeführt werden
- Weniger übersichtlich
  - durch überflüssige Informationen

# Redundanz Beispiele: referenzieren mit Werten (1/2)

- Referenzieren mit Werten ist eine verbreitete Art von Redundanz

Parameter	
Name	: String
Type	: <b>String</b>

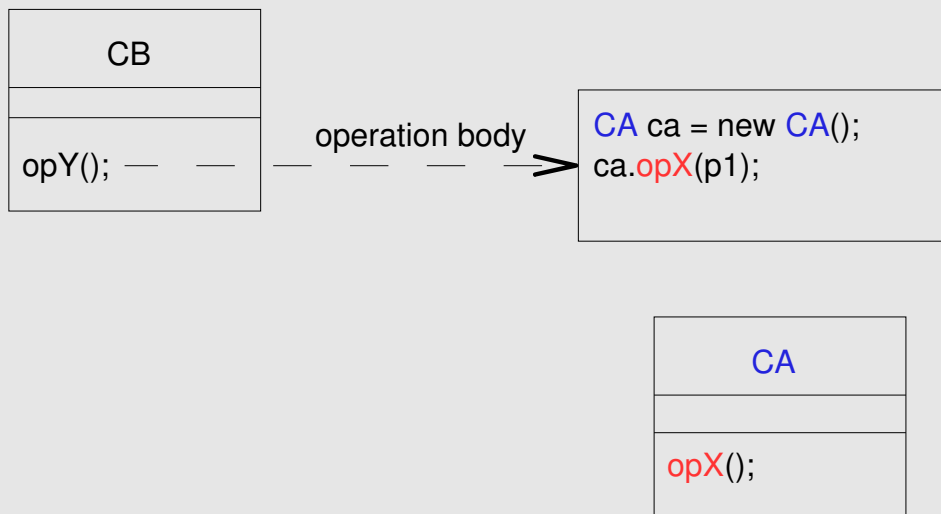
- Referenzen zu Modellelementen werden als Wert (meist als String) modelliert
- Wenn Klassenname geändert wird, müssen alle Parameter.Type geändert werden, die sich auf diese Klasse beziehen

Parameter	
Name	: String
Type	: <b>Reference</b> <Class>

- Vermeidung durch Verwendung von Referenz-Elementen
- Name trotzdem sichtbar

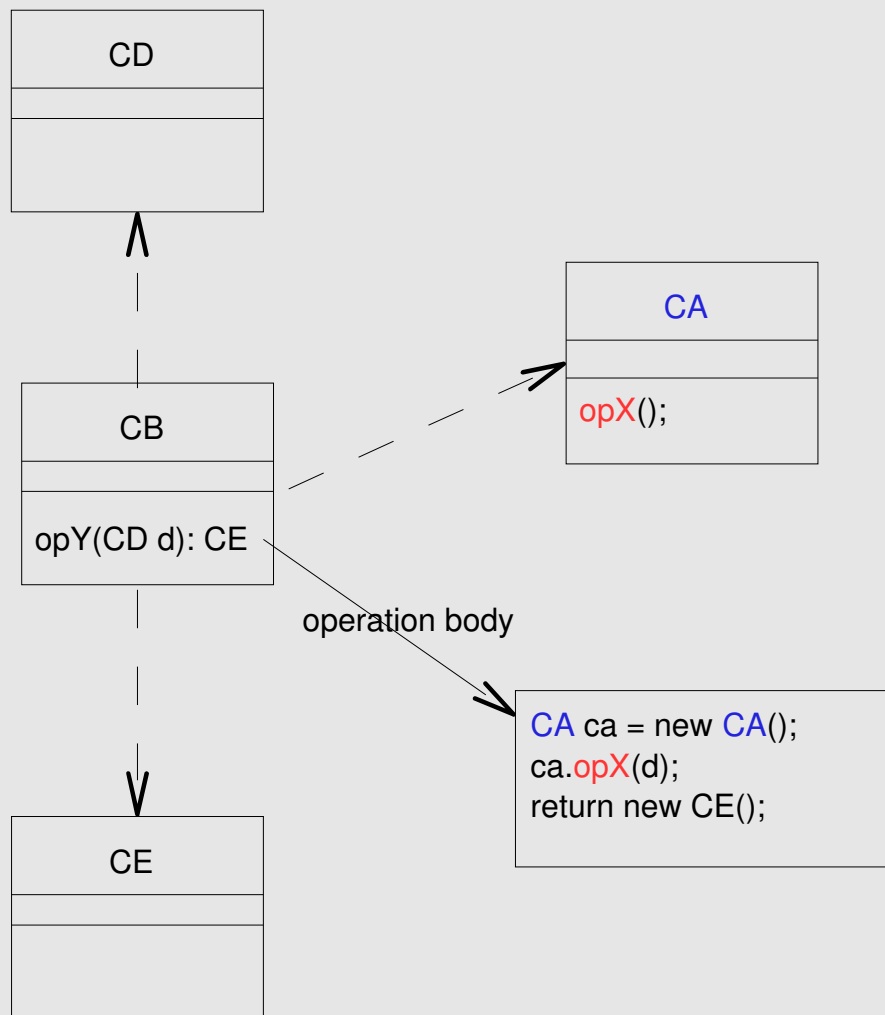
# Redundanz Beispiele: referenzieren mit Werten (2/2)

