Agile Webentwicklung mit Ruby on Rails

Prof. Dr. Oliver Vornberger
Nils Haldenwang, B.Sc.

UNIVERSITÄT OSNABRÜCK

Institut für Informatik
AG Medieninformatik

Web Application Architecture

HTTP

Hypertext Transfer Protocol

Web Application Architecture

Quellen/Literatur

HTTP/1.0: http://tools.ietf.org/html/rfc1945

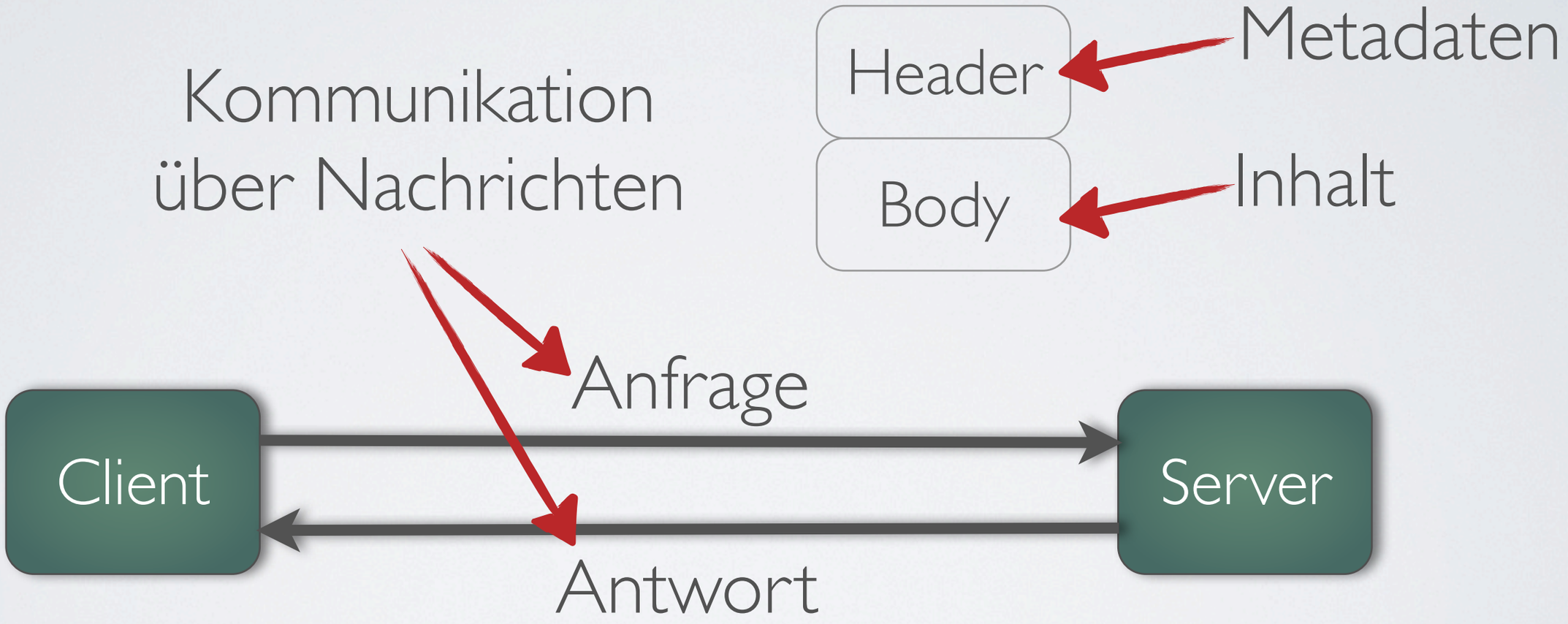
HTTP/1.1: http://tools.ietf.org/html/rfc2616

HTTP

- ab 1989 von Roy Fielding, Tim Berners-Lee und anderen am CERN zusammen mit URL und HTML entwickelt
- HTTP 0.9 im Jahr 1991, HTTP/1.0 im Mai 1996, HTTP/1.1 im Jahr 1999
- Protokoll zur Übertragung von Daten über ein Netzwerk
- Anwendungsschicht etablierter Netzwerkmodelle (ISO/OSI)
- zustandsloses Protokoll
- Client/Server-Architektur

Client/Server

Kommunikation
über Nachrichten



Anfrage (Request)

http://www.example.net/infotext.html

Protokoll

Hostname

Ressource

HTTP Version

```
GET /infotext.html HTTP/1.1
Host: www.example.net
```

HTTP Verb

HTTP Request

Antwort (Response)

Status Code

Header

Body

```
HTTP/1.1 200 OK
Server: Apache/1.3.29 (Unix) PHP/4.3.4
Content-Length: (Größe von infotext.html in Byte)
Content-Language: de
Connection: close
Content-Type: text/html

<html>
  <title>Infotext</title>
  <body>
    ...
```

HTTP Verben

GET	Anforderung einer Ressource
POST	Versenden von Daten, z.B. als Schlüssel/ Wert-Paare aus einem Formular
HEAD	Wie GET nur ohne Body
PUT	Ressource unter Angabe des Zielortes hochladen
DELETE	Löscht die angegebene Ressource vom Server

HTTP Status Codes

1xx	Informationen
2xx	Erfolgreiche Operation
3xx	Umleitung
4xx	Client-Fehler
5xx	Server-Fehler

```
HTTP/1.1 404 Not Found
...
```

Kurzbeschreibung

Parameterübertragung

GET	Teil der URL
POST	Spezielle Anfrageart im HTTP Body

```ruby
require 'uri'
puts URI.escape("/&Ä")
#  => /&%C3%84
```

Reservierte Zeichen müssen URI-kodiert werden

GET-Parameter

Beginn Parameterliste: ?

