Agile Webentwicklung mit Ruby on Rails

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

UNIVERSITÄT OSNABRÜCK

Institut für Informatik
AG Medieninformatik

Ruby on Rails

Views & Controller

Ruby on Rails

Asset Pipeline

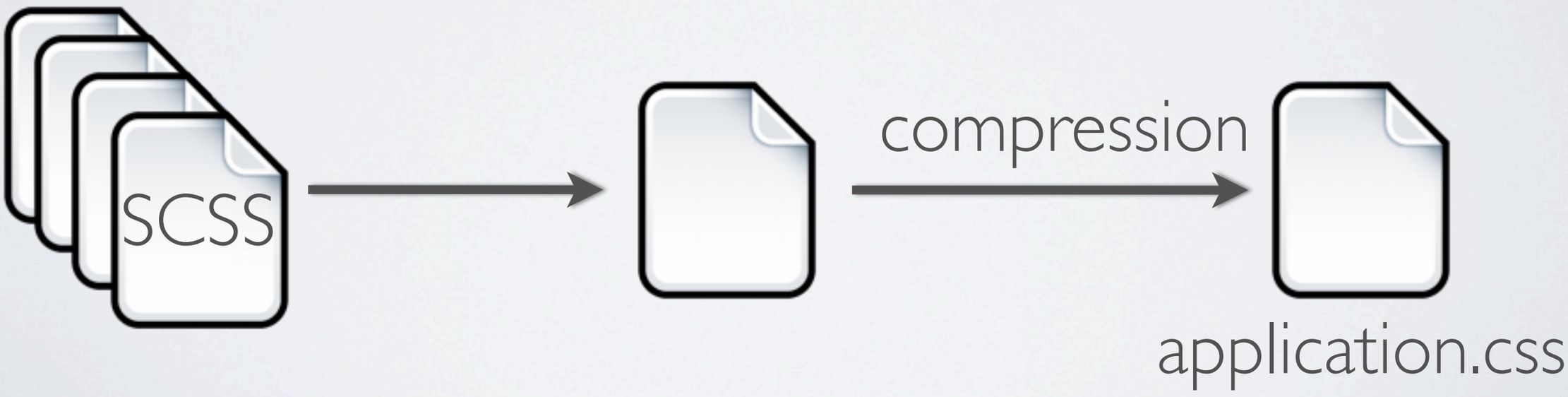
Views & Controller

Ruby on Rails

Asset Pipeline

"The asset pipeline provides a framework to concatenate and minify or compress JavaScript and CSS assets. It also adds the ability to write these assets in other languages such as CoffeeScript, Sass and ERB."

http://guides.rubyonrails.org/asset_pipeline.html



Pfade für Assets

Suchpfade

```
app/assets
lib/assets
vendor/assets
```

Application Assets

Library/Shared Assets

CSS/JS-Framework Assets

Pfad hinzufügen

```ruby
# config/application.rb

flash_path = Rails.root.join("app", "assets", "flash")
# => #{app_root}/app/assets/flash
config.assets.paths << flash_path
```

Assets einbinden

```
app/assets/javascript/home.css
lib/assets/javascript/slider.css
```

```
//= require home
//= require slider
```

