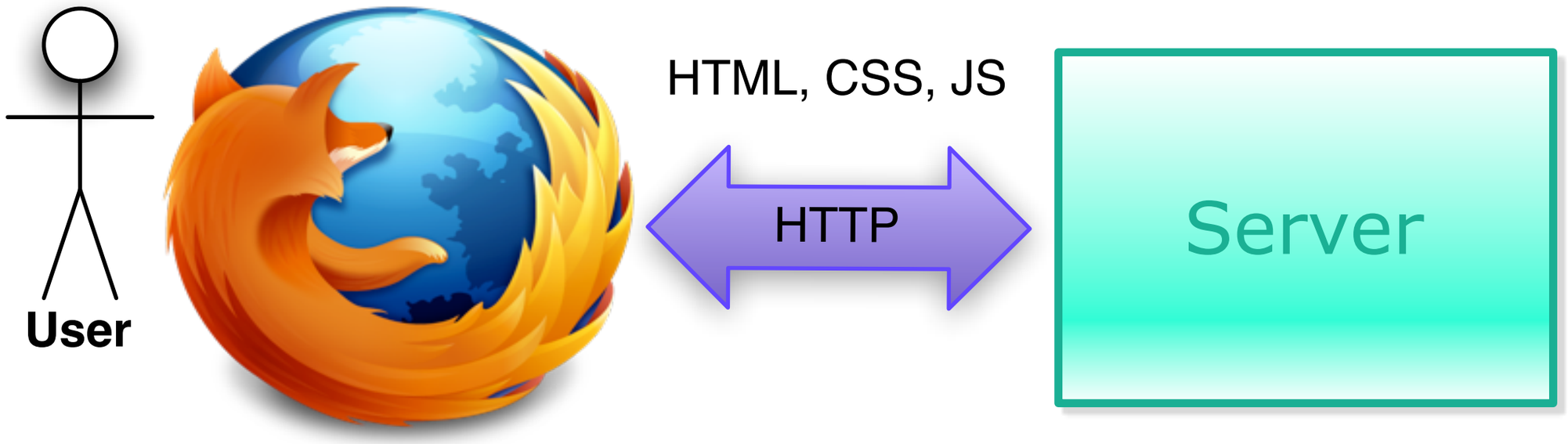
Einführung in Rails

Prof. Dr.-Ing. Carsten Bormann cabo@tzi.org

Was ist eine Web-Anwendung?

- Client: Browser, zeigt HTML/CSS/JavaScript an
- Server: Web-Anwendung, hält Zustand
- Verbindung: HTTP (Architekturprinzip: REST)



HTTP: Hypertext Transfer Protocol und URIs (Uniform Resource Identifiers)

- **Browser**
  - **hat URI http://www.informatik.uni-bremen.de/modulanmeldung**
  - **schickt an Web-Server per HTTP Anfrage**
- `GET /modulanmeldung`
  `Host: www.informatik.uni-bremen.de`
- **Web-Server schickt Antwort mit Body**
- `<html>....`
- **Browser zeigt Body an**
  - **darin sind:**
    Links (weitere URIs → GET),
    Formulare (mit URIs → POST)