Trennzeichen: &

```
GET /wiki/Spezial:Search?search=Katzen&go=Artikel HTTP/1.1
Host: de.wikipedia.org
```

Schlüssel

Wert

Vorteil: Parameter können mit Link weitergegeben werden

Nachteil: Datenmenge beschränkt

POST-Parameter

Kündigt Schlüssel/Wert-Paare im Body an

```
POST /wiki/Spezial:Search HTTP/1.1
Host: de.wikipedia.org
Content-Type: application/x-www-form-urlencoded
Content-Length: 24

search=Katzen&go=Artikel
```

Vorteil: Größe der Daten unbeschränkt

Nachteil: Link kann nicht weitergegeben werden

REST

Representational State Transfer

&

ROA

Resource-Oriented Architecture

Web Application Architecture

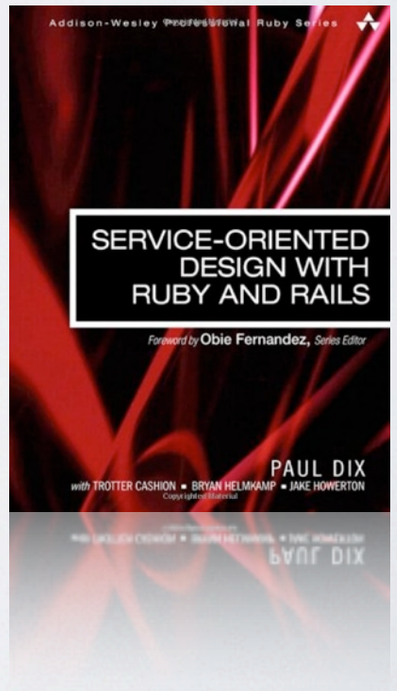
Quellen/Literatur

Fielding, Roy Thomas,
Architectural Styles and the Design of Network-based Software Architectures,
Doctoral dissertation,
University of California, Irvine, 2000

Quellen/Literatur

Richardson, L. and Ruby, S.,
RESTful Web Services
O'Reilly Media, 2007

Dix, P.,
Service-Oriented Design with Ruby on Rails,
Addison-Wesley Professional, 2010



REST

- “REST is an architectural style for distributed hypermedia systems.” - Roy Fielding, 2000

- Ziele:

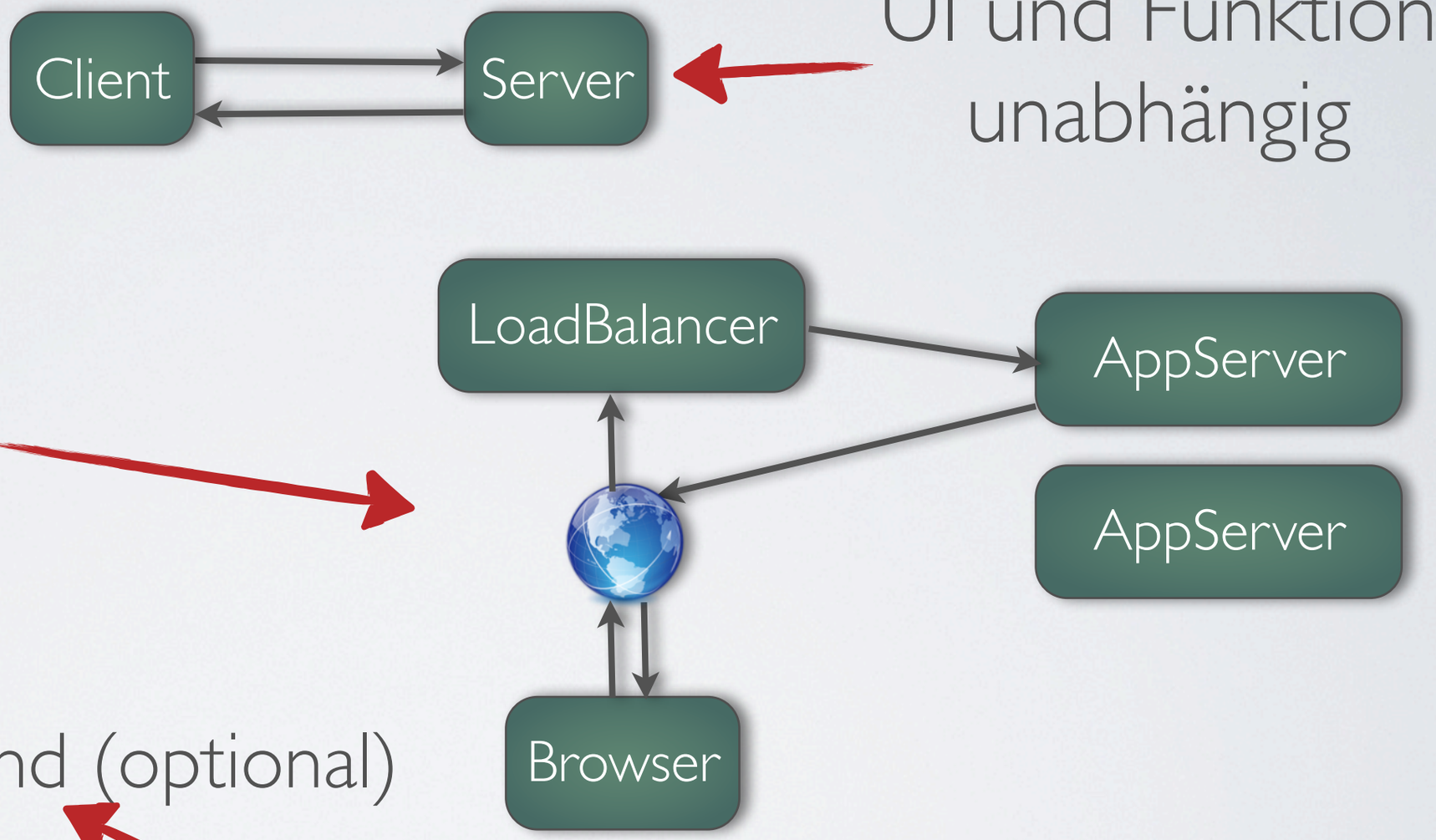
  - Skalierbare Interaktion zwischen Komponenten und größtmögliche Unabhängigkeit