- In 3. GL Programmen sind alle Referenzen als Namen modelliert



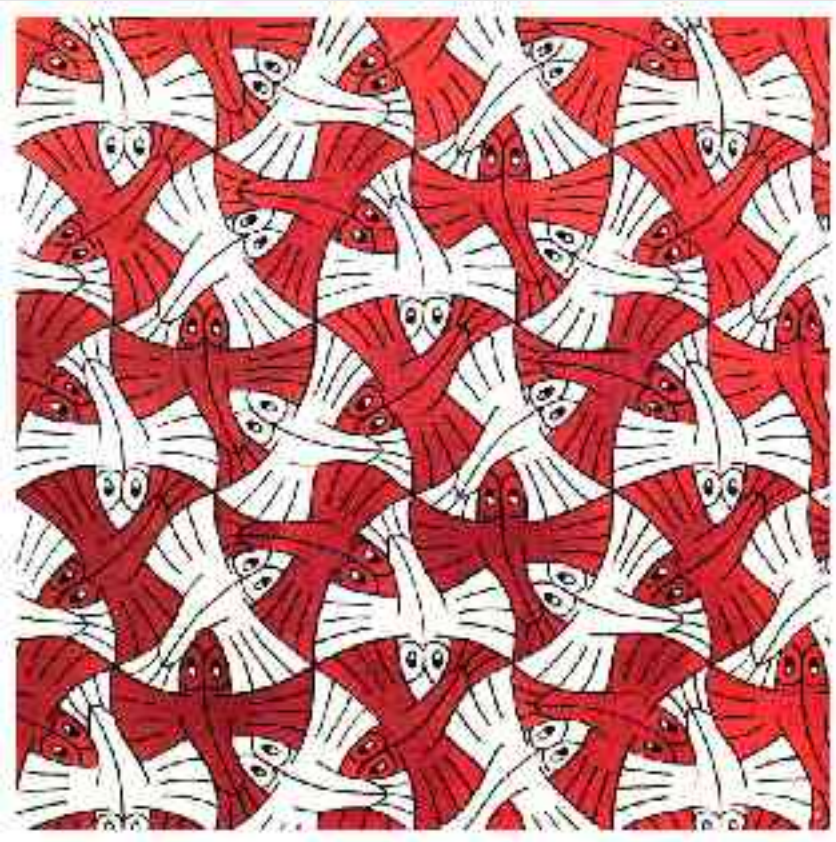
- Meta-Modelle die Verhalten als 3. GL Text modellieren erben dieses Problem von der 3. GL.
- Wenn sich der Name einer Operation ändert, müssen alle Aufrufe dieser Operation angepasst werden

# Redundanz Beispiele: weitere



- Explizites Modellieren von Dependencies in UML
  - Unnötige Arbeit
  - Diese Information ist bereits implizit vorhanden
  - Macht Modell unübersichtlich
  - Nicht mehr benötigte Dependencies werden nicht gelöscht

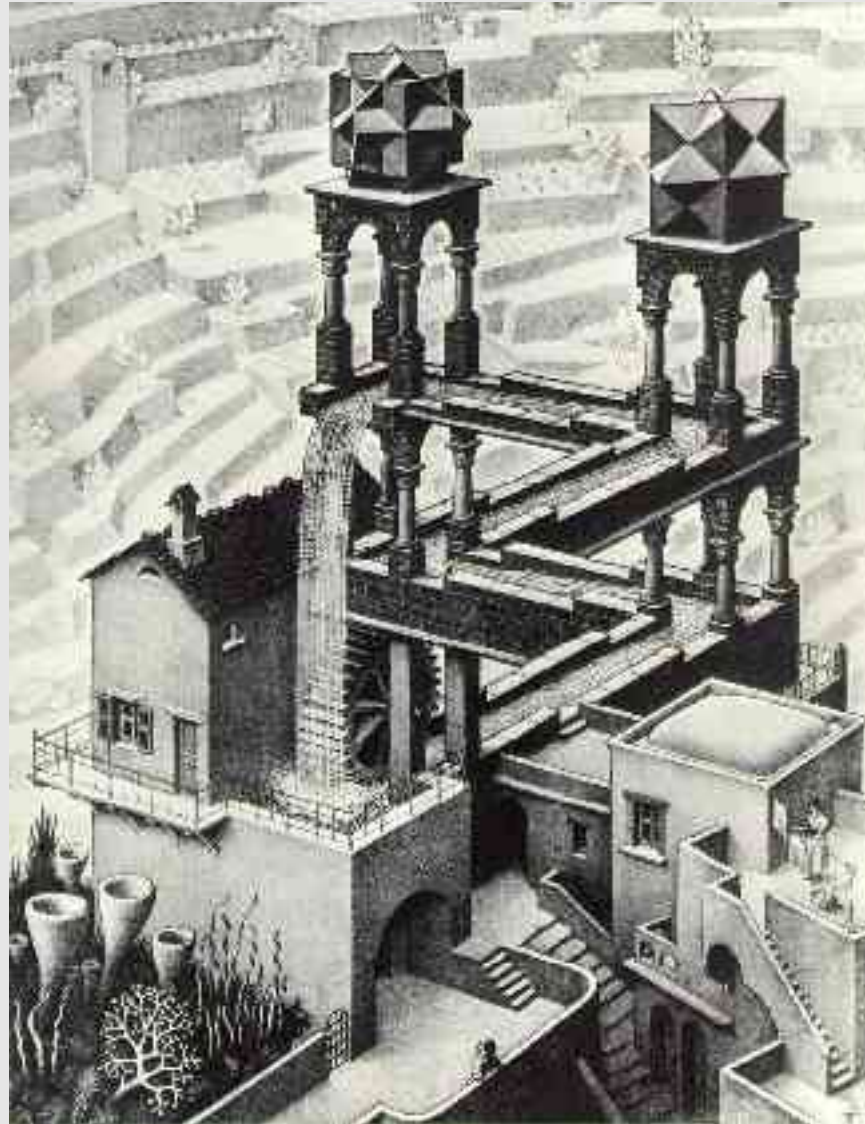
# Redundanzfreiheit!



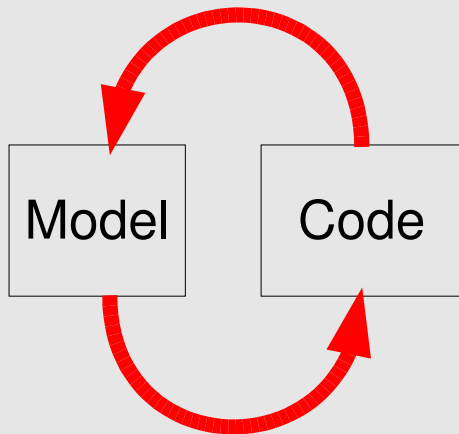
- In anderen Worten:

- Meta-Modelle sollten möglichst redundanzfreie Modelle ermöglichen!
- Referenzen mit speziellen Referenz-Elementen modellieren
- Information die implizit vorhanden ist, nicht explizit modellieren!

