Ruby & Rails - Extremcrashkurs

Jan Varwig - 9elements

(check http://jan.varwig.org for updates)

v1.0

Gliederung

- Ruby
  - Syntax
  - Methoden
  - Konventionen
  - Klassen
  - Module
  - Blöcke
  - Fehlerbehandlung
  - *Everything is an Object* und die Konsequenzen: Metaprogramming

Gliederung

- Rails
  - railties, getting started
  - Request processing: ActionController
  - ORM: Active Record
  - Codebeispiele
  - Weitere Module: ActiveResource, Actionmailer

Ruby Syntax

- Klammern unnötig:
  method(param) == method param

  Ausnahme: Geschachtelte Aufrufe
  method_one(method_two(param))

  Kein Error, nur Warning, aber trotzdem...

Ruby Syntax

- Semikolons unnötig (aber vorhanden)
  Endet die Zeile mit einer Expression,
  terminiert \n das Statement

  Gegenbeispiel:
  puts "Die Katze" +
  "tritt die Treppe krumm"

Ruby Syntax

- Block delimiter:
  def/do/begin
    ...
  end (Standard)
  statt do/end auch
  { ... } (gern genommen für Block-Einzeiler)

Ruby Syntax

- Postfix if/unless

```
def alterskontrolle(alter)
   raise „Zu jung!“ unless alter > 18
end
```

Ruby Syntax

- Herkömmliche Conditionals

```
if/unless condition
    ...
elsif other_condition
    ...
else
    ...
end
```

Ruby Syntax

- One-line prefix conditionals

  ```
  if condition then expression/statement
  else  expression/statement; end
  ```

- Für Minimalisten

  ```
  condition ? expression :
              other_expression
  ```

Ruby Syntax

- Namen
  - Constant / CONSTANT
  - $global_variable
  - @instance_variable
  - @@class_variable
  - local_variable (Vorsicht! Bei schlechter Bezeichnug verwechselbar mit Methodenaufrufen,)

Ruby Syntax

- Spezialitäten
  - parallele Zuweisung:
    a, b = b, a
  - Perl-Erbe: jede Menge lustige Variablen
    $_ $! $& $@ $~ $/
    So gut wie nie gebraucht, Ausnahmen
    - $! Die letzte Exception
    - $~, $1 - $9 Die letzte MatchData sowie die Matches der Subgroups

Methoden

- Automatische Rückgabe der letzten Expression:

```
def identity(arg)
    arg
end
def identity(arg)
    return arg
end
```

Methoden

- Variable Anzahl von Argumenten möglich.
- So nicht:

```
def varargs(args)
    args.inspect
end

>> varargs 1, 2
#ArgumentError
```

- Stattdessen

```
def varargs(*args)
    args.inspect
end

>> varargs 1, 2
=> "[1, 2]"
```

- Behold the mighty Splat Operator (Black Magick)

http://redhanded.hobix.com/bits/theSiphoningSplat.html
http://redhanded.hobix.com/bits/wonderOfTheWhenBeFlat.html

Methoden

- Named Parameters
  für optionale Argumente
- Nicht vorgesehen, aber leicht
  zu faken
- Gern genutzt in Rails
- Hash als Parameter

```
def test(options)
   options.inspect
end

>> test "a" => 1, "b" => 2
=> "{"a"=>1, "b"=>2}"
```

- Auch Kombination möglich

```
def test(arg1, arg2, options)
   puts arg1
   puts arg2
   puts options.inspect
end

>> test(1,2, "a" => 3, "b" => 4)

1
2
"{"a"=>3, "b"=>4}"
```

Operathoden :-)

- Operatoren sind auch nur Methoden!
  - 1 + 2
  - 1.+(2)
- Unterliegen gewohnten Präzendenzregeln

Konventionen

- Methodennamen
  - Boolean Abfragen mit Fragezeichen
    o.is_a? String, o.respond_to? :quack
  - Potentiell destruktiv mit Ausrufezeichen
    array.reverse! vs. array.reverse
  - Setter

```
def variable=(arg)
    @variable = arg
end
```

