Agile Webentwicklung mit Ruby on Rails

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

UNIVERSITÄT OSNABRÜCK

Institut für Informatik
AG Medieninformatik

Ruby on Rails

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

Universität Osnabrück
http://www-lehre.inf.uos.de/~ror
21.06.2012

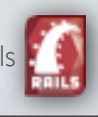
Agile Webentwicklung mit Ruby on Rails

Sommersemester 2012

Blatt 6

Im Ordner `blog` finden Sie die in der Vorlesung erzeugte Applikation *Blog*. Alle Aufgaben sollten nacheinander in derselben Applikation bearbeitet werden.

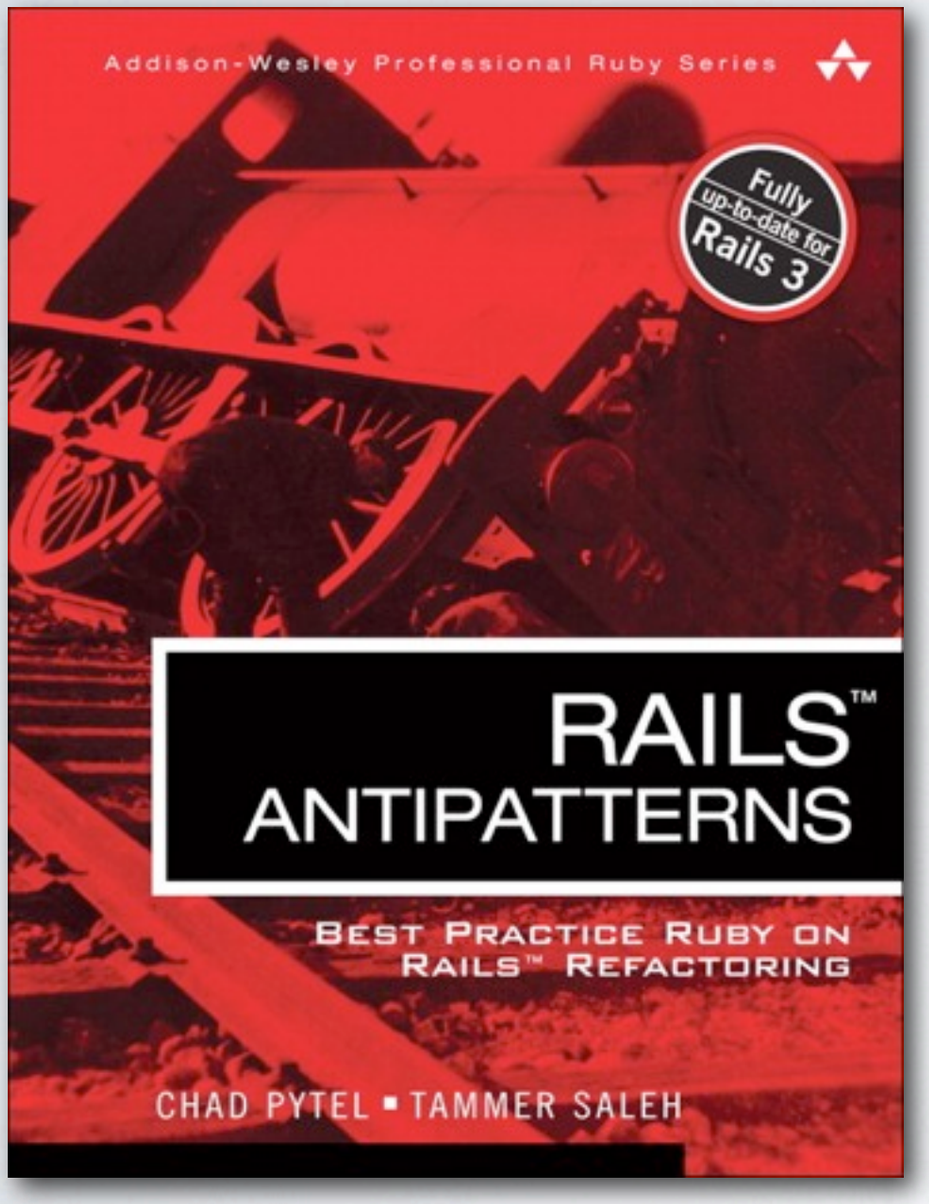
Aufgabe 1: User-Website



Nested Resources

Ruby on Rails

Quellen/Literatur



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

S. 170 ff.

Nested Resources

- Abbildung von 1:n-Beziehungen in einer zu REST konformen Art und Weise

- Gut anwendbar bei starken Eltern-Kind-Beziehungen, wenn eine Ressource nur im Kontext einer anderen bearbeitet werden soll