# Mit Round-Trip geht's nicht aufwärts

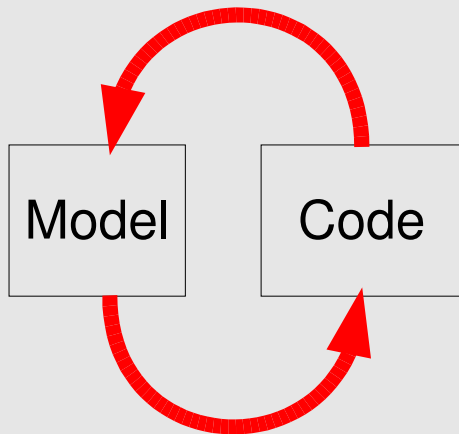


# Was ist Round-Trip Engineering?



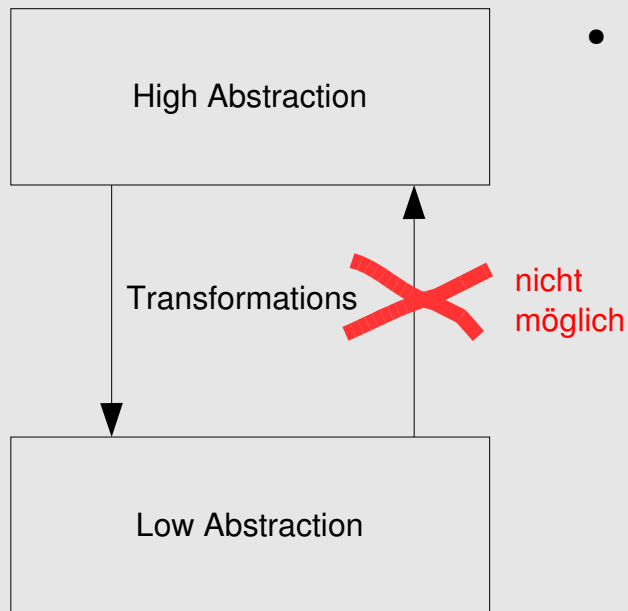
- Ein wenig Modellieren, ein wenig Codieren
- Beim Round-Trip Engineering kann nicht nur im Modell, sondern auch im Code geändert werden.
- Änderungen im Code werden automatisch ins Modell transformiert.

# Warum ist Round-Trip so populär?



- Viele Software-Entwickler
  - beherrschen eine 3. GL wie ihre Muttersprache
  - sie wollen nicht modellieren sondern codieren
  - sie müssen modellieren
- Round-Trip verspricht
  - dass die gewohnte Arbeitsweise beibehalten werden kann
  - unbeliebte Modellier-Arbeit zu automatisieren
- Das findet Anklang
- Gibt es solche Tools wirklich?
- Ja, aber..

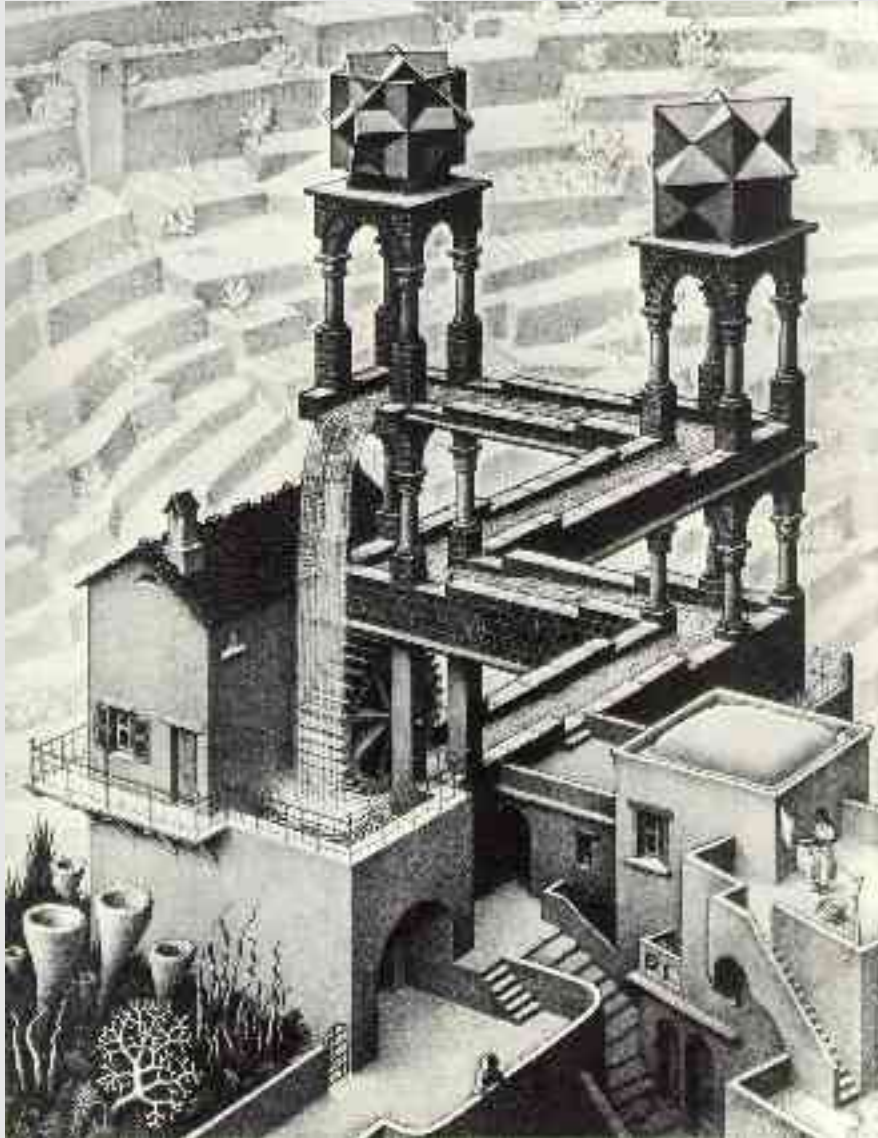
# Round-Trip, die Illusion



- Beispiele:

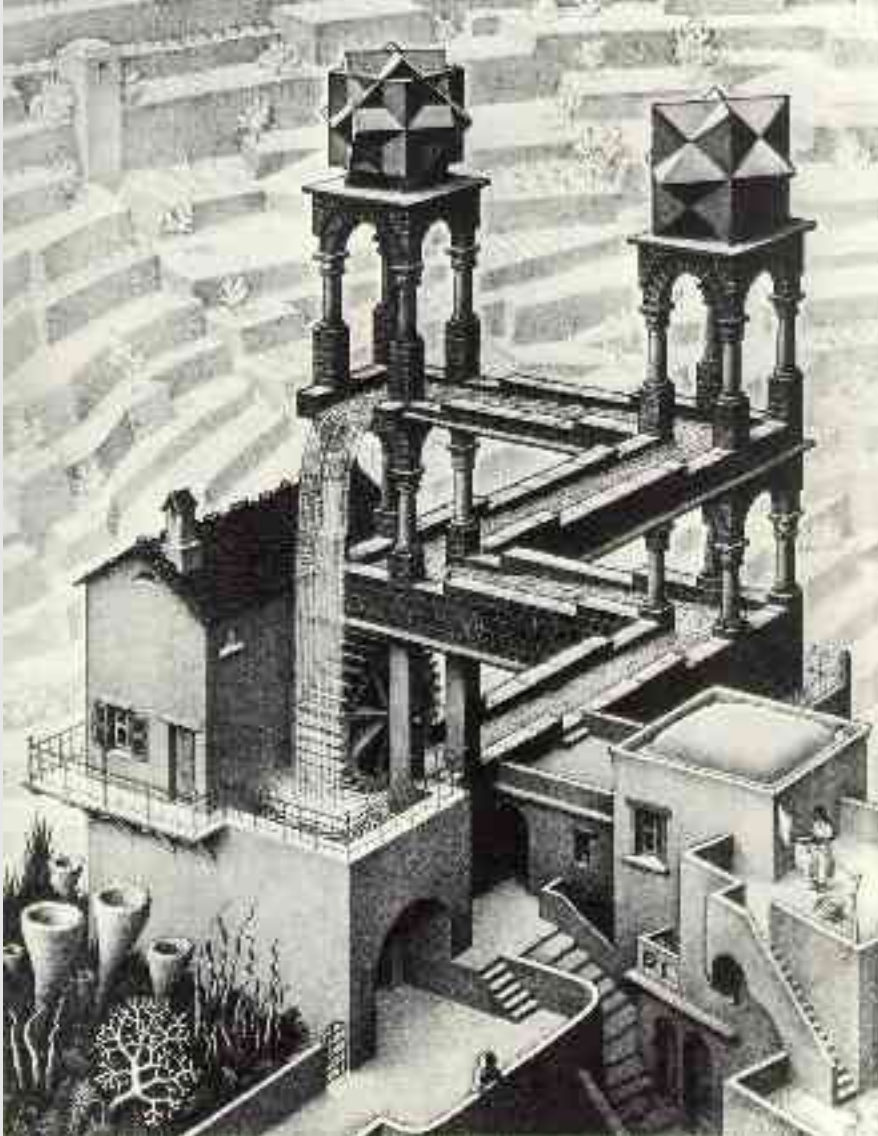
- Aus einem Bitmap, das Text darstellt kann man keine Absatzformate erkennen.
- Aus einem Member in Java kann man nicht erkennen, ob dieses einem Relationship **by value** oder **by reference** entspricht.
- Aus 3. GL. Code kann man nicht erkennen welche Abschnitte **State-machine-Semantik** haben.

# Round-Trip, Folgerung



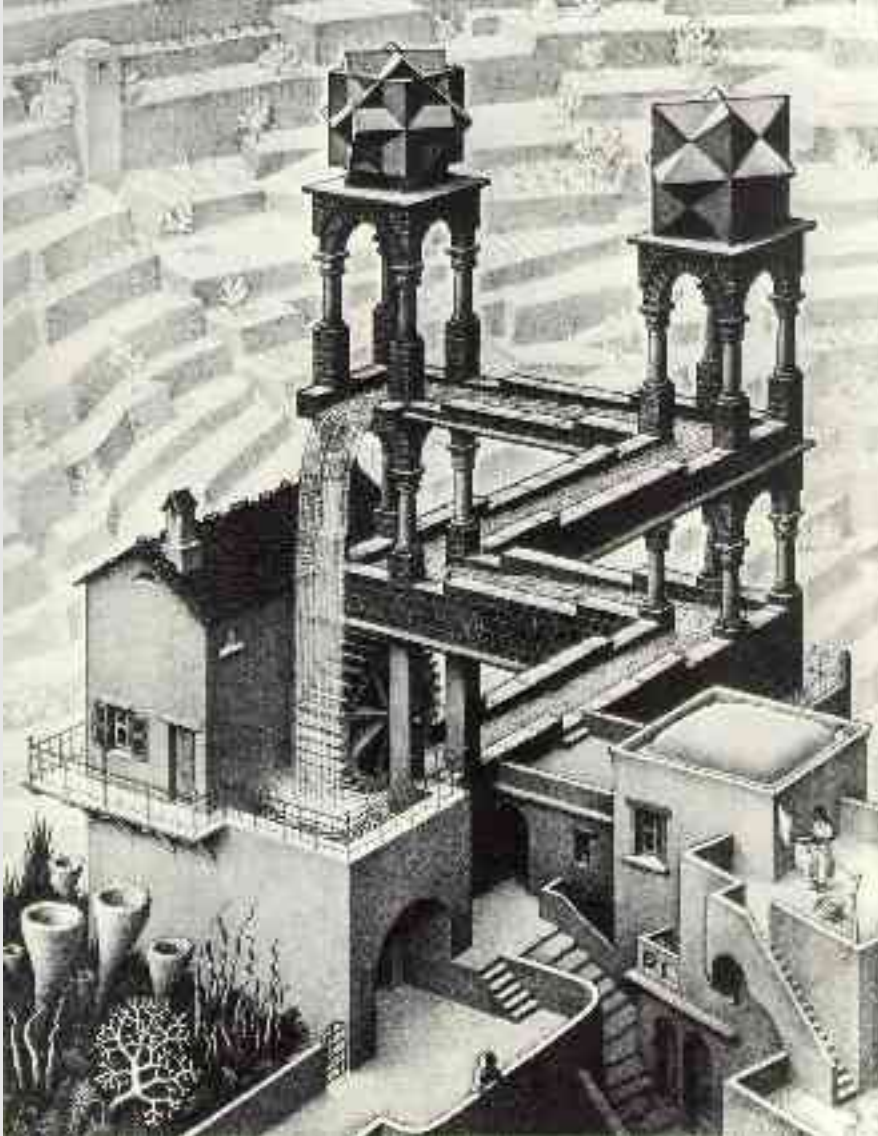
- Mit Round-Trip bewegt man sich auf dem Abstraktionsniveau von Code
- Man wechselt nicht die Abstraktion, nur die Darstellung