HTTP und URIs, REST

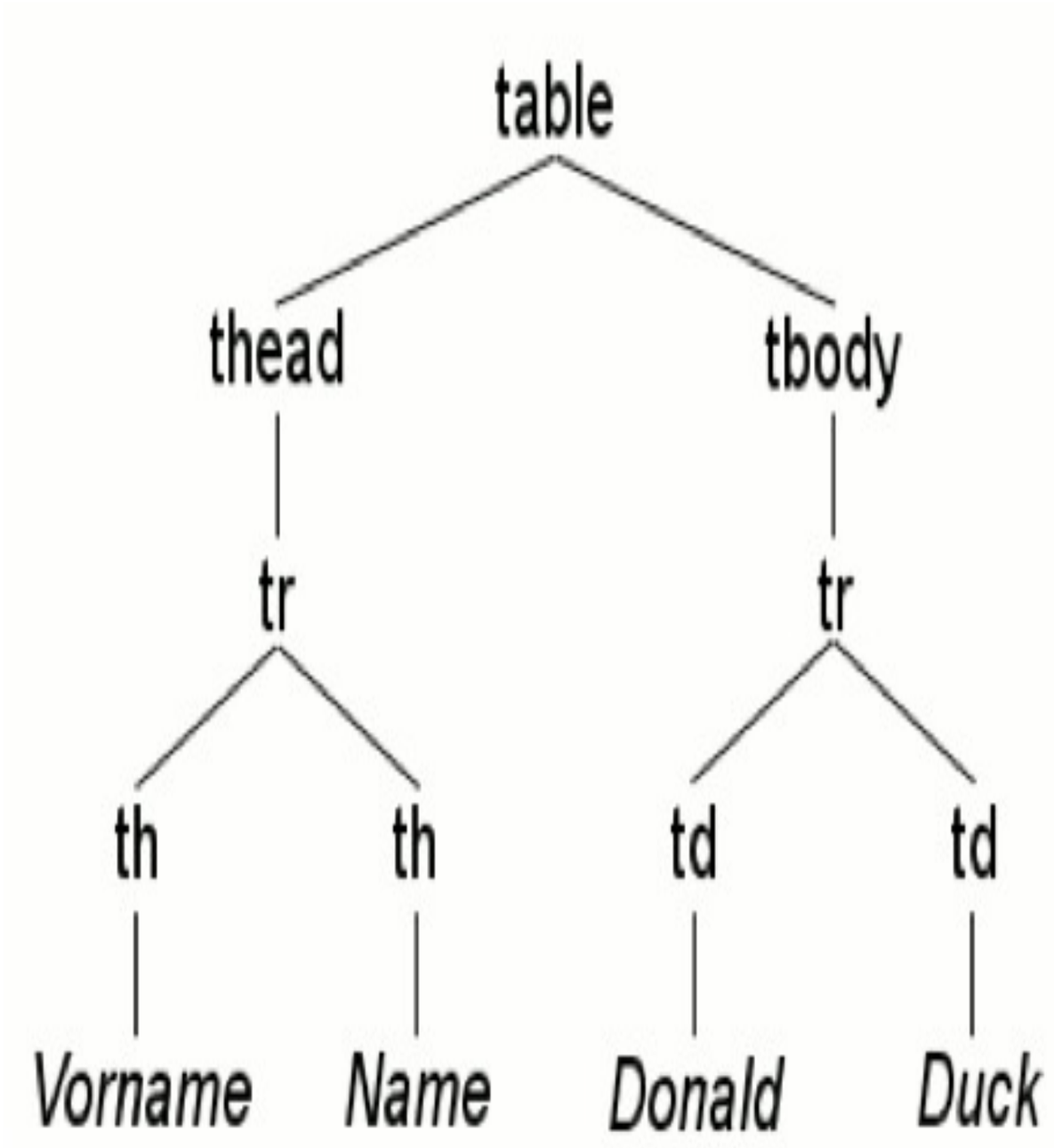
- **Hypermedia as the Engine of Application State (HATEOAS)**
  - Server steuert über Body, was für URIs (Links/Formulare) aus dem angezeigten UI verfolgt werden kann
- **GET: Anforderung weiterer Bodies**
  - URIs der verfolgten Links (<a href=...>)
  - aber auch eingebettete Bilder etc. (<img src=...>)
- **POST: Auslösen von Aktionen**
  - URI steht in Formular (<form action=...>)
  - Body wird bei Anfrage mitgeschickt (Formularinhalt)
- **Ausserdem: PUT, DELETE (nicht aus Browser)**
- **(vgl.: CRUD: Create, Read, Update, Delete)**

HTML: Hypertext Markup Language

- **Statischer Inhalt der Webseite**
  - **Text, evtl. geeignet strukturiert**
    `<body><div class="header">...</div><p>...</p></body>`
  - **eingebette Verweise auf Bilder etc.**
    `<img src="..."/>`
- **Links, Formulare (HATEOAS)**
  - `<a href="http://www.tzi.de">TZI</a>` ➔ **GET**
  - `<form action="http://.../anmeldungen">`
    `<input.../></form>` ➔ **POST**
- **Verweise auf CSS (Stil), JavaScript (Interaktion)**
- **Basis: SGML (bzw. XML in XHTML)**
- **Document Object Model (DOM):**
  - **abstrakte Struktur des HTML-Dokuments**

DOM: Document Object Model

```
<table>
  <thead>
    <tr>
      <th>Vorname</th>
      <th>Name</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Donald</td>
      <td>Duck</td>
    </tr>
  </tbody>
</table>
```