Typkonvertierung

- Ruby ist STATISCH GETYPT
- Bloß schert sich niemand um die Klassen
- Stattdessen respond_to? Anfragen und Konvertierungen
- to_s, to_i, to_a, to_sym

Konventionen

- Klassen/Module immer CamelCase
- Methoden/Variablen klein_mit_underscores
- Konstanten GROSS_MIT_UNDERSCORES
- Einrückungen: 2 Spaces

Strings & Literale

- String:
  "", " wie php
  (bzgl. substitutions)
  <<<-EOF
  Heredoc
  EOF

- Hash:
  {:a => 1, :b => 2}

- Array:
  [1,2,3]

- String-Arrays:
  %W(abc def ghi)
  =
  ["abc", "def", "ghi"]

  %w entspricht ' '

- String Interpolation
  "Es ist #{Time.now} "
  Beliebige Ausdrücke

- RegExps
  "Chunky Bacon" =~ /nky/
  1st class, wie Perl

Klassen

```
class Car
  @@cars_produced = 0              # Müssen in der Klasse initializiert werden
                                   # damit Ruby weiss wo die CVar abgespeichert
  attr_reader   :leistung          # werden soll
  attr_accessor :farbe

  def initialize(farbe, leistung)
    @farbe          = farbe        # Werden bei erster Verwendung init.
    @leistung       = leistung
    @@cars_produced = @@cars_produced + 1
  end

  def self.cars_produced           # = cattr_reader :cars_produced
    @@cars_produced
  end

private                            # Verwendung ohne expliziten empfänger
  ...
protected                          # Verwendung nur durch Objekte gleicher Klasse
public                             # Verwendung ohne Einschränkungen
end
```

Klassen

- Vererbung

```
class Diesel < Car
  def stinken
    puts "Stink " * 3
  end
end
```

- Single Inheritance only
- Besser als Multiple Inheritance: Mixins

Klassenmethoden

```
public class Test {
        public static xxx() {
                ...
        }
}

Test.xxx();
Test t = new Test();
t.xxx()
```

```
class Test
  def self.blubb
    puts blubb
  end
end

>> Test.blubb
blubb

>> Test.new.blubb
#NoMethodError
```

- Nicht zu verwechseln mit statischen Methoden in Java
- Aufrufbar auf der Klasse, nicht auf Instanzen der Klasse
- Hängt mit dem Method Dispatching von Ruby zusammen

Module

- Container für Methoden und Konstanten
- Keine speziellen Dateinamen nötig (wie etwa bei Python)
- Erfüllen mehrere Aufgaben

```
module A
  puts self.inspect
end


module A
  module B
    puts self.inspect
  end
end

Ausgabe:

A
A::B
```

Module

- Module als Namespaces
  - Zugriff auf innere Konstanten
  - Klassen in einem Modul sind nichts anderes (Klassen werden als Konstanten deklariert)

```
module A
  ID = "Module #{self.inspect}"
end


module B
  module A
    ID = "Module #{self.inspect}"
  end

  ID = "Module #{self.inspect}"

  puts ID
  puts A::ID
  puts ::A::ID
end

####

Module B
Module B::A
Module A
```

Module

- Mixins
  - Instanzmethoden des included Moduls werden Instanzmethoden des includenden Moduls/Klasse
  - Klassenmethoden etwas tricky, wieder aufgrund der Arbeitsweise des Lookups:
    - Definition in eigenem Submodul
    - Anhängen über `extend` im `included` Callback

```
module Mixin
  module ClassMethods
    def mixed_in_class_method
      "Mixed in Class Method, self = #{self.inspect}"
    end
  end

  def mixed_in_instance_method
    "Mixed in Instance Method, self = #{self.inspect}"
  end

  def self.included(klass)
    klass.extend(ClassMethods)
  end
end

class BoringClass
  include Mixin
end

puts BoringClass.mixed_in_class_method
puts BoringClass.new.mixed_in_instance_method
puts BoringClass.included_modules
```

```
Mixed in Class Method, self = BoringClass
Mixed in Instance Method, self = #<BoringClass:0x8adcc>
Mixin
Kernel
```