# Nachteile von Round-Trip für MDD



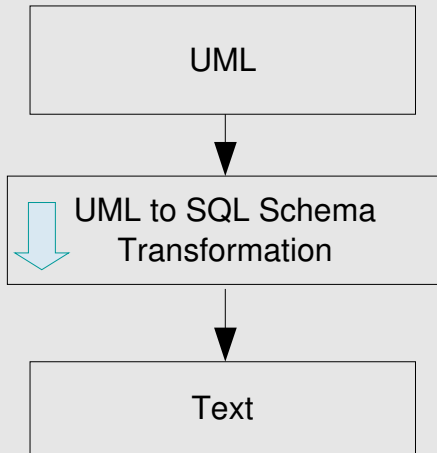
- Wer mit Round-Trip arbeitet **verzichtet** auf:
  - höhere Abstraktion
  - das Hauptinstrument von MDD!
- Damit **verzichtet** man auch auf:
  - einfache übersichtliche Modelle,
  - schnellere Modellierung,
  - besseres Verständnis der Modelle.
- Umkehrung gilt nicht!  
„Wer auf Round-Trip verzichtet ....“

# Mit Round-Trip geht's nicht aufwärts!



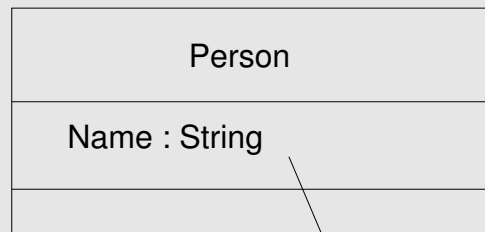
- In anderen Worten:
  - Generiere ausschliesslich vorwärts
  - Das abstrakte Modell ist der Master
  - Verwende höhere Abstraktion  
- verzichte auf Round-Trip
  - Arbeite konsequent top-down

# Dekorieren statt verschmutzen



- Motivation

- Transformationen, z.B. Code-Generatoren werden in der Regel so definiert, dass sie jedes Modell eines bestimmten Metamodells transformieren können.
- Oft möchte man den Code-Generator anweisen, bei einem Modellelement etwas Spezielles zu generieren



```
create index Name_index ON Person (Person_Name)
```

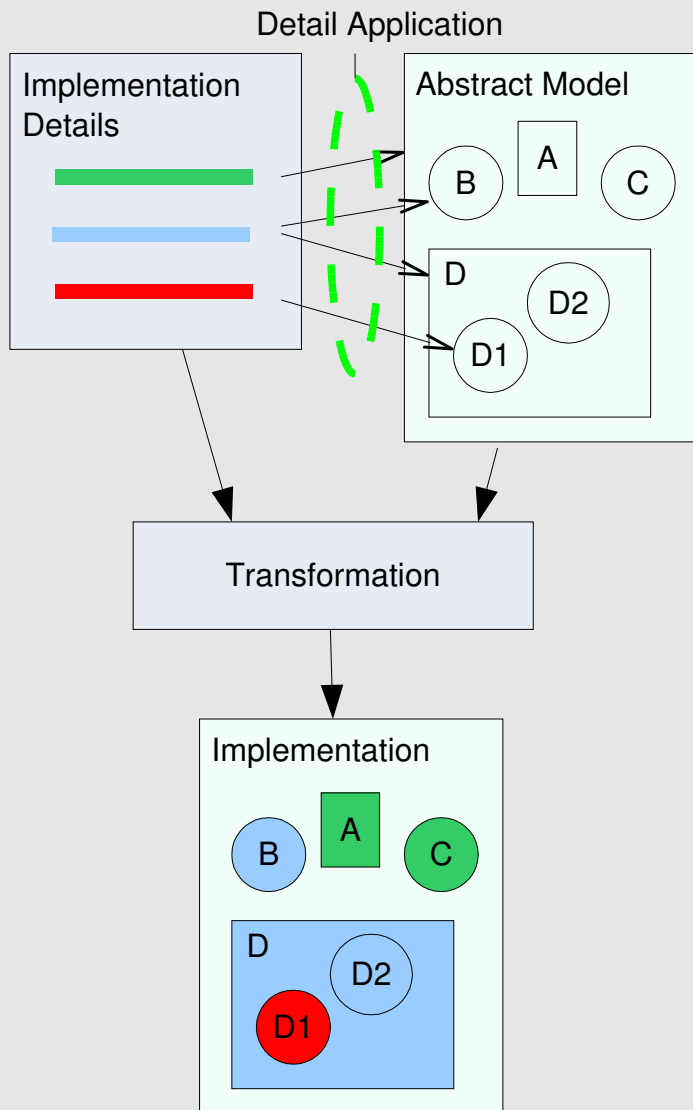
- Z.B. der Codegenerator soll zu einem bestimmten Attribut einen Index in der Datenbank anlegen

# Dekorieren statt verschmutzen

Person
Name : String <<Index>>

- Gängiger Weg:
  - Hinweise zur Code-Generierung im abstrakten Modell
  
- Nachteile:
  - Verschmutzung des Modells
  - unübersichtlich
  - keine klare Trennung zwischen abstraktem Modell und Implementation
  - bei mehreren Zielplattformen addieren sich diese Nachteile