Beispiel: Alben & Songs

```
class Album < ActiveRecord::Base
  has_many :songs
end

class Song < ActiveRecord::Base
  belongs_to :album
end
```

Album#show

```
#app/views/albums/show.html.erb

<h2><%= @album.title %></h2>
<p>By: <%= @album.artist %></p>

<ul>
  <% @album.songs.each do |song| %>
    <li><%= link_to song.title, song %></li>
  <% end %>
</ul>

<%= link_to "Add song",
      new_song_url(album_id: @album.id) %>
```

Controller und Formular

```ruby
class SongsController < ApplicationController
  def new
    @song = Song.new(album_id: params[:album_id])
  end
  # ...
end
```

Code Smell!

```erb
<%= form_for(@song) do |f| %>
  <%= f.hidden_field :album_id %>
  <%# ... %>
<% end %>
```

Lösung: Nested Resource

```ruby
# config/routes.rb
MyApp::Application.routes.draw do
  resources :albums do
    resources :songs
  end
  # ...
end
```

```
    album_songs GET    /albums/:album_id/songs(.:format)
                POST   /albums/:album_id/songs(.:format)
 new_album_song GET    /albums/:album_id/songs/new(.:format)
edit_album_song GET    /albums/:album_id/songs/:id/edit(.:format)
     album_song GET    /albums/:album_id/songs/:id(.:format)
                PUT    /albums/:album_id/songs/:id(.:format)
                DELETE /albums/:album_id/songs/:id(.:format)
```

Album laden mit Before-Filter

```
class SongsController < ApplicationController
  before_filter :grab_album_from_album_id

  # ...

  protected
  def grab_album_from_album_id
    @album = Album.find(params[:album_id])
  end
end
```

Zugriff über Association-Proxy

```
class SongsController < ApplicationController
  before_filter :grab_album_from_album_id

  def index
    @songs = @album.songs.all
  end

  def create
    @song = @album.songs.new(params[:song])
    if @song.save
      redirect_to([@album, @song])
    else
      render action: "new"
    end
  end
  # ...
end
```

Anpassungen in der View

```
# app/views/songs/_form.html.erb
<%= form_for([@album, @song]) do |f| %>
  # ...
<% end %>

# app/views/songs/edit.html.erb
<h1>Editing song</h1>
<%= render 'form' %>

<%= link_to 'Show', [@album, @song] %>
```

Nested und Un-nested: Routendefinition

```
# config/routes.rb
MyApp::Application.routes.draw do
  resources :albums do
    resources :songs
  end
  resources :songs
  # ...
end
```

Nested und Un-nested: Routes

```
    album_songs GET    /albums/:album_id/songs(.:format)
                POST   /albums/:album_id/songs(.:format)
 new_album_song GET    /albums/:album_id/songs/new(.:format)
edit_album_song GET    /albums/:album_id/songs/:id/edit(.:format)
     album_song GET    /albums/:album_id/songs/:id(.:format)
                PUT    /albums/:album_id/songs/:id(.:format)
                DELETE /albums/:album_id/songs/:id(.:format)
          songs GET    /songs(.:format)
                POST   /songs(.:format)
       new_song GET    /songs/new(.:format)
      edit_song GET    /songs/:id/edit(.:format)
           song GET    /songs/:id(.:format)
                PUT    /songs/:id(.:format)
                DELETE /songs/:id(.:format)
```

Nested und Un-nested

```
class SongsController < ApplicationController
  before_filter :grab_album_from_album_id

  def index
    @songs = songs.all
  end

  protected
  def songs
    @album ? @album.songs : Song
  end

  def grab_album_from_album_id
    if params[:album_id]
      @album = Album.find(params[:album_id])
    end
  end
end
```