Method Lookup

- PickAxe 2nd Ed. Chapter 24
- Core Api: Module#include, Object#extend
  http://ruby-doc.org/core/
- http://whytheluckystiff.net/articles/seeingMetaclassesClearly.html

Blöcke

- Schleifen sind in Ruby selten
- Stattdessen Verwendung von Enumeratoren mit Blöcken:

```
[1,2,3].each do |val|
    puts val
end

[:a => 1, :b => 2].each { |key, val| ... }
```

Blöcke

- Blöcke haben Zugriff auf Symbole im umschließenden Scope:

```
a = "Hey "
(1..3).each do |num|
    puts a * num
end
```

```
Hey
Hey Hey
Hey Hey Hey
```

Closures

- Blöcke haben nicht nur Zugriff, sie merken sich auch selbst die Referenz
- D.h. Referenzen haben noch Gültigkeit wenn der umschließende Scope nicht mehr exisitert
- Verschieden Namen:

  lambdas, Procs, anonyme Methoden

On-the-fly Methoden

```
def gen_times(factor)
  return lambda {|n| n*factor }
end

times3 = gen_times(3)
times5 = gen_times(5)

times3.call(12)                 #=> 36
times5.call(5)                  #=> 25
times3.call(times5.call(4))     #=> 60
```

Gotchas

- Feine Unterschiede zwischen `Proc.new`, `lambda` und Blöcken
- Von Matz als unschön erkannt und in Ruby 2.0 geändert
- In der Praxis selten ein Problem: für anonyme Methoden idR. einfach `lambda` benutzen
- http://en.wikibooks.org/wiki/Ruby_Programming/Syntax/Method_Calls#Understanding_blocks.2C_Procs_and_methods
- http://blade.nagaokaut.ac.jp/cgi-bin/vframe.rb/ruby/ruby-talk/237500?237421-246391

```ruby
def foo
  f = Proc.new { return "return from foo from inside proc" }
  f.call # control leaves foo here
  return "return from foo"
end

def bar
  f = lambda { return "return from lambda" }
  f.call # control does not leave bar here
  return "return from bar"
end

puts foo # prints "return from foo from inside proc"
puts bar # prints "return from bar"
```

Eigene Enumeratoren

- Jede Methode kann Blöcke verarbeiten

```
def method()
  # Schleife über Elemente, Zeilen einer Datei o.ä.
    yield var1, var2 if block_given?
  # Schleifenende
end
```

- Funktioniert natürlich auch ohne Schleife, einmaliger Aufruf von yield

Enumeratoren

- Klassisch:
  while/until boolean_expression
    body
  end
- while/until auch als postfix Modifier:
  puts "Bla" while true
- Extrem selten gebraucht
- In Iteratorblöcken bekannte Kontrollbefehle:
  retry, next, break

Fehlerbehandlung

- Ruby doesn‘t *try*!*

```
def method
  do_something
rescue SyntaxError, NameError => boom
  puts "Oops: #{boom}"
  raise
else
  puts "Nothing raised"
ensure
  puts "Always executed"
end

raise
raise "bad mp3 encoding"
raise InterfaceException, "Keyboard failure"
```

* http://www.youtube.com/watch?v=Du_RTMmofWM

Weitere Resourcen

- IRB
- ri
- http://www.poignantguide.net/ruby
- http://www.ruby-doc.org/core
- ruby-debug

Lookup

- Include vs. Extend
  - include fügt Modul in eine Klasse ein
  - extend fügt Modul in Instanz ein
  - Klasse.extend fügt Modul-Methoden also als Klassenmethoden ein