  - Allgemeine Schnittstellen

  - Zwischenschichten zur Verbesserung von Sicherheit, Latenz und dem Kapseln von Legacy-Systemen

REST-Constraints

- Client/Server
- Statelessness
- Layered System
- Cacheable
- Code on Demand (optional)
- Uniform Interface



UI und Funktion unabhängig

z.B. JavaScript nachladen

ROA

- "The ROA is a way of turning a problem into a RESTful Webservice." - L. Richardson, S. Ruby, 2007
- Rückbesinnung auf Ideen, die das Web erfolgreich gemacht haben
- Adressability & URIs
- Resources & Representations
- Statelessness
- Connectedness
- Uniform Interface

Was sind Ressourcen?

- Dinge, die als eigene Entität referenziert oder manipuliert werden können
- Beispiele:
  - Version 1.3.1 einer Software
  - Die aktuelle Version einer Software
  - Die nächste Primzahl nach 1024
  - Eine Liste von zu behebenden Bugs

Adressierbarkeit und URI

Uniform Resource Identifier

- Jede Ressource hat einen URI
- Der URI ist sowohl der Name als auch die Adresse einer Ressource

Verschiedene Ressourcen, gleiche Daten

- Die URI sollte die Ressource beschreiben:
  - http://www.example.com/software/releases/1.3.1.tar.gz
  - http://www.example.com/software/releases/latest.tar.gz
  - http://www.example.com/nextprime/1024
  - http://www.example.com/bugs/by-state/open

Zustandslosigkeit

- Jeder Request muss in völliger Isolation bearbeitet werden können und alle dafür nötigen Informationen mitliefern
- Der Server speichert keinen Client-Zustand, sondern nur Ressourcen-Zustände
- Die vom Server durchgeführte Aktion hängt nicht vom Client-Zustand ab
- Vorteile:
  - Caching und Load Balancing vereinfachen sich
  - Der Zurück-Button funktioniert

Repräsentationen

Teilmengen von Ressourcen-Zuständen

Manipulation von Ressourcen durch Austausch von Repräsentationen

```
<person>
  <name = “Nils Haldenwang” />
  <description>...</description>
</person>
```

Einheitliche Schnittstelle

"Was soll mit wem getan werden?"

```
GET /info.txt HTTP/1.1
Host: example.com
```

Zuordnung einer sinnvollen und einheitlichen Semantik zu den HTTP Verben

Semantik der HTTP Verben

GET	Anfordern der Repräsenation einer Ressource
PUT	Ersetzt existierende Ressource oder legt neue an übergebener URI an
POST	Annotation existenter Ressourcen, Übergabe von Daten an berechnende Prozesse und Anhängen an vorhandene Collections
DELETE	Löscht die angegebene Ressource vom Server

HTTP Verben in Rails/ROA

URI	GET	PUT	POST	DELETE
/users	Liste aller Nutzer	-	Nutzer anlegen	-
/users/3	Repräsenta-tion von Nutzer 3	Update von Nutzer 3	-	Nutzer 3 löschen

Formatwahl: http://www.example.com/users/3.xml

Anhängen des Repräsentationsformates

Sicherheit und Idempotenz

Ressource ändert sich nicht

Wiederholte Anwedung ändert nichts

$42 \times 0 = 42 \times 0 \times 0 \times 0 = 0$

Verb	sicher	idempotent
GET	✓	✓
PUT	✗	✓
POST	✗	✗
DELETE	✗	✓

Vorteil:
Zuverlässige Anfragen über unzuverlässiges Netzwerk

Ruby on Rails

Ruby on Rails

- In Ruby geschriebenes Webapplication Framework
- Forciert Resource-Oriented Architecture
- Erstes Framework seiner Art
- Open Source
- 2005 Veröffentlichung Version 1.0
- Aktuelle Version: 3.2, erschienen im Januar 2012