Authentication & Authorization

Ruby on Rails

Definitionen

Authentication

Bestätigung der Identität einer Person oder Software

Authorization

Spezifikation und Überprüfung der Zugriffsrechte auf Ressourcen

Authentication mit Devise

https://github.com/plataformatec/devise
http://railscasts.com/episodes/209-introducing-devise

- Rails-Engine mit eigenen Views und Controllern
- Zahlreiche mögliche Backends (Datenbank, OAuth, OpenID, CAS,)
- Modularer Aufbau (Database Authenticable, Token Authenticable, Omniauthable, Confirmable, Recoverable, Registerable, Rememberble, Trackable, Timeoutable, Validatable, Lockable)

Einfaches Beispiel

```
class User < ActiveRecord::Base
  devise :database_authenticable, :rememberable
end

class ApplicationController < ActionController::Base
  before_filter :authenticate_user!
end
```

Weiterleitung zur Login-Seite bei nicht authentifiziertem User

```
# provided helpers

user_signed_in?
current_user
user_session
```

Authorization mit CanCan

https://github.com/ryanb/cancan

http://railscasts.com/episodes/192-authorization-with-cancan

- Auf REST aufgebaute Bibliothek zur Verwaltung von Zugriffsrechten auf Ressourcen
- Zugriffsrechte werden an einem festen Ort definiert und nicht über Views und Controller verteilt
- Arbeitet nach dem *Whitelist-Prinzip*

Rechtedefinition

```
class Ability
  include CanCan::Ability

  def initialize(user)
   # guest user (not logged in)
    user ||= User.new
    if user.admin?
      can :manage, :all
    else
      can :read, :all
    end
  end
end
```

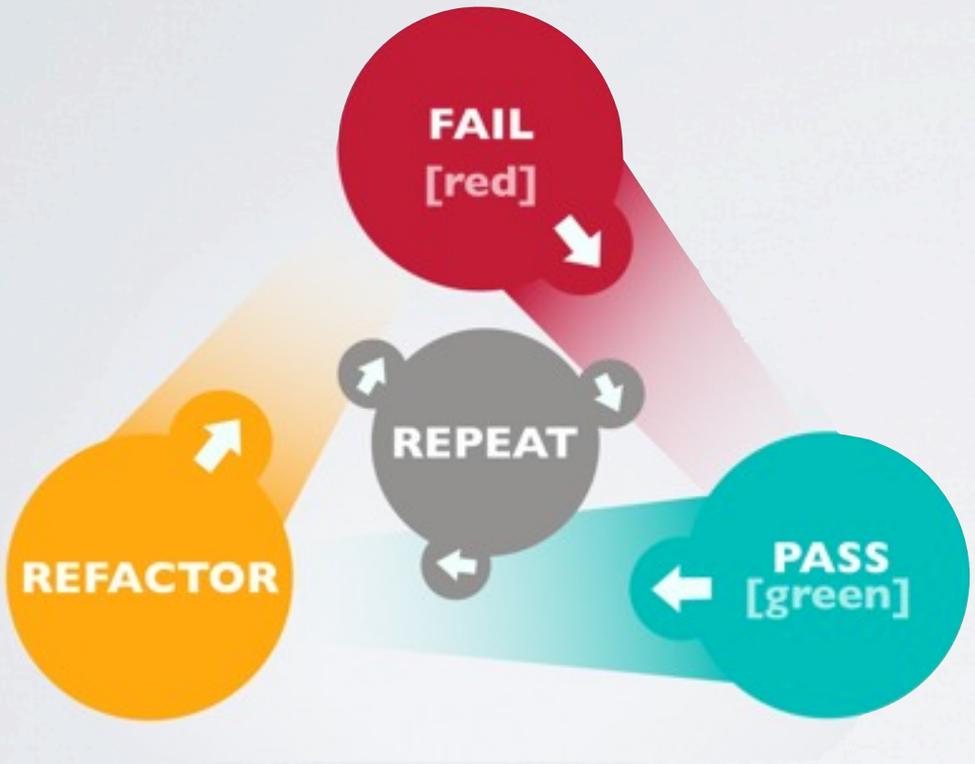
Rechte prüfen

```
<% if can? :update, @article %>
  <%= link_to "Edit", edit_article_path(@article) %>
<% end %>
```

```
def show
  @article = Article.find(params[:id])
  authorize! :read, @article
end
```

```
class ArticlesController < ApplicationController
  load_and_authorize_resource

  def show # @article is already loaded and authorized
  end
end
```



Agile Software-entwicklung

Agiles Manifest

http://agilemanifesto.org/

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and Interactions over processes and tools
Working Software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more."