CSS

- **Cascading Style Sheets**
- **Haken sich in HTML-DOM ein**
  - **Selektor: Welche Dom-Elemente sind betroffen**
    - **hakt in Elementname, Class-Attribut etc. ein**
  - **Eigenschaften: Was wird wie eingestellt**
    - `Eigenschaft: Wert`
  -
```
html {background-color: #eee}
body {font-size:75%;color:#222;background:#fff;
font-family:Optima, "Helvetica Neue", Arial,
Helvetica, sans-serif; width:450px; margin:0 auto;}
a {color:#06c; text-decoration:underline;}
.tiny {font-size:.5em; line-height:1.875em;}
```
- **Direkt in HTML (`<style...> ... </style>`)**
- **Referenz: *Stylesheet* (`<link rel="stylesheet" href="..." .../>`)**

JavaScript

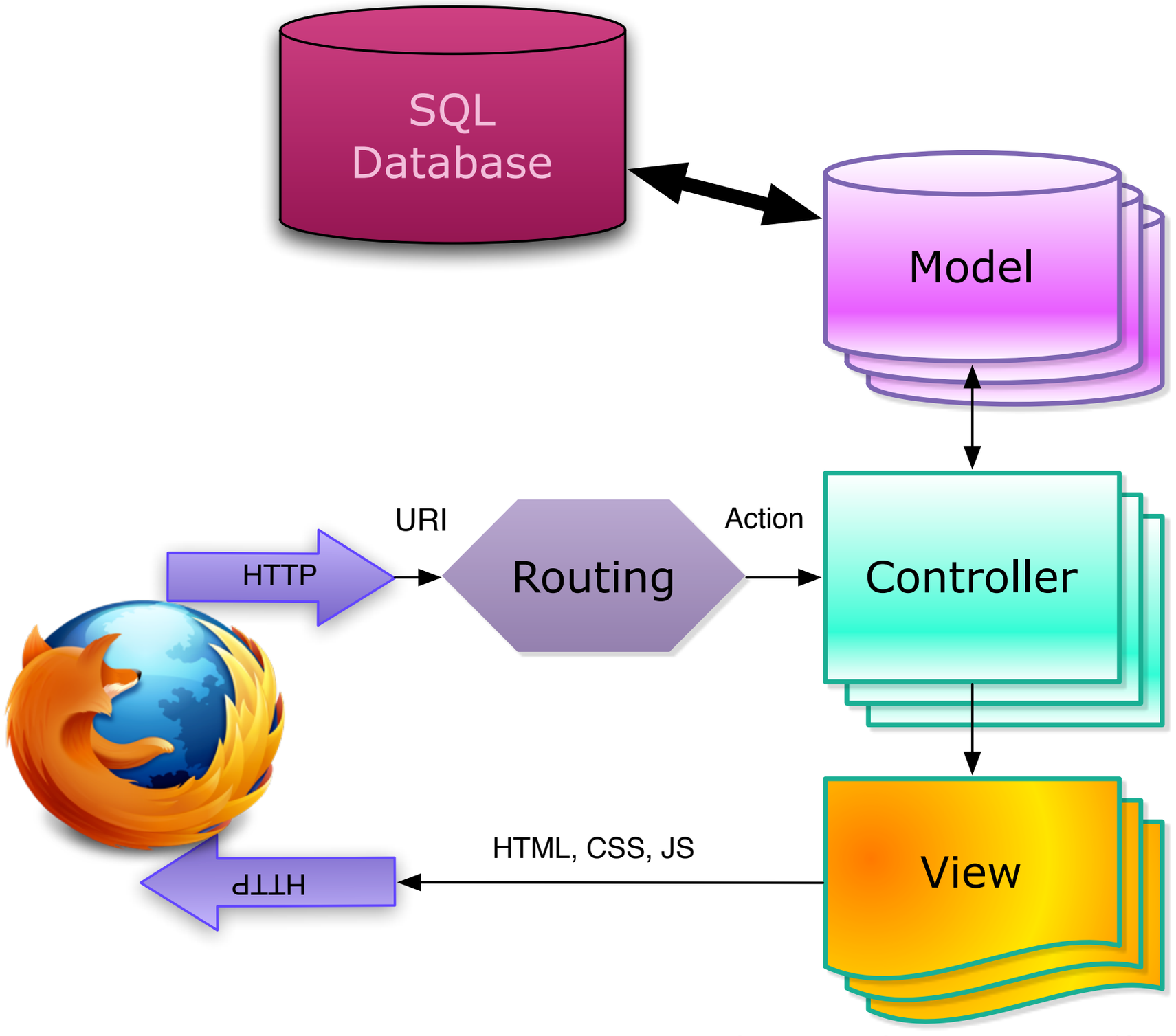
- **Dynamische Programmiersprache**
  - **eine der Big 5**
  - **besonderes Augenmerk auf Performance (V8, Jägermonkey, ...)**
- **Browser-spezifische Objekte**
  - `document`**: Document Object Model (DOM)**
  - `window`**: das aktuelle Fenster**
- **Direkt in HTML (**`<script...> ... </script>`**)**
  - **auch über Attribute wie** `onclick`
- **Referenz (**`<script src="..."/>`**)**