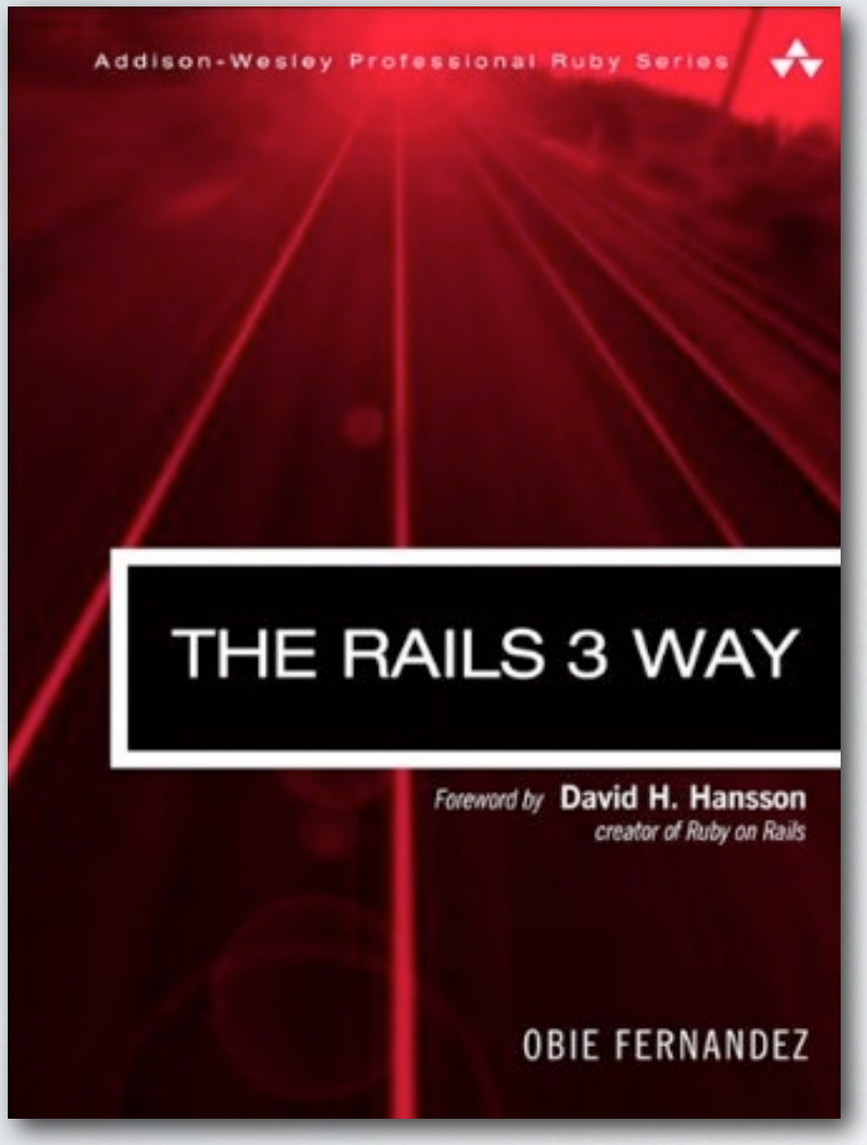
Manifest-File
z.B. application.css

```
//= require_self
//= require_tree .
```

Default

Rendern im Controller

Quellen/Literatur



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

S. 92 ff.

Rendern von Views

```
class EventsController < ActionController::Base
  def new
    # renders app/views/events/new.html.erb
  end

  def create
    @event = Event.new(params[:event])
    if @event.save
      flash[:notice] = "Event created!"
      redirect_to @event
    else
      render action: "new"
      # does NOT execute the new action
    end
  end
end
```

Explizites Rendern

```
render '/products/index.html.erb'
render 'products/index.html.erb'
render 'products/index.html'
render 'products/index'
render 'index'
render :index
```

Relativ zum View-Ordner des aktuellen Controllers

Richtlinie: So viel wie nötig, so wenig wie möglich

Weitere Beispiele

```
render text: "Submission accepted",
        content_type: "text/plain"

render json: @article, include: :author

render xml: @article

render nothing: true, status: 401 # Unauthorized

render :index, layout: false

render :show, layout: 'another_layout'
```

View Partials

View Partials

Idee: Auslagerung mehrfach verwendeter Teil-Views

```
# app/views/person/_person.html.erb
<h1><%= name %></h1>
```

```
# app/views/person/show.html.erb
<%= render partial: "person",
      locals: { name: @person.name } %>
Age: <%= @person.age %>
```

Beispiel

```
<%# app/views/users/show.html.erb %>
<% @user.articles.each do |article| %>
  <div>
    <h2><%= link_to article.title, article %></h2>
    <p><%= article.created_at %></p>
    <p class='lead'><%= article.summary %></p>
  </div>
<% end %>
```

```
<%# app/views/categories/show.html.erb %>
<% @category.articles.each do |article| %>
  <div>
    <h2><%= link_to article.title, article %></h2>
    <p><%= article.created_at %></p>
    <p class='lead'><%= article.summary %></p>
  </div>
<% end %>
```