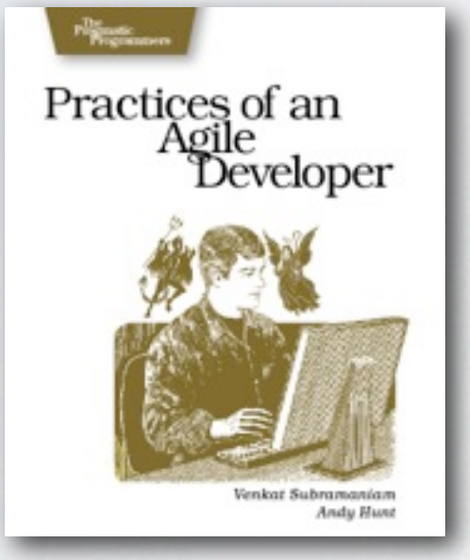
Wichtige Prinzipien

- Customer collaboration over contract negotiation
- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
- Welcome changing requirements, even late in development
- Continuous attention to technical excellence and good design enhances agility
- Simplicity - the art of maximizing the amount of work not done - is essential

Zusammenfassung

"Agile development uses feedback to make constant adjustment in a highly collaborative environment."

- Venkat Subramanian und Andy Hunt

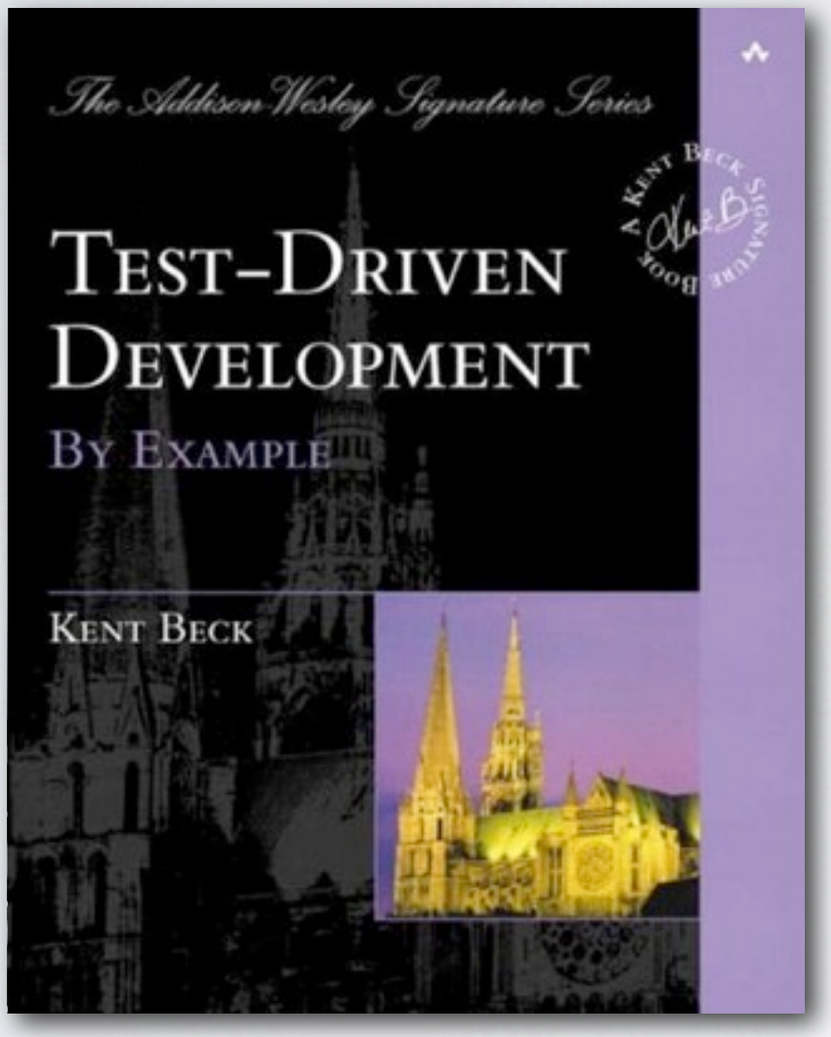


Practices of an Agile Developer,
Subramanian, V. and Hunt, A.,
Pragmatic Bookshelf 2006