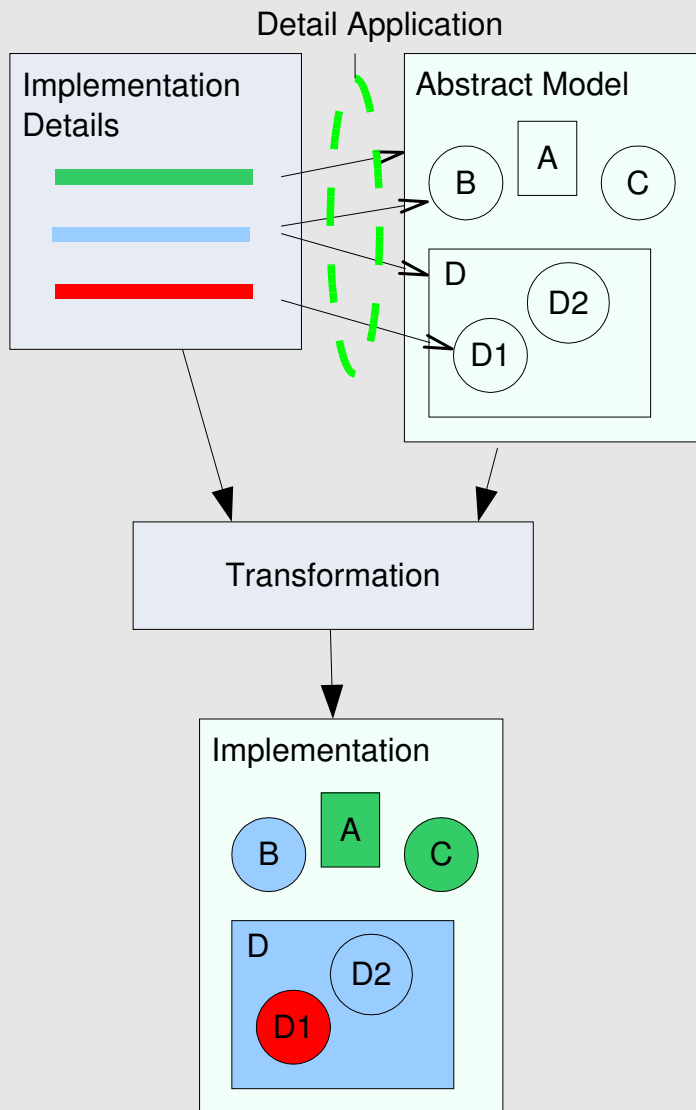
# Dekorieren statt verschmutzen



- Nicht-invasive Methode

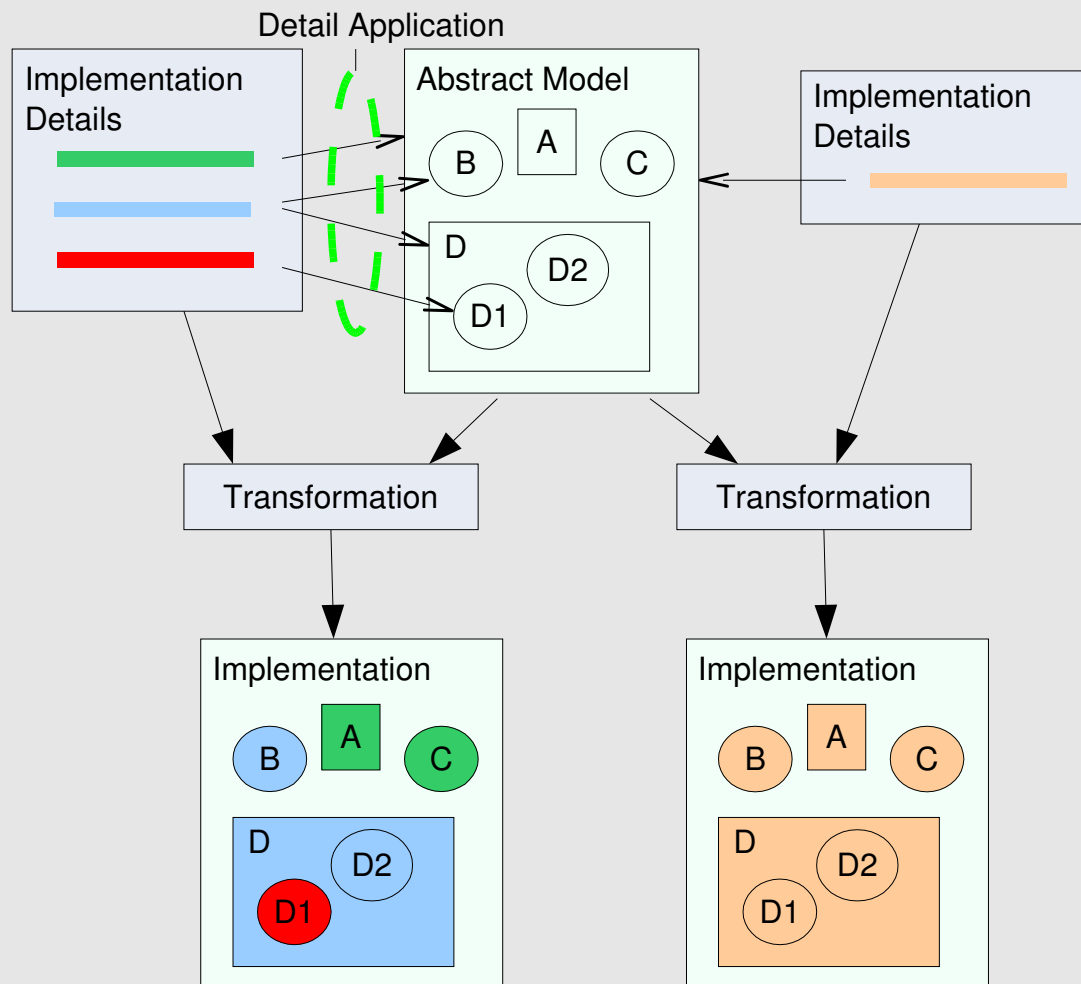
- Hinweise für den Code-Generator ausserhalb des abstrakten Modells
- Abstrakte Modellelemente werden nur referenziert