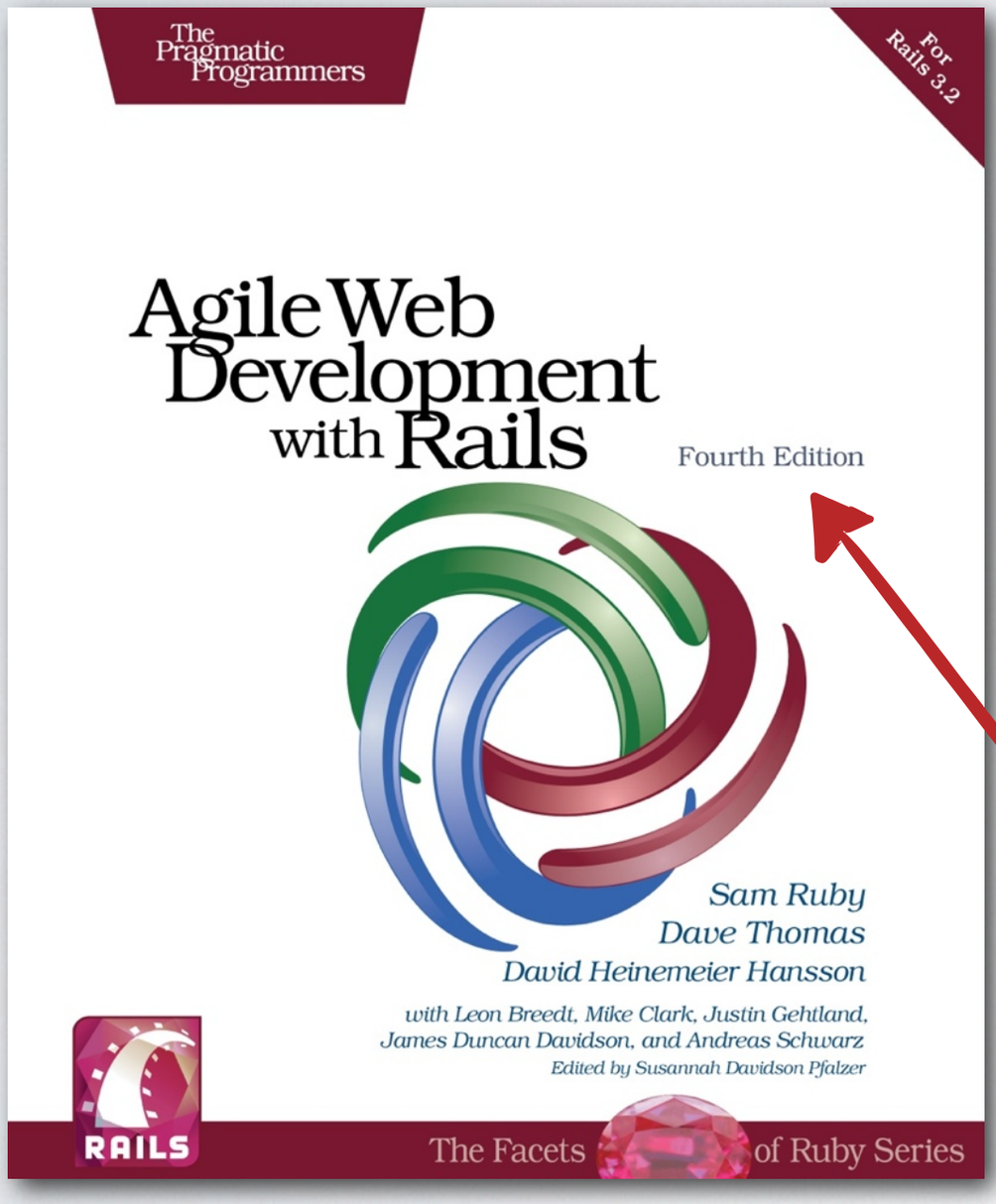
Ruby on Rails

Rails is about allowing beautiful code to solve the problems most people have most of the time in web-application development. It's about taking the pain away and making you happy.

This might all sound mighty fluffy, but only until you recognize that the single-most important factor in programmer productivity is motivation. And, happy programmers are certainly motivated programmers. Thus, if you optimize for happiness, you're optimizing for motivation, which ultimately leads to an optimization for productivity.