Lookup

- Klassenvariablen @@variable
  - Initializiert in Klassendefinition
  - Zugreifbar aus Instanzen und Klassenmethoden
  - Wird bei Vererbung nicht überdeckt
    => Identische Daten für gesamte Vererbunghierarchie.
    Manchmal unerwünscht

Lookup

- Klasseninstanzvariablen
  - class Klasse
    @class_inst_var = :a
    def Klasse.get_civ
    return @class_inst_var
    end
    end
  - Gebunden an Instanz von *Class*
  - Erbende Klassen sind andere Instanzen von *Class*, haben also keinen Zugriff
  - Stark genutzt von ActiveRecord

Lookup

```
class Klasse
  @class_inst_var = :civ_klasse
  @@class_var     = :cvar_klasse
  def self.get_civ;   @class_inst_var; end
  def self.get_cvar; @@class_var;      end
end
puts "Klasse Class Variable: #{Klasse.get_cvar}" #
cvar_klasse

class SubKlasse < Klasse
  @class_inst_var = :civ_subklasse
  @@class_var     = :cvar_subklasse
  def self.get_civ;   @class_inst_var; end
end

puts Klasse.get_civ     # civ_klasse
puts SubKlasse.get_civ  # civ_subklasse
puts
puts Klasse.get_cvar    # cvar_subklasse
puts SubKlasse.get_cvar # cvar_subklasse
```

Nachlesen

- http://whytheluckystiff.net/articles/seeingMetaclassesClearly.html
- Programming Ruby 2nd Edition - Chapter 24
  Classes and Objects, speziell ab Seite 371 „Class Instance Variables“
- Programming Ruby 2nd Edition - Chapter 26
  Reflection, ObjectSpace, and Distributed Ruby

Metaprogramming

- EVERYTHING is an Object:
  - 1.class => Fixnum
  - "BlaBla".class => String
  - :blabla.class => Symbol
  - Symbol.class => Class
  - Symbol.class.class => Class

Metaprogramming

- Klassen inspizieren:
  o.class.superclass, o.class.ancestors, o.class.included_modules
- Objekte inspizieren:
  o.methods, o.instance_variables
- Methoden zu Klassen hinzufügen:
  class ExistingClass; def method; ...; end; end
- Methoden zu Objekten hinzufügen:
  class << object; def method; ...; end; end
- Methoden aus Klassen entfernen:
  class ExistingClass; undef_method :method ; end

Rails

Getting started

- InstantRails
  Rails a lá XAMPP, All-In-One, ready to go (?)
- Standardinstallation
  - Ruby One-Click Installer (ruby-lang.org)
  - reboot (wegen PATH)
  - gem update --system
  - gem install rails / ruby-debug -v 0.9.3 / mysql / mongrel
- Mac OS X Leopard: Keine Ahnung, Darwinports wahrscheinlich als beste Lösung
- Debian/Ubuntu: apt-get install ruby, danach zu Fuß weiter

Anlegen einer App

- rails -d mysql <app_name>
- config/database.yml checken und die Datenbanken (development/test) anlegen
- script/generate model xyz
  - app/models/user.rb
  - test/unit/user_test.rb
  - test/fixtures/users.yml
  - db/migrate/001_create_users.rb
- script/generate controller user
  - app/views/user
  - app/controllers/user_controller.rb
  - test/functional/user_controller_test.rb
  - app/helpers/user_helper.rb
- script/generate resource subscription
  - create app/views/subscriptions
  - dependency model
  - create app/models/subscription.rb
  - create test/unit/subscription_test.rb
  - create test/fixtures/subscriptions.yml
  - create db/migrate/002_create_subscriptions.rb
  - create app/controllers/subscriptions_controller.rb
  - create test/functional/subscriptions_controller_test.rb
  - create app/helpers/subscriptions_helper.rb
  - route map.resources :subscriptions
- script/generate migration
  - db/migrate/002_users_add_email_field.rb