Server-Seite

- **Web-Server**
  - Apache
  - nginx
  - Application-Server wie mongrel, thin
- **CGI: Common Gateway Interface**
  - Parameter der Anfrage im Environment
  - HTTP-Header, Leerzeile, Body → stdout
- **Application-Server emulieren dies meist**
  - Ruby: Rack

Model-View-Controller (MVC)

- **Strukturmodell für interaktive Anwendungen (→Smalltalk)**
- **Model = Daten/Zustand (Datenbank)**
- **Controller = Interaktion**
- **View = Ansicht**



Ruby on Rails

- 2004 von David Heinemeier Hansson aus Basecamp (37signals) extrahiert
- Ruby + DSL + Metaprogramming
- MIT-Lizenz
- Weihnachten 2008 mit Merb (Yehuda Katz) zusammengeführt
- Rails 3.0

Rails: Komponenten

- **Models: Active-Record**
  - **ORM (object-relational mapper)**
    - **Programm denkt in Objekten; SQL-Datenbank denkt in Relationen**
  - **Pro Tabelle eine Klasse, kann pro Zeile ein Objekt anlegen**
  - **Repräsentiert meist die *Geschäftslogik***
- **Controller, Views, Routing: Action-Pack**
  - **Routing: Wandelt Anfrage (URI) in Controller/Action/Parameter**
  - **Controller: Instanz wird bei Anfrage angelegt, Action aufgerufen**
  - **View: kein reines Ruby, sondern ERB oder HAML**
    - **pro Controller/Aktion**
    - **Layout-Views**
- **Infrastruktur: Railties, Active-Support**
  - **Active-Support erweitert Ruby um nützliche Idiome**

Controller (show: GET-Aktion)

```
class ProductsController < ApplicationController
  ...
  def show
    @product = Product.find(params[:id])
    ...
  end
  ...
end
```


View

```
<p>
  <b>Name:</b>
  <%=h @product.name %>
</p>

<p>
  <b>Price:</b>
  <%=h @product.price %>
</p>

<%= link_to 'Edit', edit_product_path(@product) %> |
<%= link_to 'Back', products_path %>
```

(Dazu kommt ein Layout)

Controller (create: POST-Aktion)

```ruby
class ArticlesController < ApplicationController
  def create
    @article = Article.new(params[:article])
    if @article.save
      flash[:notice] = 'Article was successfully created.'
      redirect_to(@article)
    else
      render :action => "new"
    end
  end
end
```

Model

```ruby
class Order < ActiveRecord::Base
  has_many :order_items

  def add_product(product, amount)
    order_item = OrderItem.new(product, amount)
    self.price_total += product.price * amount
    order_items << order_item
  end
end
```

```ruby
ActiveRecord::Schema.define(:version => 20090216052716) do

  create_table "orders", :force => true do |t|
    t.float    "price_total"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

...
```

```
CREATE TABLE orders (
  id INTEGER PRIMARY KEY NOT NULL,
  order_date TIMESTAMP NOT NULL,
  price_total DOUBLE NOT NULL
);
```


Datenbankschema

- Aufbau über Migrationen
  - Hinzufügen, Löschen von Tabellen und Spalten
  - Evtl. Aktionen für Migration der Daten
- Entstehendes Schema wird als Ruby-DSL verwaltet

```ruby
ActiveRecord::Schema.define(:version => 2) do
  create_table "comments", :force => true do |t|
    t.text :body
    t.integer :post_id
  end
  create_table "posts", :force => true do |t|
    t.string :title, :author_name
    t.text :body
    t.integer :comments_count, :default => 0
    t.timestamps   # t.datetime :created_at, :updated_at
  end
end
```