- David Heinemeier Hansson

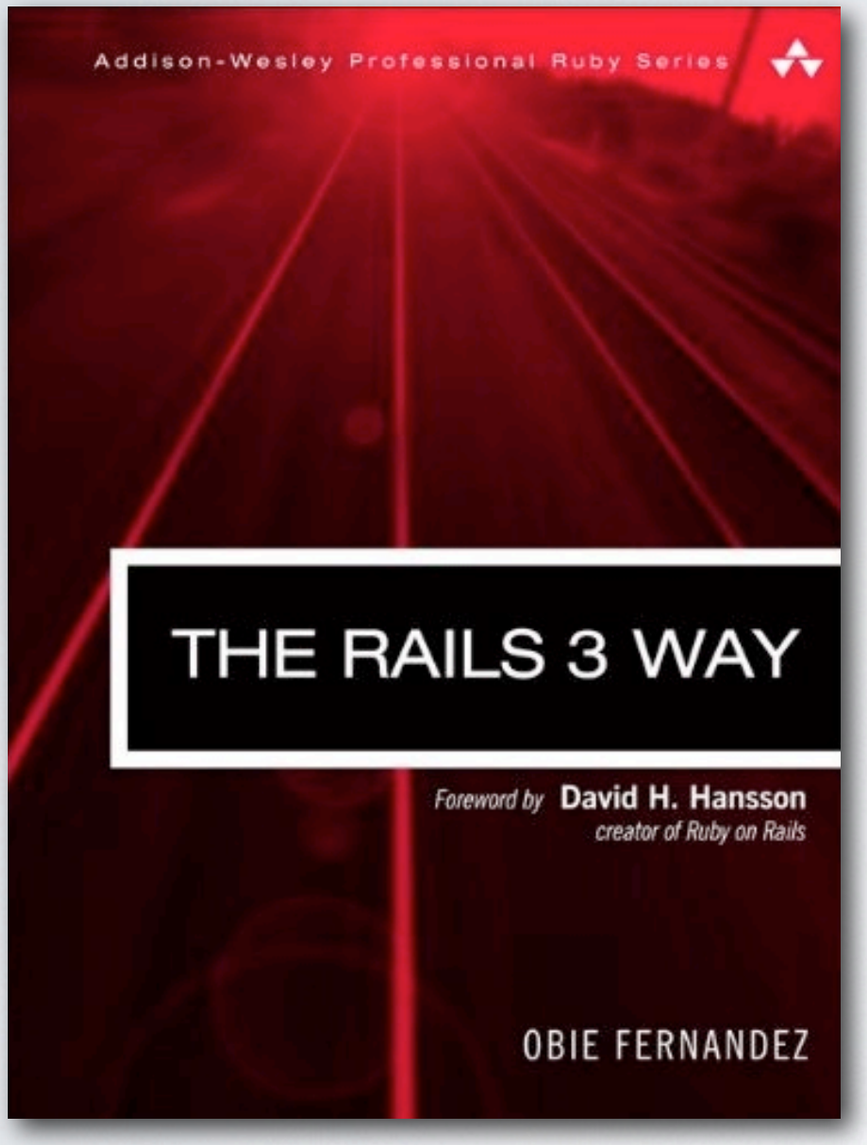
Quellen/Literatur



Sam Ruby, Dave Thomas, David Heinmeier Hansson, *Agile Web Development with Rails*, Fourth Edition Pragmatic Bookshelf, 2011