Migrations

```
class CreateMemberships < ActiveRecord::Migration
  def self.up
    create_table(:memberships, :options => "DEFAULT CHARSET=utf8") do |t|
      t.integer  :contact_id
      t.date     :anfang
      t.date     :ende
      t.boolean  :ausserordentlich
    end
    add_index(:memberships, :contact_id)
  end

  def self.down
    drop_table :memberships
  end
end
```

- 2 Richtungen: Up and Down
- Jeglicher Rails Code möglich
- Datenbank Statements (ActiveRecord, Schema statements), raw SQL
- [RAILS_ENV=<test>] rake db:migrate [VERSION=<version>]
- Initialisieren einer Datenbank aus bestehendem Schema NICHT über Migrations sondern über rake db:schema:load

ActionController

- Request Processing
  - Routing -> Controller, Action, Parameter, Method
  - Filter verarbeitung
  - Action-Verarbeitung
    - Interaktion mit ActiveRecord, ActiveResource, ActionMailer, Filesystem etc.
    - Redirect oder Render -> performed Flag
    - Wenn kein performed, automatischer Aufruf der Template, basierend auf (Controller, Action, Accept-Header)

ActionController

- Sessions
  - Einfach wie Hash verwenden
  - session[:userid] = User.find_by_name params["username"]
  - Default store in Rails 2.0.2 in Cookies, mögliche aber auch: Filesystem, Datenbank, RAM, memcached
- Flash
  - Einmalige Mini-Session, überlebt nur einen Request
  - gedacht für Fehlermeldungen, Kommnikation zwischen Controllern bei aufeinanderfolgenden Requests etc.
  - flash[:warn] = "Es sind Fehler aufgetreten" unless object.save

ActionController

- Routing
  - einfache Formatstrings die URLs auf Controller, Actions und Parameter abbilden
  - Klassisch: map.connect "/:controller/:action/:id.:format"
  - REST-Style: map.resources :users

```
            users GET    /users                {:action=>"index",  :controller=>"users"}
  formatted_users GET    /users.:format        {:action=>"index",  :controller=>"users"}
                  POST   /users                {:action=>"create", :controller=>"users"}
                  POST   /users.:format        {:action=>"create", :controller=>"users"}
         new_user GET    /users/new            {:action=>"new",    :controller=>"users"}
formatted_new_user GET    /users/new.:format    {:action=>"new",    :controller=>"users"}
        edit_user GET    /users/:id/edit       {:action=>"edit",   :controller=>"users"}
formatted_edit_user GET  /users/:id/edit.:format {:action=>"edit",  :controller=>"users"}
             user GET    /users/:id            {:action=>"show",   :controller=>"users"}
    formatted_user GET    /users/:id.:format    {:action=>"show",   :controller=>"users"}
                  PUT    /users/:id            {:action=>"update", :controller=>"users"}
                  PUT    /users/:id.:format    {:action=>"update", :controller=>"users"}
```

ActionController

- Vererbung
  - Filter, Actions, Settings etc. werden vererbt
  - Oberster Controller idR. der ApplicationController < ActionController
  - Andere Controller erben vom ApplicationController

ActionController

- Rendern
  - Expliziter Aufruf von render / redirect
  - automatisches Rendern app/views/controller/action.html.erb
  - Für gewöhnlich im Kontext eines Layouts
  - Layout-Lookup nach Controllername, durch die Vererbungshierarchie
  - Im Layout Platzierung des Views durch `yield`

ActionController

- Helper
  - app/helpers
  - class DingsBumsController
  - module DingsBumsHelper
  - Sammlung von Methoden die im View verwendet werden können
  - Vererbung gilt:
    ApplicationHelper steht überall zur Verfügung

ActiveRecord

- Rails ORM Layer
- Model.find, Model.new
- record.delete, record.save
- Pro Tabellenspalte ein Attribut im richtigen Typ (DATETIME => Date, TINYINT(1) => Bool, etc.)