Beispiel

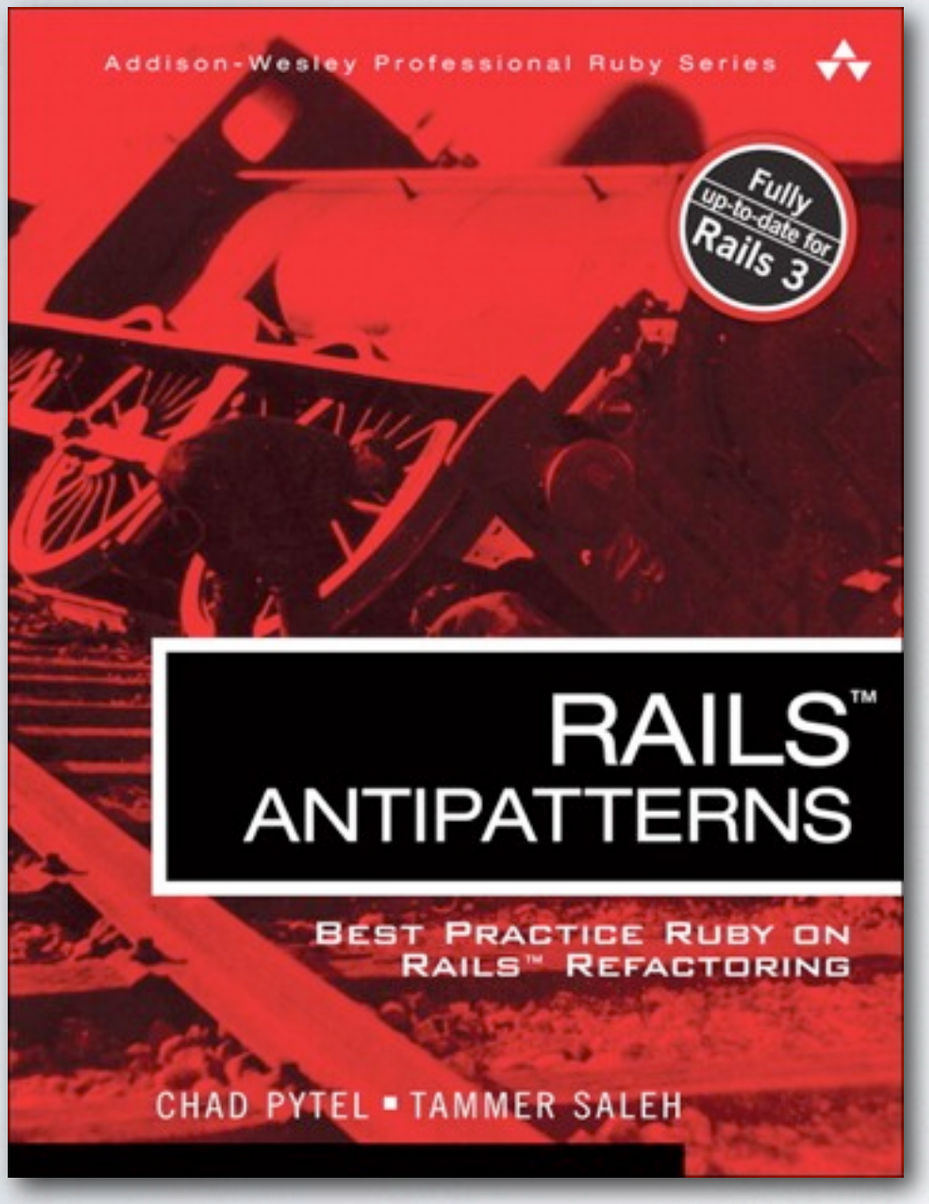
```
<%# app/views/shared/_article.html.erb %>
<div>
  <h2><%= link_to article.title, article %></h2>
  <p><%= article.created_at %></p>
  <p class='lead'><%= article.summary %></p>
</div>
```

```
<%# app/views/users/show.html.erb %>
<%= render partial: "shared/article",
           collection: @user.articles %>
```

```
<%# app/views/categories/show.html.erb %>
<%= render partial: "shared/article",
           collection: @category.articles %>
```