< 4 ist veraltet: Rails 2

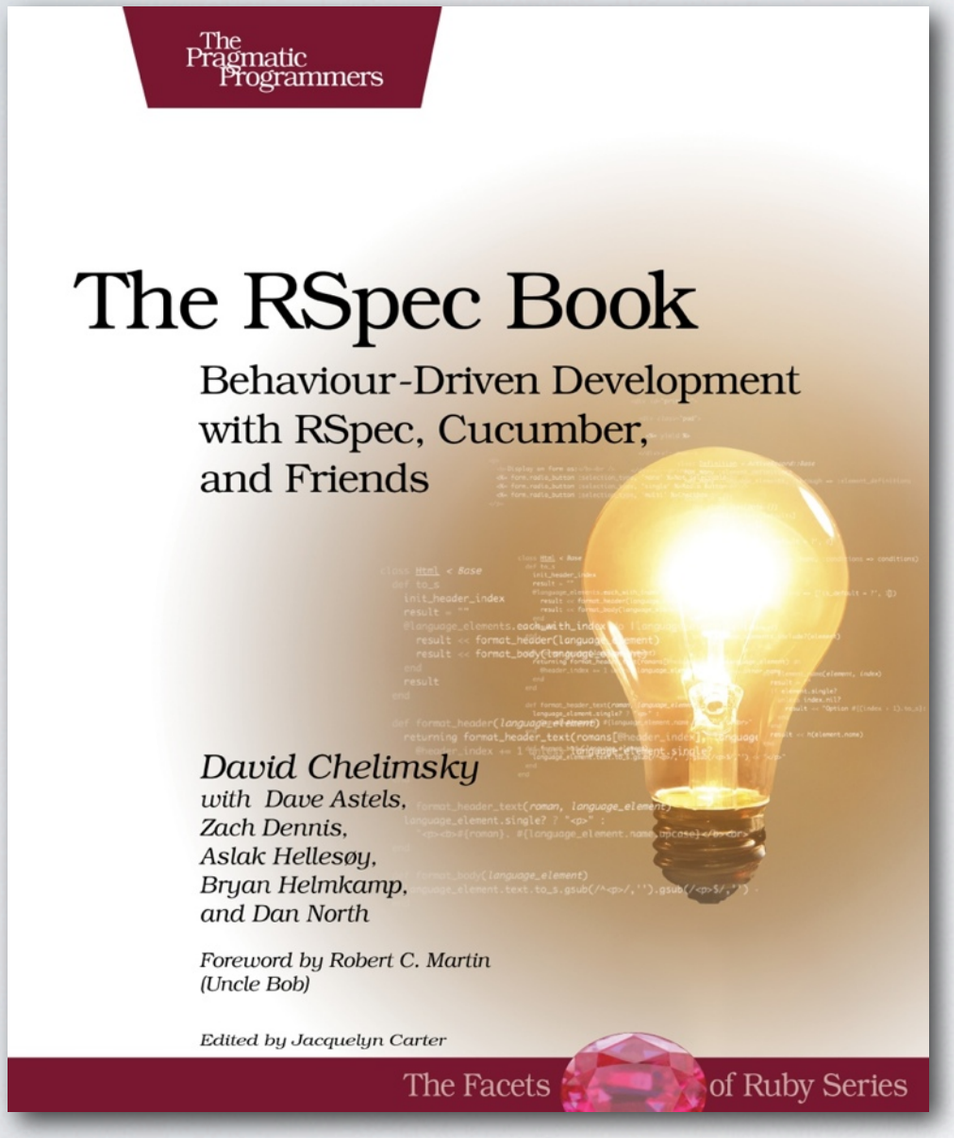
Quellen/Literatur



Obie Fernandez,
The Rails 3 Way,
Addison-Wesley Professional,
2nd Edition, 2010

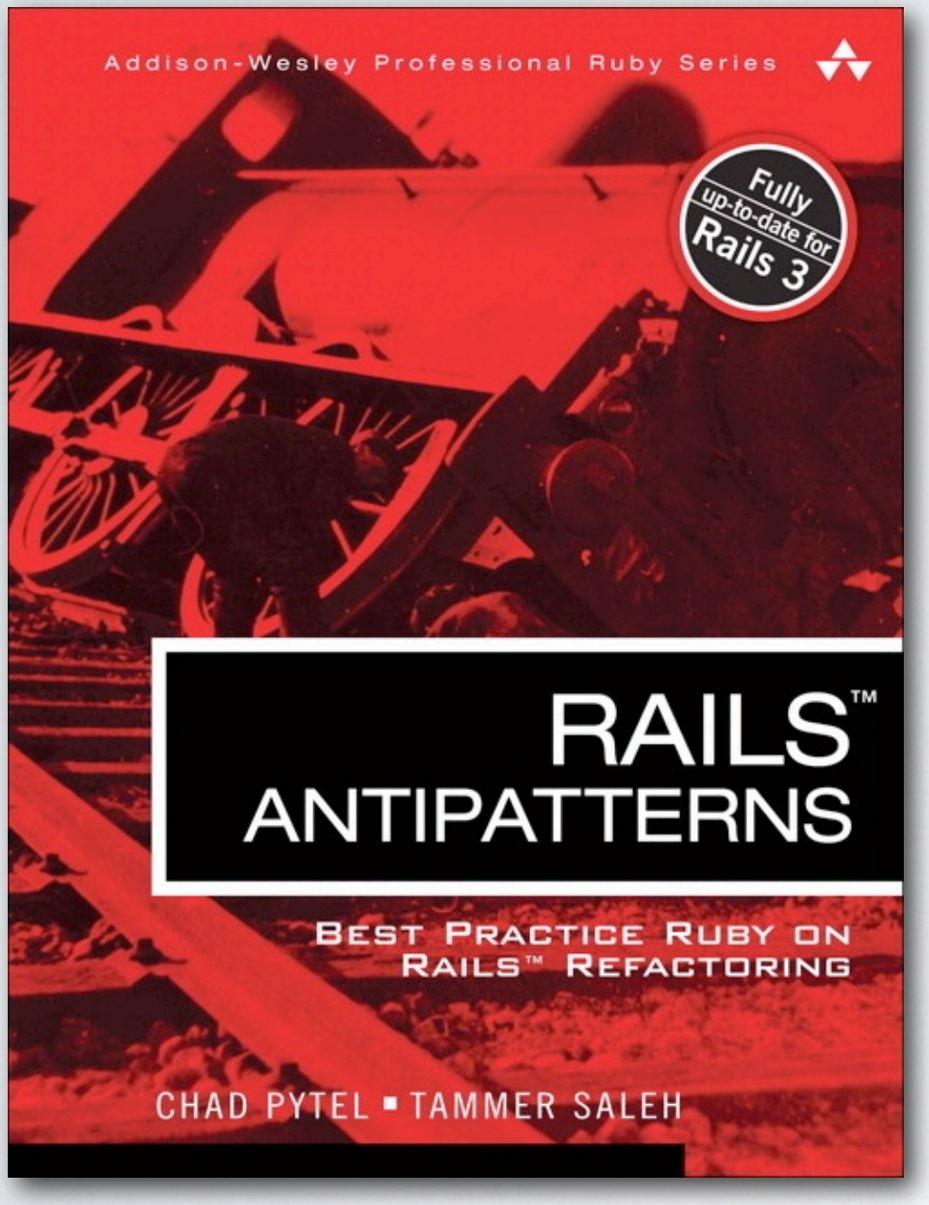
Erste Edition ist veraltet: Rails 2

Quellen/Literatur



Chelimsky, D. und Astels, D. und Dennis, Z. und Hellesoy, A. and Helmkamp, B., *The RSpec Book*, Pragmatic Bookshelf, 2010