ActiveRecord

- Konventionen
  - Models singular und CamelCase
    User
  - Entsprechende Tabellen plural, underscored
    users
  - stets eine auto_increment Spalte id
  - Foreign Keys über other_model_id
    In posts Tabelle z.B. user_id
- Ausnahmen lassen sich einrichten (andere Tabellennamen, andere primary keys, andere foreign keys, eigene Pluralisierungsregeln)

ActiveRecord

- Validations
  - vordefinierte validates_format, etc.
  - eigene validates_each
  - Überschreiben von ActiveRecord#validate_on_create / validate_on_update

Errors

- Von den Validations erzeugt um Fehlschläge festzuhalten
- model.errors[:attribute]
- self.errors.add :attribute, "Fehlerbeschreibung"

```
def validate
  super
  self.ratings.each do |r|
    errors.add 'ratings', "Rating #{r.inspect} is invalid" unless rating_valid?(r)
  end
end
```

Associations

- One-to-one
- class ModelA < ActiveRecord
  has_one :model_b
  end
- class ModelB < ActiveRecord
  belongs_to :model_a
  end
- Tabelle mit Foreign Key erhält belongs_to, die andere has_one

Associations

- Many-to-one
- class ModelA < ActiveRecord
  has_many :model_b
  end
- class ModelB < ActiveRecord
  belongs_to :model_a
  end

Associations

- Many-to-Many
- class Category < ActiveRecord
  has_and_belongs_to_many :products
  end
- class Product < ActiveRecord
  has_and_belongs_to_many :categories
  end
- + Join Table categories_products (alphab.) mit Spalten category_id, product_id

Associations

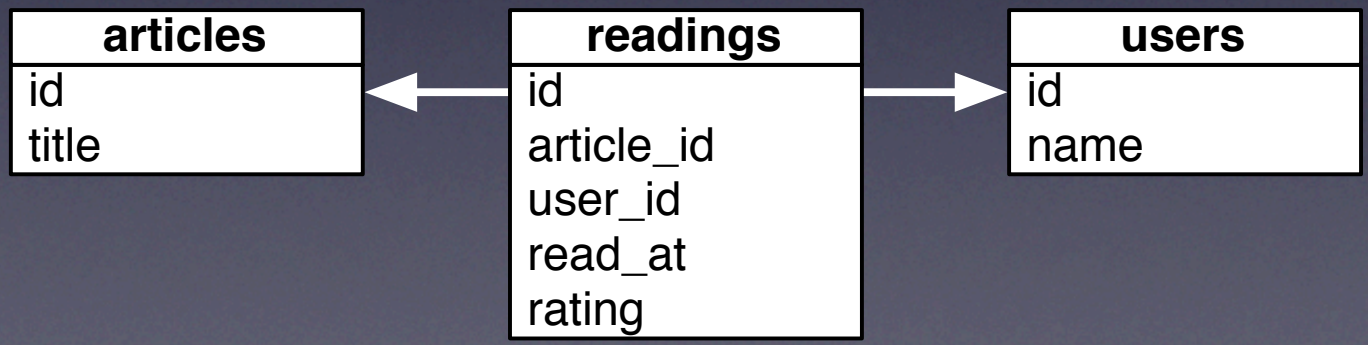
- HABTM lässt keine Informationen über die Assoziation zu
- Daher: has_many :through
- Nicht nur Join Tables sondern Join Models

Associations

```
class Article < ActiveRecord::Base
  has_many :readings
  has_many :users, :through => :readings
end

class User < ActiveRecord::Base
  has_many :readings
  has_many :articles, :through => :readings
end

class Reading < ActiveRecord::Base
  belongs_to :article
  belongs_to :user
end
```



Associations

- Association Helpers
  - #other
  - #other=(other)
  - #build_other(attributes={})
  - #create_other(attributes={})
  - #other.create!(attributes={})
  - #other.nil?
- Association Proxies für *-to-many Assoziationen
  - #others
  - #others=(other,other,...)
  - #other_ids
  - #other_ids=(id,id,...)
  - #others.push
  - #others.concat
  - #others.build(attributes={})
  - #others.count
  - #others.empty?
  - #others.clear
  - #others.delete(other,other,...)
  - #others.delete_all
  - #others.find(*args)
  - #others.find_first