# Dekorieren statt verschmutzen, die Vorteile für MDD (1/2)



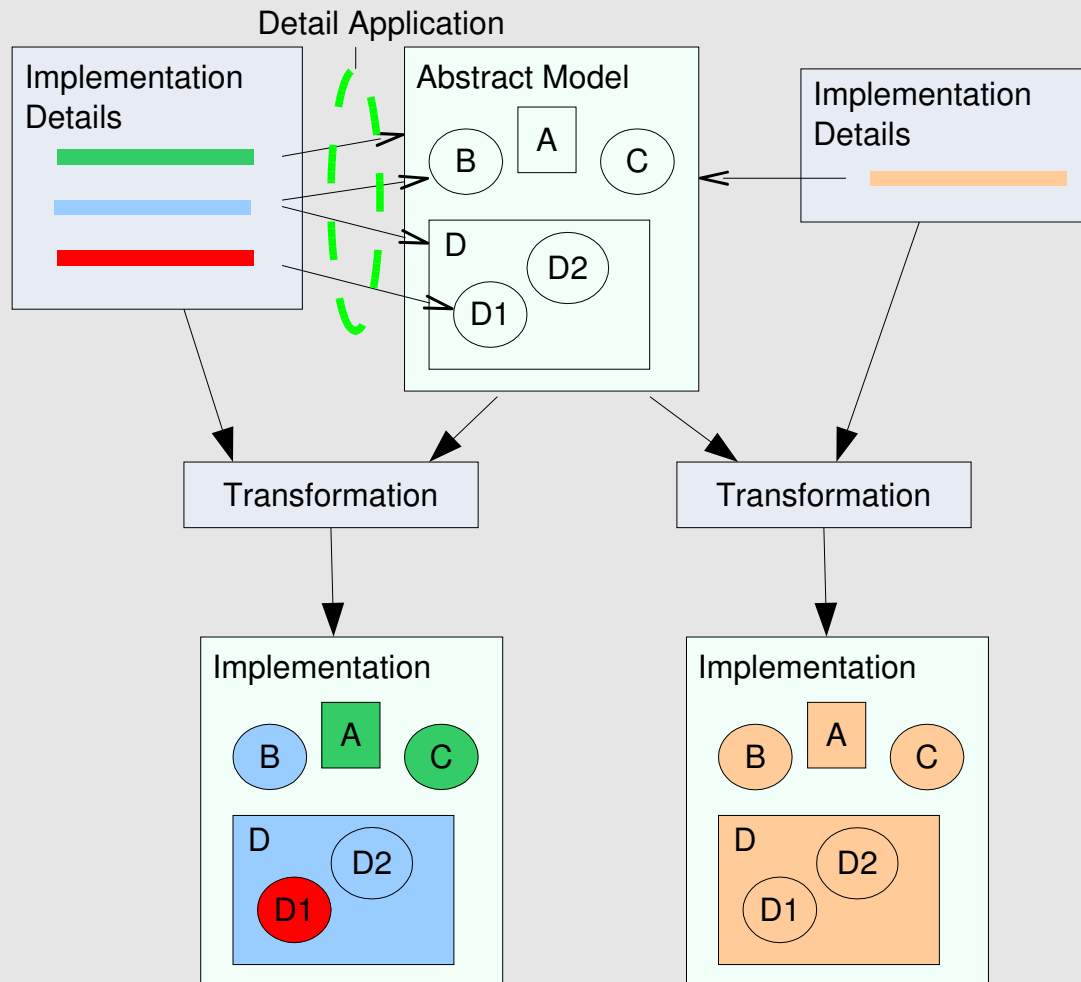
- Abstrakte Modelle
  - bleiben rein, übersichtlich, zeitlos, „stainless“
  - definieren nur was das System können muss!  
- Nicht wie es gelöst wird
  - dadurch bleibt das Hauptinstrument von MDD intakt!
- Arbeitsteilung
  - Domain-Experte
  - Software-Experte

# Dekorieren statt verschmutzen, die Vorteile für MDD (2/2)



- Verschiedene Anreicherungen sind möglich
  - Performance Profiling
  - Debugging
  - Documentation
  - verschiedene Plattformen
  - verschiedene Anwendungsgruppen
  - ...

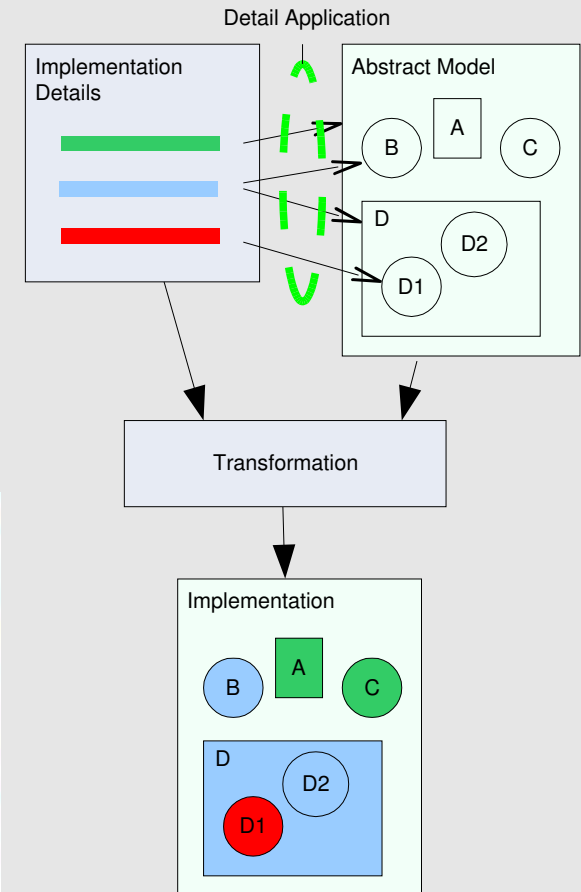
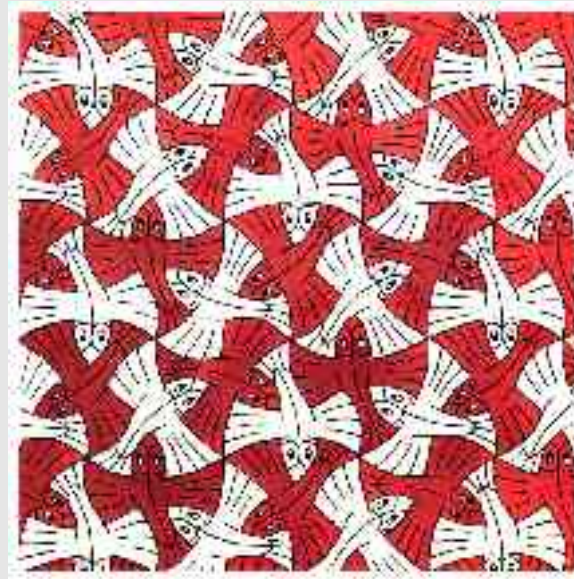
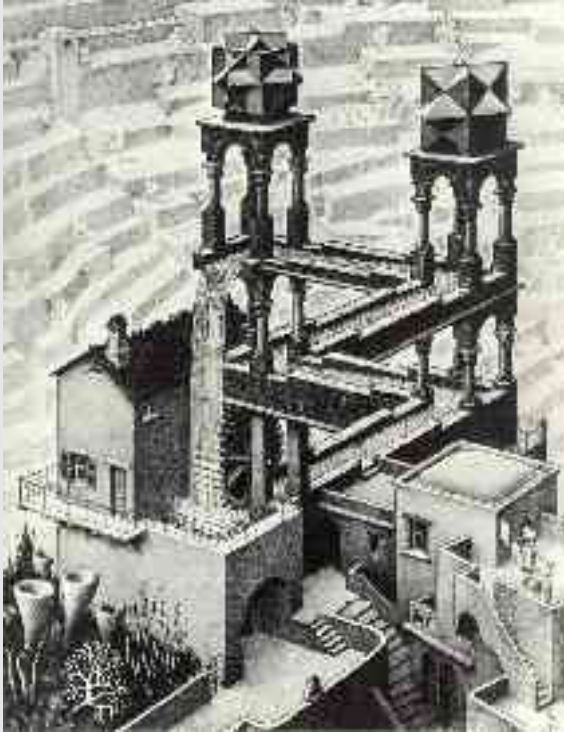
# Dekorieren statt verschmutzen!