Quellen/Literatur



Chad Pytel, Tammer Saleh

Rails Antipatterns - Best practice Ruby on Rails Refactoring,

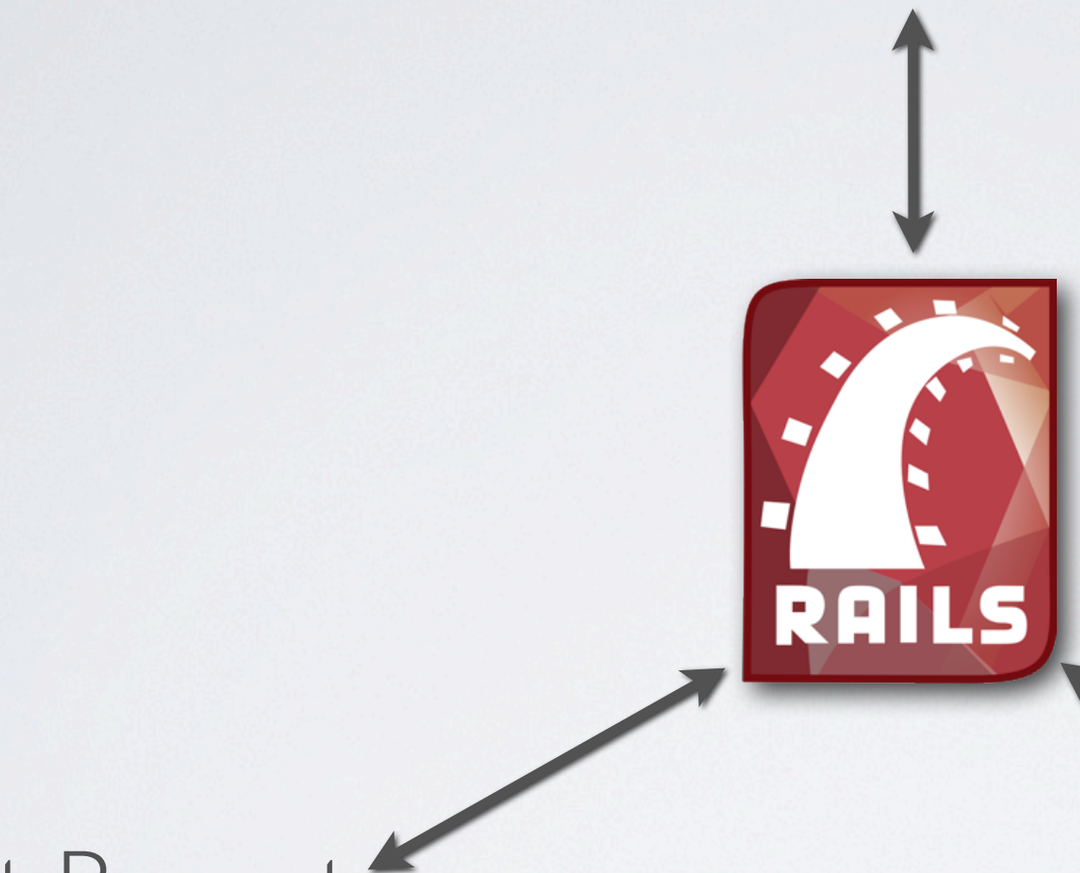
Addison-Wesley Professional, 2010

Ruby on Rails

Konzepte & Design Patterns

Ruby on Rails

Model-View-Controller (MVC)

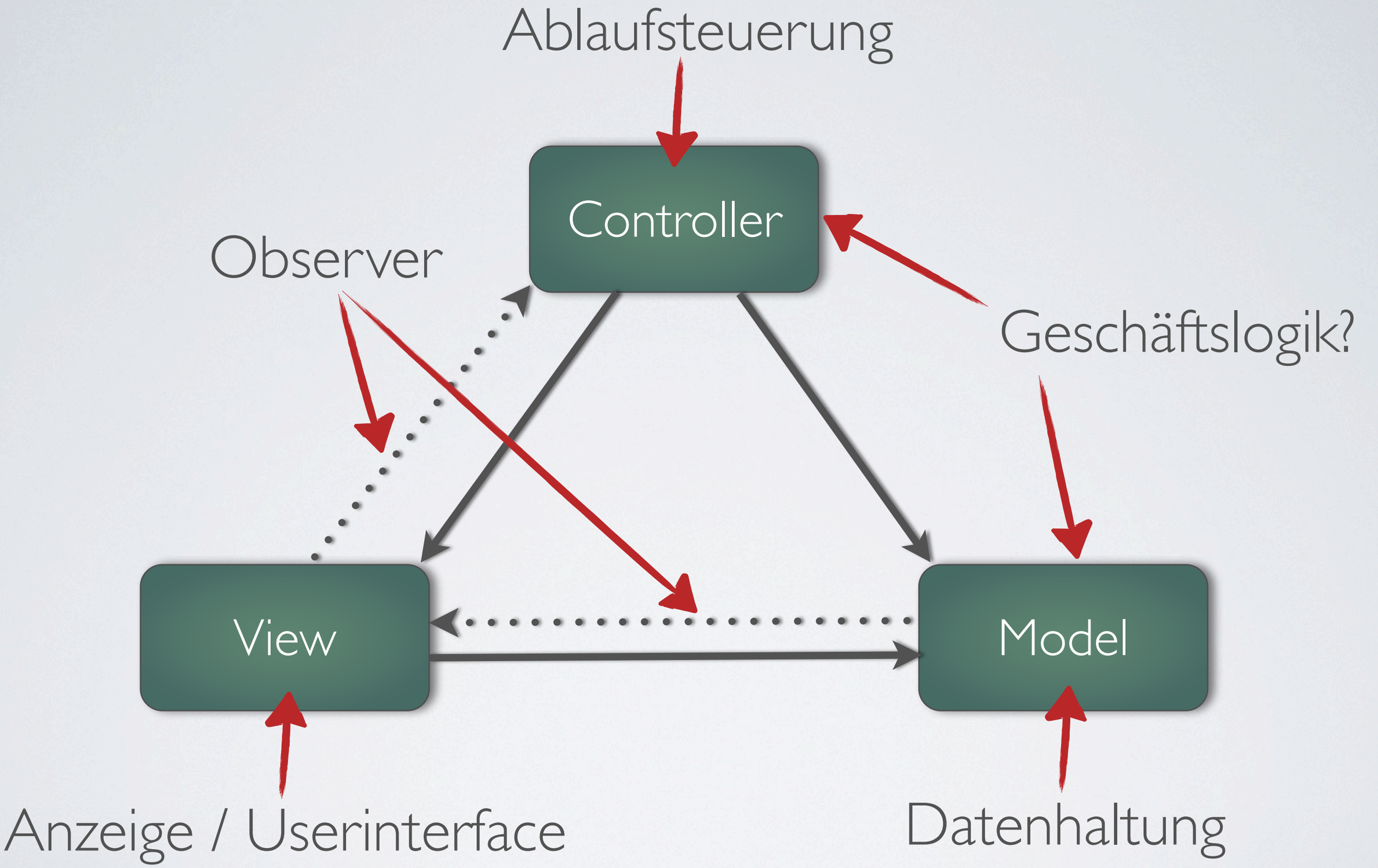


Don't Repeat Yourself (DRY)