Snippets

```
def self.authenticate(email, pass)
   u = find(:first, :conditions=>["email = ?", email])
   return nil if u.nil?
   return u if User.encrypt(pass)==u.password
   nil
end
```

Snippets

```
module ApplicationHelper
   def flash_div(*keys)
       keys.collect {|key|
           content_tag(:div, flash[key], :class => "flash #{key}") if flash[key]
       }.join
   end
end
```

Testing

- rake test
- Unit Tests
  Testen Models
- Functional Tests
  Testen (einzelne) Controller
- Integration Tests
  Testen Zusammenspiel von Controllern, Zugriff auf Datenbank
- Automatisch angelegt durch script/generate

Unit-Test

```
require File.dirname(__FILE__) + '/../test_helper'
class ProductTest < Test::Unit::TestCase
  fixtures :products

def test_invalid_with_empty_attributes
    product = Product.new
    assert !product.valid?
    assert product.errors.invalid?(:title)
    assert product.errors.invalid?(:description)
    assert product.errors.invalid?(:price)
    assert product.errors.invalid?(:image_url)
  end
end
```

Functional Test

```
require File.dirname(__FILE__) + '/../test_helper'
require 'login_controller'

# Re-raise errors caught by the controller.
class LoginController; def rescue_action(e) raise e end; end

class LoginControllerTest < Test::Unit::TestCase
  def setup
    @controller = LoginController.new
    @request    = ActionController::TestRequest.new
    @response   = ActionController::TestResponse.new
  end

  def test_index_without_user
    get :index
    assert_redirected_to :action => "login"
    assert_equal "Please log in", flash[:notice]
  end
end
```

Integration Test

```
require "#{File.dirname(__FILE__)}/../test_helper"

class UserStoriesTest < ActionController::IntegrationTest
  fixtures :products

  # A user goes to the index page. They select a product, adding it to their
  # cart, and check out, filling in their details on the checkout form. When
  # they submit, an order is created containing their information, along with a
  # single line item corresponding to the product they added to their cart.

  def test_buying_a_product
    LineItem.delete_all
    Order.delete_all
    ruby_book = products(:ruby_book)

    get "/store/index"
    assert_response :success
    assert_template "index"

    xml_http_request :put, "/store/add_to_cart", :id => ruby_book.id
    assert_response :success

    cart = session[:cart]
    assert_equal 1, cart.items.size
    assert_equal ruby_book, cart.items[0].product

    post "/store/checkout"
    assert_response :success
    assert_template "checkout"
```

Integration Test

```
    post_via_redirect "/store/save_order",
                      :order => { :name     => "Dave Thomas",
                                  :address  => "123 The Street",
                                  :email    => "dave@pragprog.com",
                                  :pay_type => "check" }
    assert_response :success
    assert_template "index"
    assert_equal 0, session[:cart].items.size

    orders = Order.find(:all)
    assert_equal 1, orders.size
    order = orders[0]

    assert_equal "Dave Thomas",       order.name
    assert_equal "123 The Street",    order.address
    assert_equal "dave@pragprog.com", order.email
    assert_equal "check",             order.pay_type

    assert_equal 1, order.line_items.size
    line_item = order.line_items[0]
    assert_equal ruby_book, line_item.product
  end
end
```

Testing

- Test-Datenbank vorbereiten mit rake db:test:prepare
- Starten mit rake test[:unit|:functionals]
- Fixtures in test/fixtures

RTFM

- http://api.rubyonrails.com/
- http://guides.rubyonrails.com/
- RoR-Talk (Google Groups)
- Google
- The Rails Way (Obie Fernandez)
- Agile Web Development with RoR 3rd Ed.

Selbsthilfe

- script/server --debugger
- debugger
- http://cheat.errtheblog.com/s/rdebug/