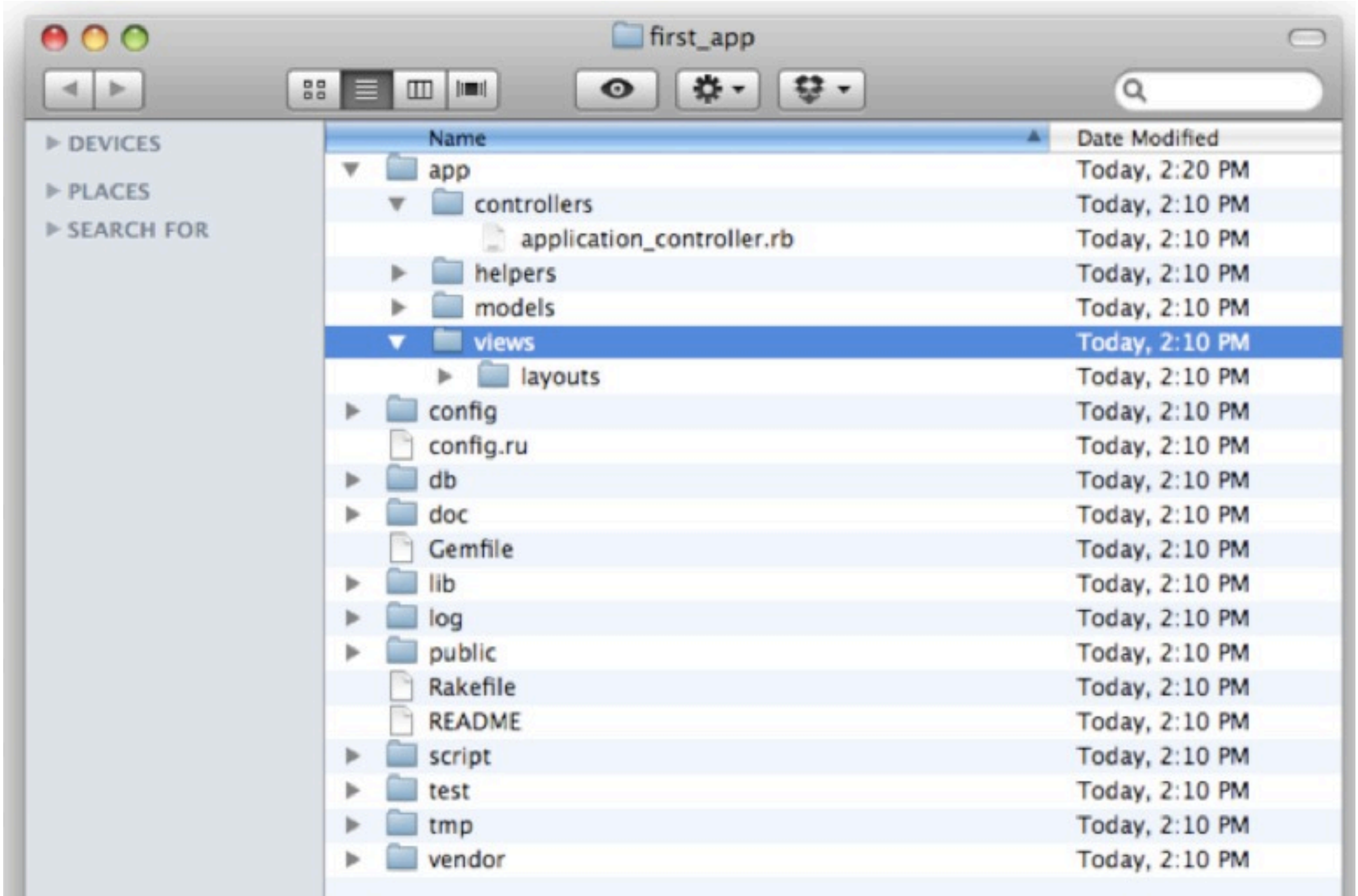

Konventionen

- **Viel wird von Rails einfach vorgegeben**
- **Standard-Action-Namen:**
  - **collection: index, create,**
    member: show, update, destroy,
    (collection: new, member: edit)
- **Standard-Routing,**
  URI-Helper
- **...**