The Law of Demeter

Quellen/Literatur



Chad Pytel, Tammer Saleh
Rails Antipatterns - Best practice Ruby on Rails Refactoring,
Addison-Wesley Professional, 2010

S. 2 ff.

The Law of Demeter

Auch bekannt als: *Principle of Least Knowledge*

- Jede Komponente sollte nur limitiertes Wissen über andere Komponenten haben und zwar nur über solche, zu denen sie in enger Beziehung steht

- Jede Komponente kommuniziert nur mit den eng zu ihr in Beziehungen stehenden anderen Komponenten

⇒ So wenig Annahmen wie möglich über

die Struktur anderer Komponenten machen!

Beispiel

```
class User < ActiveRecord::Base
  attr_accessible :first_name, :last_name, :middle_name
  has_one :address
end

class Address < ActiveRecord::Base
  attr_accessible :street, :city, :state, :zip_code
  belongs_to :user
end
```

Beispiel: User-Show-View

```
<h1><%= "#{@user.first_name} #{@user.last_name}" %></h1>
<p>
  Street: <%= @user.address.street %><br />
  City: <%= @user.address.city %><br />
  State: <%= @user.address.state %><br />
  Zip-Code: <%= @user.address.zip_code %>
</p>
```

Law of Demeter
verletzt

Merkregel: Nie mehr als einen Punkt verwenden!

Lösung: Methoden in User

```ruby
class User < ActiveRecord::Base
  # ...
  def address_street
    address.street
  end
  # ...
end
```

```erb
<h1><%= "#{@user.first_name} #{@user.last_name}" %></h1>
<p>
  Street: <%= @user.address_street %><br />
  City: <%= @user.address_city %><br />
  State: <%= @user.address_state %><br />
  Zip-Code: <%= @user.address_zip_code %>
</p>
```

Bessere Lösung: Delegation

```
class User < ActiveRecord::Base
  attr_accessible :first_name, :last_name, :middle_name
  has_one :address

  delegate :city, :street, :state, :zip_code,
           to: :address,
           prefix: true
end
```