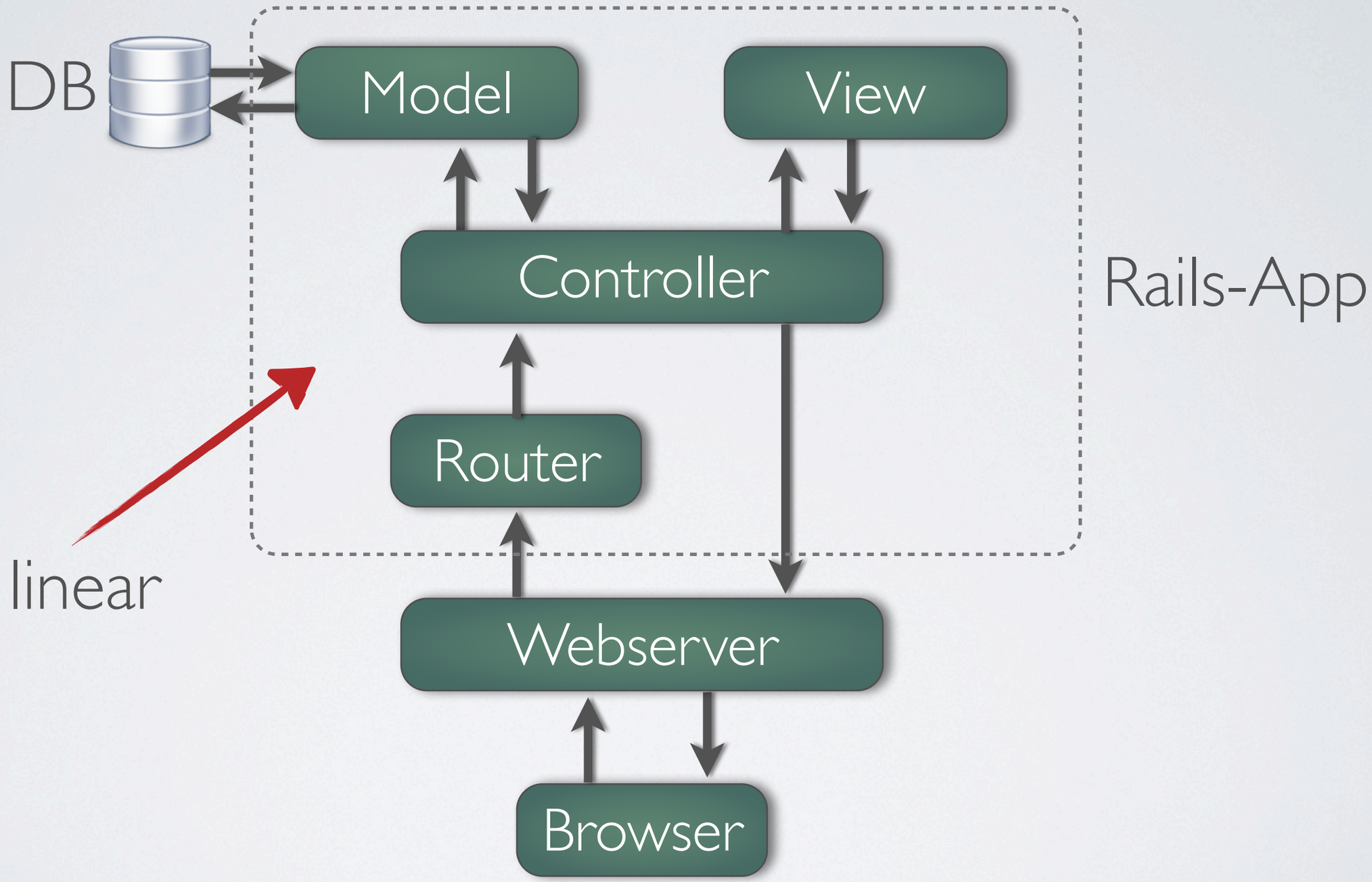
Convention Over Configuration (COC)

Model-View-Controller (MVC)

- Design Pattern zur Strukturierung von Software-Entwicklung
- Entwicklung 1979 für Oberflächen mit Smalltalk
- De-facto-Standard für Grobentwurf vieler komplexer Softwaresysteme
- Ziel: Reduktion der Komplexität und verbesserte Wiederverwendbarkeit, Wartbarkeit, Flexibilität
- Idee: Isolation von Geschäftslogik und Userinterface und Separierung in lose gekoppelte Komponenten


Ablaufsteuerung
Controller
Observer
Geschäftslogik?
View
Model
Anzeige / Userinterface
Datenhaltung

MVC in Rails



Ablauf linear

Relationale Datenbanken

Konvention: pluralisierter snake_case

Konvention: snake_case

Tabelle: users

id	first_name	last_name	age
1	Nils	Haldenwang	24
2	Luke	Skywalker	42

```
SELECT first_name,
       last_name,
       id, age
FROM users
WHERE age > 40
```

```
INSERT INTO users (first_name, last_name, age)
  VALUES ('Darth', 'Vader', 73)
```

Model: ORM

Design Pattern: Active Record

Object Relational Mapper

Tabelle: users

Klasse: User

id	first_name	last_name	age
1	Nils	Haldenwang	24
2	Luke	Skywalker	42

Instanzen von User

Idee: Repräsentiere die Tabelle durch eine Klasse und die Zeilen durch einzelne Instanzen.

Model: User

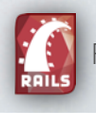
app/models/user.rb

DRY

```ruby
class User < ActiveRecord::Base
end

u = User.new first_name: "Obi-Wan", last_name: "Kenobi"
u.persisted? # => false
u.save
u.persisted? #  => true
puts "#{u.first_name}, #{u.last_name}"
#  => Obi-Wan Kenobi

obi_wan = User.find_by_first_name("Obi-Wan")
puts obi_wan.last_name #  => Kenobi
```



Ruby on Rails

Controller: UsersController

GET /users/42

```
class UsersController < ApplicationController
  def show
    @user = User.find(params[:id])
  end
end
```

app/controllers/users_controller.rb

View: Embedded Ruby (ERB)

app/views/users/show.html.erb

```
<div id='user-<%= @user.id %>'>
  <h2>
    <%= @user.first_name %>
    <%= @user.last_name %>
  </h2>
  <p>Age: <%= @user.age %></p>
</div>
```