Method	URL Path	Controller	Action
GET	/groups/	"groups"	"index"
POST	/groups/	"groups"	"create"
GET	/groups/new/	"groups"	"new"
GET	/groups/:id/edit/	"groups"	"edit"
GET	/groups/:id/	"groups"	"show"
PUT	/groups/:id/	"groups"	"update"
DELETE	/groups/:id/	"groups"	"destroy"

- **Singular-Plural, Groß-/Kleinschreibung:**
  - **Tabelle "orders"**
  - **Klasse "Order"**
- **Struktur der Verzeichnisse, Namen der Dateien**

Rails: Struktur einer Anwendung



Generatoren

- **Generator für ganze Anwendung:**

```
$ rails new anwendungsname
```

- **Generatoren für Modelle, Controller etc.**

```
$ rails generate controller Pages home contact
$ rails generate model User name:string email:string
$ rails generate migration add_email_uniqueness_index
$ rails generate integration_test users
```

- **Scaffolding:**

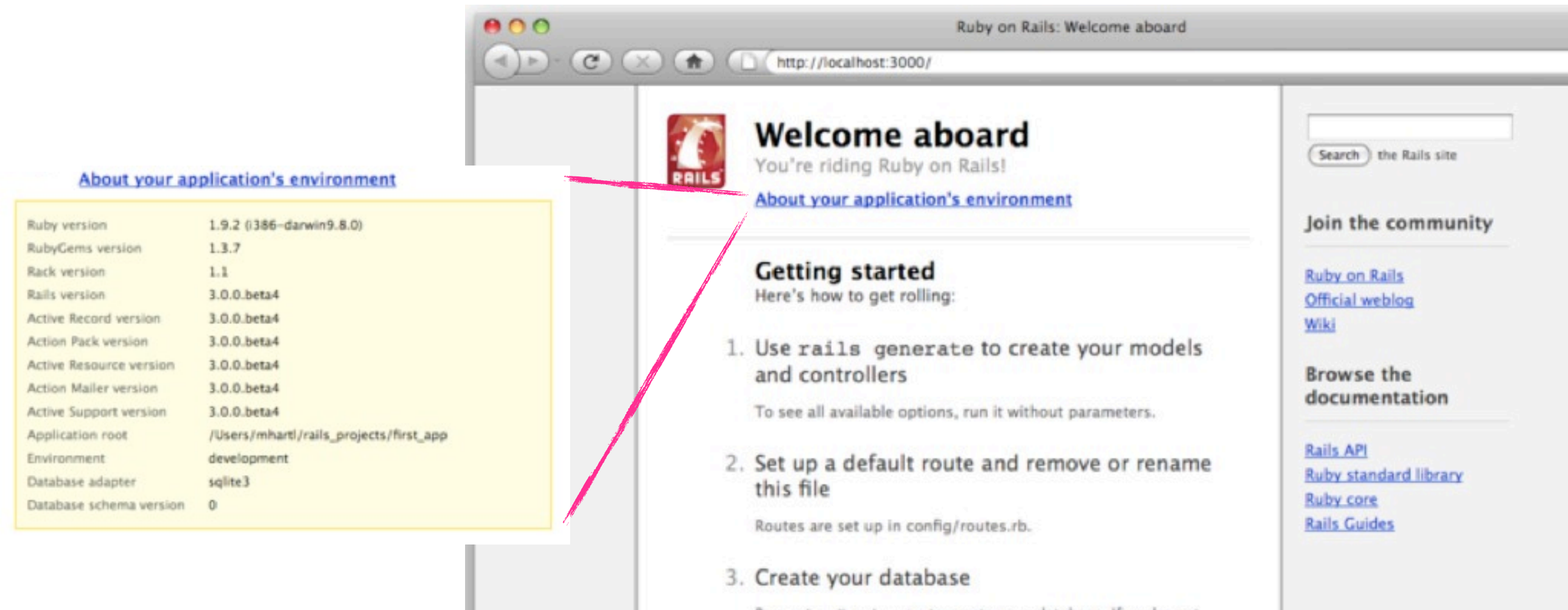
```
$ rails generate scaffold User name:string email:string
```