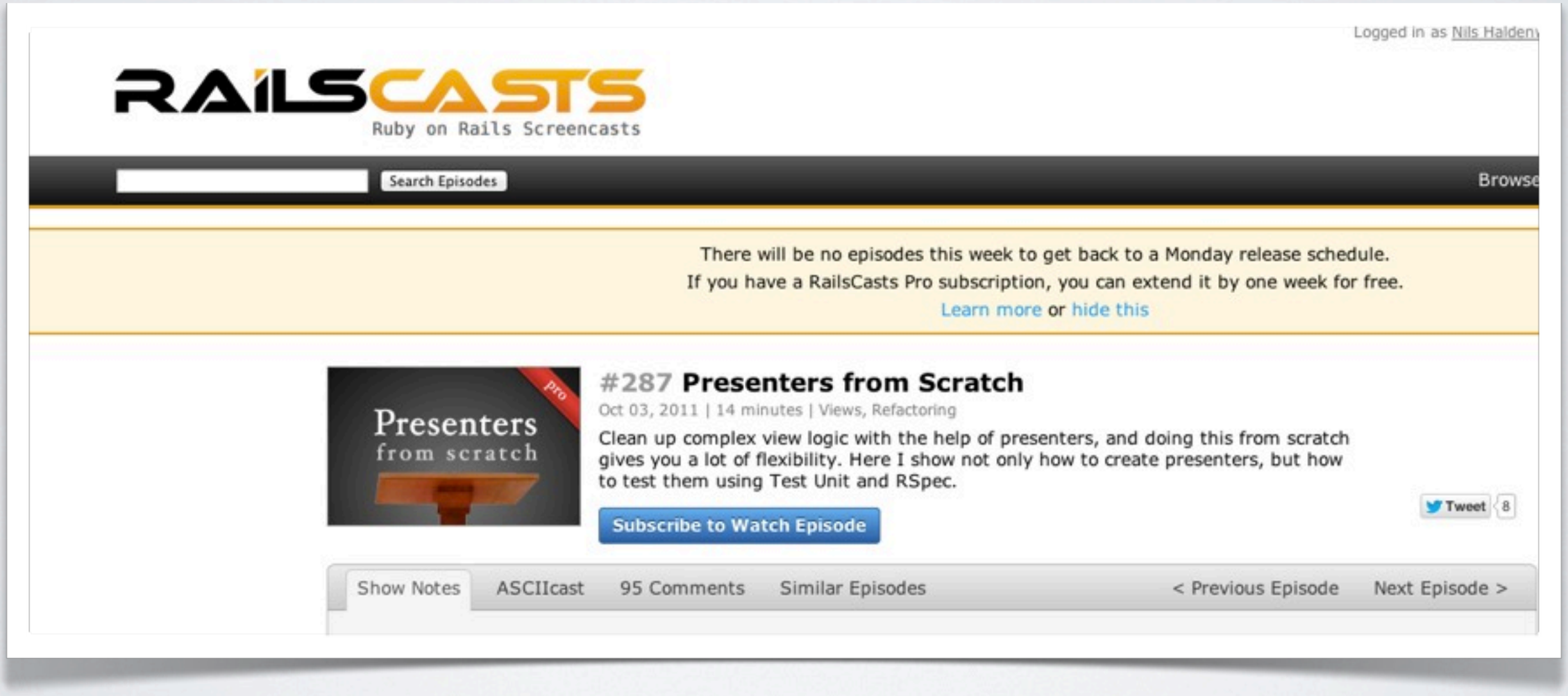
Design Pattern: Presenter

Views & Controller

Ruby on Rails

Quellen/Literatur

http://railscasts.com/episodes/287-presenters-from-scratch



Warum Presenter?

Probleme:

- Konsequente Einhaltung des Law of Demeter führt zu viel Code im Model, der eigentlicht zur Unterstützung der Darstellung dient

- Optionale Attribute führen oft zu komplizierter Logik in Views, wodurch diese leicht unübersichtlich werden

Idee:

Kapselung in eigene Klasse, die sowohl das Model als auch die View kennt

Beispiel: Optionale URL

```
<h1>
  <% if user.url.present? %>
    <%= link_to "#{@user.first_name} #{@user.last_name}",
                user.url %>
  <% else %>
    <%= "#{@user.first_name} #{@user.last_name}" %>
  <% end %>
</h1>

<p>
  Street: <%= @user.address.street %><br />
  City: <%= @user.address.city %><br />
  State: <%= @user.address.state %><br />
  Zip-Code: <%= @user.address.zip_code %>
</p>
```

```ruby
# app/presenters/user_presenter.rb
class UserPresenter
  attr_reader :address, :user

  delegates :city, :state, :street, :zip_code,
            to: :address

  def initialize(user, template)
    @user, @template = user, template
    @address = user.address
  end

  def method_missing(*args, &block)
    @template.send(*args, &block)
  end

  def linked_name
    link_to_if user.url.present?,
      "#{user.first_name} #{user.last_name}", user.url
  end
end
```

View-Helper

```ruby
# app/helpers/application_helper.rb
module ApplicationHelper
  def present(object, klass=nil)
    klass ||= "#{object.class}Presenter".constantize
    presenter = klass.new(object, self)
    yield presenter if block_given?
    presenter
  end
end
```