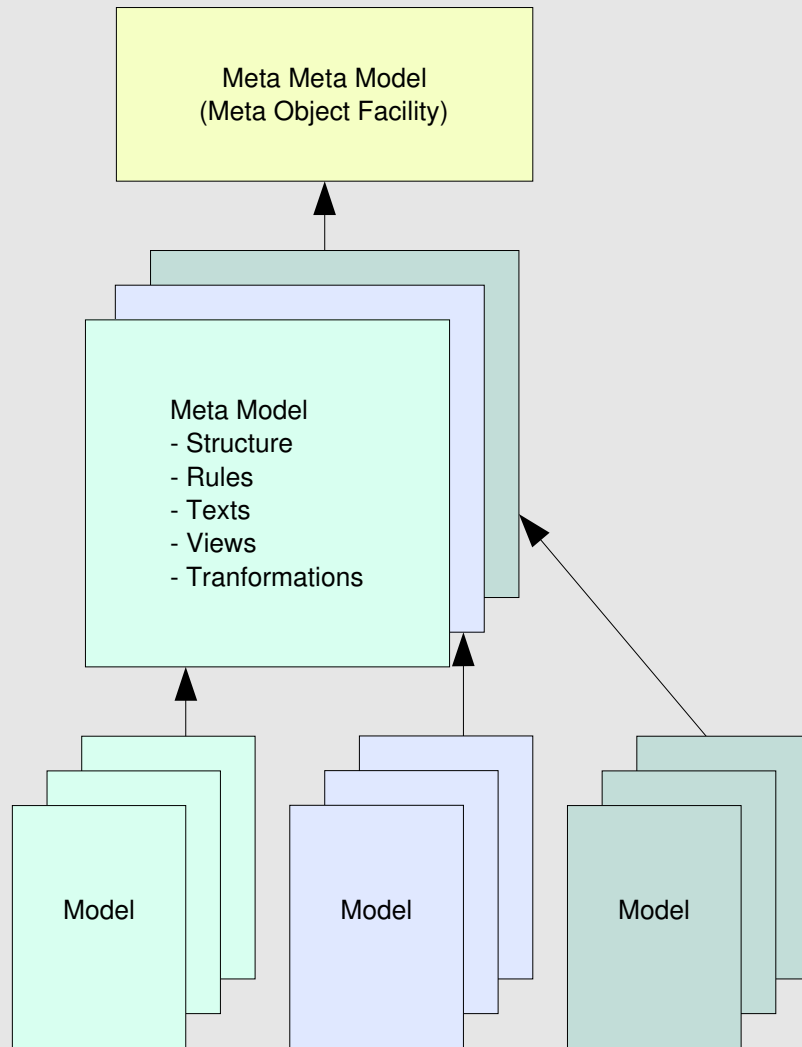
- In anderen Worten

- Implementations Details vom abstrakten Modell separieren
- Nicht-invasiv anreichern

# Die Grundprinzipien Zusammenfassung



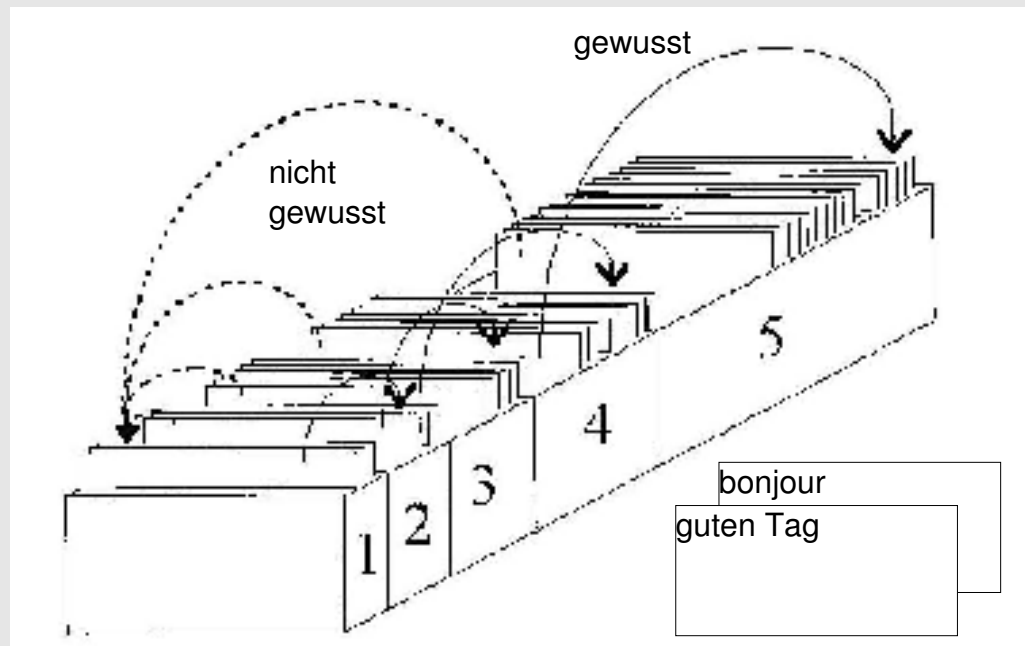
# MBSD Tool



- MBSD = Model Based Software Development
- MBSD selbst basiert auf Modellen
- MBSD-UML ist ein Meta-Modell für UML
  - redundanzfrei
  - höhere Abstraktionen wie Relationship, Statemachine, Active Object ...
  - mit nicht invasiven Implementationsdetails
- Bald öffentlich verfügbar

# MBSD, Demo

- Demo-Beispiel „Quanda“
  - Lernprogramm das auf dem Prinzip einer Lernkartei beruht
- Lernkartei
  - Hilfsmittel um beispielsweise Vokalbein zu lernen



# MBSD Demo