  - **erzeugt Gerüst für Model und Controller**
  - **kann Arbeit stark abkürzen**
  - **kann aber auch Korsett liefern...**

```
$ rails generate scaffold User name:string email:string
      invoke  active_record
      create    db/migrate/20100615004000_create_users.rb
      create    app/models/user.rb
      invoke    test_unit
      create      test/unit/user_test.rb
      create      test/fixtures/users.yml
       route  resources :users
      invoke  scaffold_controller
      create    app/controllers/users_controller.rb
      invoke    erb
      create      app/views/users
      create      app/views/users/index.html.erb
      create      app/views/users/edit.html.erb
      create      app/views/users/show.html.erb
      create      app/views/users/new.html.erb
      create      app/views/users/_form.html.erb
      invoke    test_unit
      create      test/functional/users_controller_test.rb
      invoke    helper
      create      app/helpers/users_helper.rb
      invoke      test_unit
      create        test/unit/helpers/users_helper_test.rb
      invoke  stylesheets
      create    public/stylesheets/scaffold.css
```

Rails-Server

- **Rails benutzt Webrick oder Mongrel**
- `rails s` **(Abkürzung für** `rails server`**)**
- `http://localhost:3000`



Bundler

- Eine Rails-Anwendung enthält ein halbes bis mehrere Dutzend Gems
- Wie sicherstellen, dass die Versionen zueinander passen?
- Bundler: auch ein Gem (z.Z. Version 1.0.10)
  - bundle install
- Arbeitet mit rvm zusammen: gemsets
  - freischwimmen von veralteten Betriebssystemumgebungen

Installation/Voraussetzungen

- **Ruby-Installation auf dem Laptop**
  - Windows: railsinstaller
  - Besser: Linux/OSX mit rvm
  - Ruby-Version: 1.9.2p136
- **rails 3.0.3, sqlite3-ruby, rspec, rspec-rails, webrat**
- **git (empfohlen) und/oder Subversion**
- **Entwicklungsumgebung? Freie Wahl**
  - IDEs: Eclipse (RadRails), NetBeans, ...
  - Textmate, e
  - Emacs, vim
- **Browser**
  - Safari, Chrome, Firefox (mit Firebug)

Vorgehen in der LV

- Aufgabenstellung durch „Kunden“
- Umsetzung durch Euch
- Planning Game, Story Cards, Features, Tasks
- TDD!

- Kommunikation zwischen Gruppen (Tip-Jar!)
- Kommunikationsrunden
- (Inhalte:) Neue Themen pro Tag, ggfs einbauen

Aufgabenstellungen

- Die Aufgabenstellungen geben die mehr oder minder naive Vorstellung eines Kunden wieder.
- Manche Aspekte sind nicht realisierbar.
- Manche würden gehen, ergeben dann aber in der Benutzung wenig Sinn.
- Manche sind realisierbar, aber zu aufwendig (Budget: 10 Tage!)

Vorgehen

- Aufgabenstellung interpretieren
- Ziele (evtl. mit Auftraggeber) priorisieren
  - Aufwand
  - Wirkung
- Erste Version einer Implementierung generieren
  - Dokumente können, müssen aber nicht dazugehören
  - Wichtig: Diese Version muss fertig werden!
- Feedback-Schleife mit Auftraggeber
- goto 1

Iterationen

- **Wir wissen nicht:**
  - **welche Features wirklich notwendig sind**
  - **was die Features kosten, und für wieviele wir Zeit haben werden**
- **All successful systems start simple.**
- **Work for outcome: Continuous delivery of value**
- **Abends fertig nach Hause gehen.**

Problem vs. Lösung

- Kunde hat vielleicht schon eine Lösung im Kopf
- Wir wollen aber sein Problem lösen!
  Stories können dabei helfen, das Problem einzufangen.
- Die Vorstellungskraft ist begrenzt.
  Nicht abstrakt, sondern

  über die entstehende Anwendung kommunizieren.

Paarprogrammierung

- **You never code alone.**
  - **Driver: sitzt an Tastatur/Maus.**
  - **Navigator: Denkt, denkt mit**
- **Naive Sicht: Halbe Produktivität.**
- **Tatsächlich:**
  - **Weniger Zeitverschwendung durch Debugging.**
  - **Schnellere Entwicklung!**
- **TDD: Red/green/refactor**

Vorgehen bis 21.02.2011

- http://railstutorial.org/book durcharbeiten
  - 12 Kapitel: eines pro Tag!
- Möglichst viel in der Gruppe zusammen machen
  - Paarprogrammierung üben
- Wäsche waschen, Kühlschrank füllen :-)
- Am 21.02.2011, 09:15: Gruppen festnageln
  - Montag: Sprint (alle Gruppen haben gleiche Aufgabe)
  - Ab Dienstag arbeiten Gruppen in ihren einzelnen Projekten