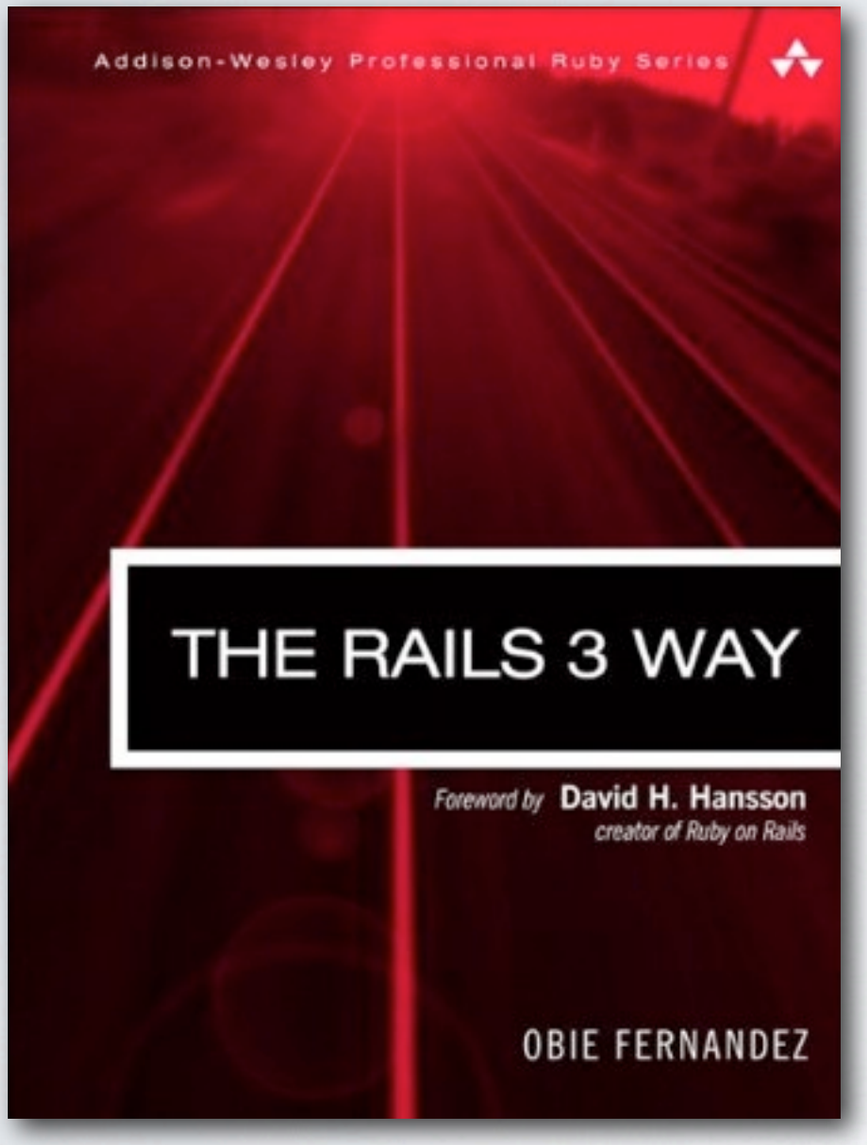
Aktuelle View

View mit Presenter

```
<% present @user do |user_presenter| %>
  <h1><%= user_presenter.linked_name %></h1>
  <p>
    Street: <%= user_presenter.street %><br />
    City: <%= user_presenter.city %><br />
    State: <%= user_presenter.state %><br />
    Zip-Code: <%= user_presenter.zip_code %>
  </p>
<% end %>
```

Controller Filter

Quellen/Literatur



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

S. 105 ff.

Einfacher Filter

Pre- und Postprocessing für mehrere Actions

```
class SomeController < ApplicationController
  before_filter :require_authentication

  # ...

  protected
  def require_authentication
    unless current_user.logged_in?
      flash[:error] = "Please log in."
      redirect_to new_session_url
    end
  end
end
```

Abbruch der Verarbeitung

Filter und Vererbung

```
class ApplicationController < ActionController::Base
  after_filter :compress # compress response
end

class BankController < ApplicationController
  before_filter :audit # log requests
  # ...
end

class VaultController < BankController
  skip_before_filter :audit
  before_filter :verify_credentials
end
```

Filter werden vererbt, wobei die Aufrufreihenfolge der Vererbungshierarchie entspricht.

Reihenfolge der Filter

```
class ShoppingController < ActionController::Base
  before_filter :verify_open_shop
end

class CheckoutController < ShoppingController
  prepend_before_filter :ensure_items_in_cart,
                        :ensure_items_in_stock
end
```

Filter-Chain für CheckoutController:
ensure_items_in_cart
ensure_items_in_stock
verify_open_shop

Around-Filter

```
class SomeController < ApplicationController
  around_filter :catch_exception

  protected
  def catch_exceptions
    yield
  rescue => exception
    # Pokemon-Exception-Handling not evil here
    logger.debug "Caught exception! #{exception}"
    raise
  end
end
```

Institut für Informatik
Prof. Dr. Oliver Vornberger
Nils Haldenwang, B.Sc.

Universität Osnabrück
http://www-lehre.inf.uos.de/~ror
14.06.2012

Agile Webentwicklung mit Ruby on Rails

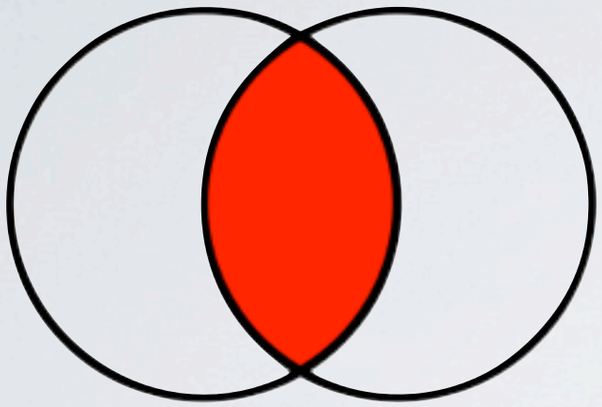
Sommersemester 2012

Blatt 5

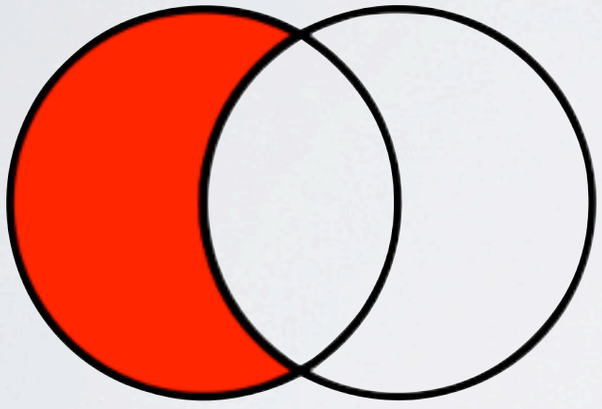
Im Ordner `blog` finden Sie die in der Vorlesung erzeugte Applikation *Blog*. Ziel dieses Blattes ist die Erweiterung dieser Applikation um einige Features. Alle Aufgaben sollten nacheinander in derselben Applikation bearbeitet werden.

Demo

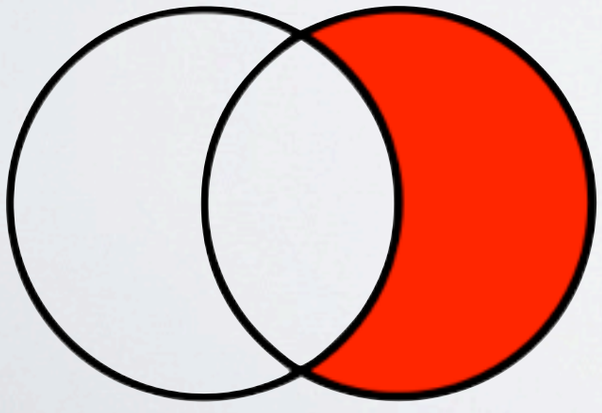
Tag-Mengen



Alt ∩ Neu: Nichts zu tun



Alt \ Neu: Beziehungen entfernen



Neu \ Alt: Beziehungen hinzufügen

Demo