Test-Driven Development

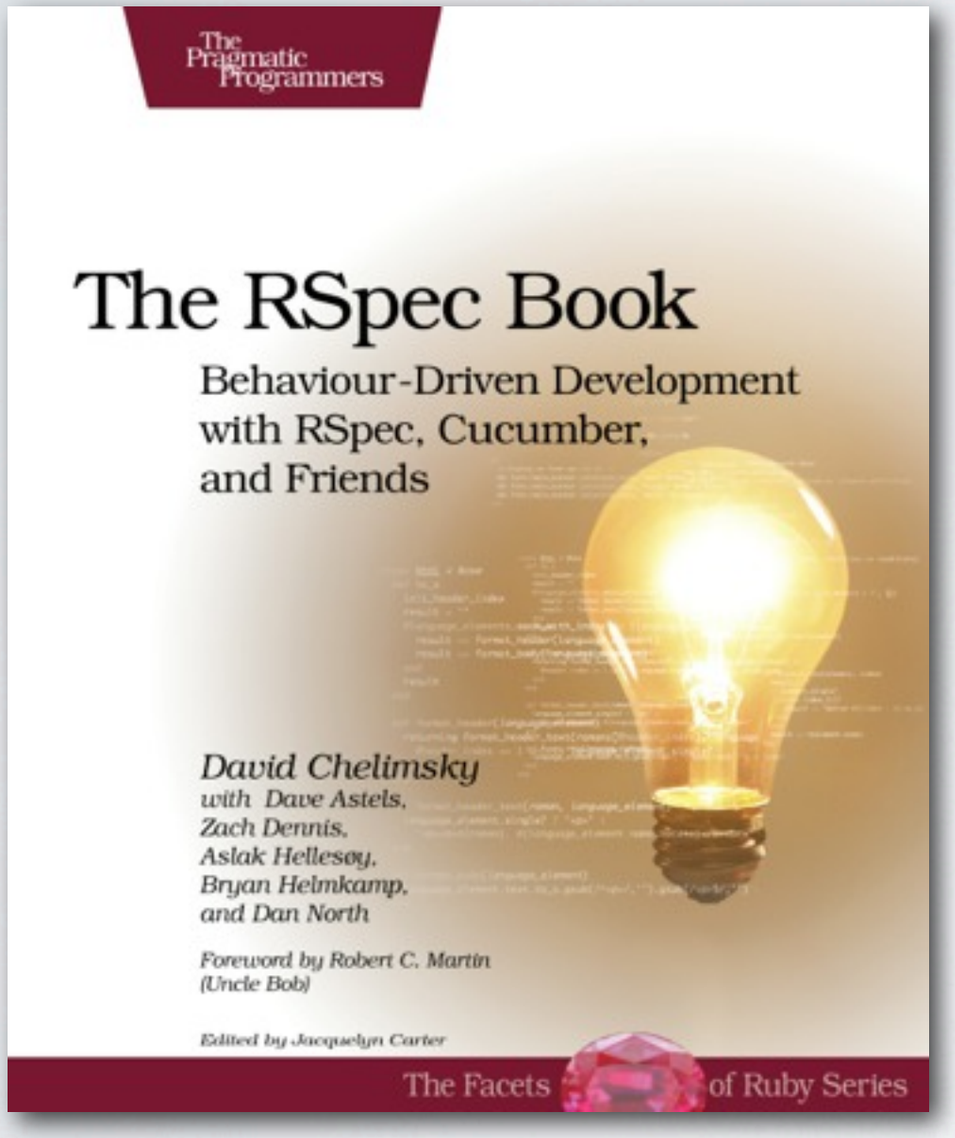
Agile Software-entwicklung

Quelle / Literatur



Test-Driven Development By Example,
Beck, K.,
Addison-Wesley, 2003

Quellen/Literatur



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

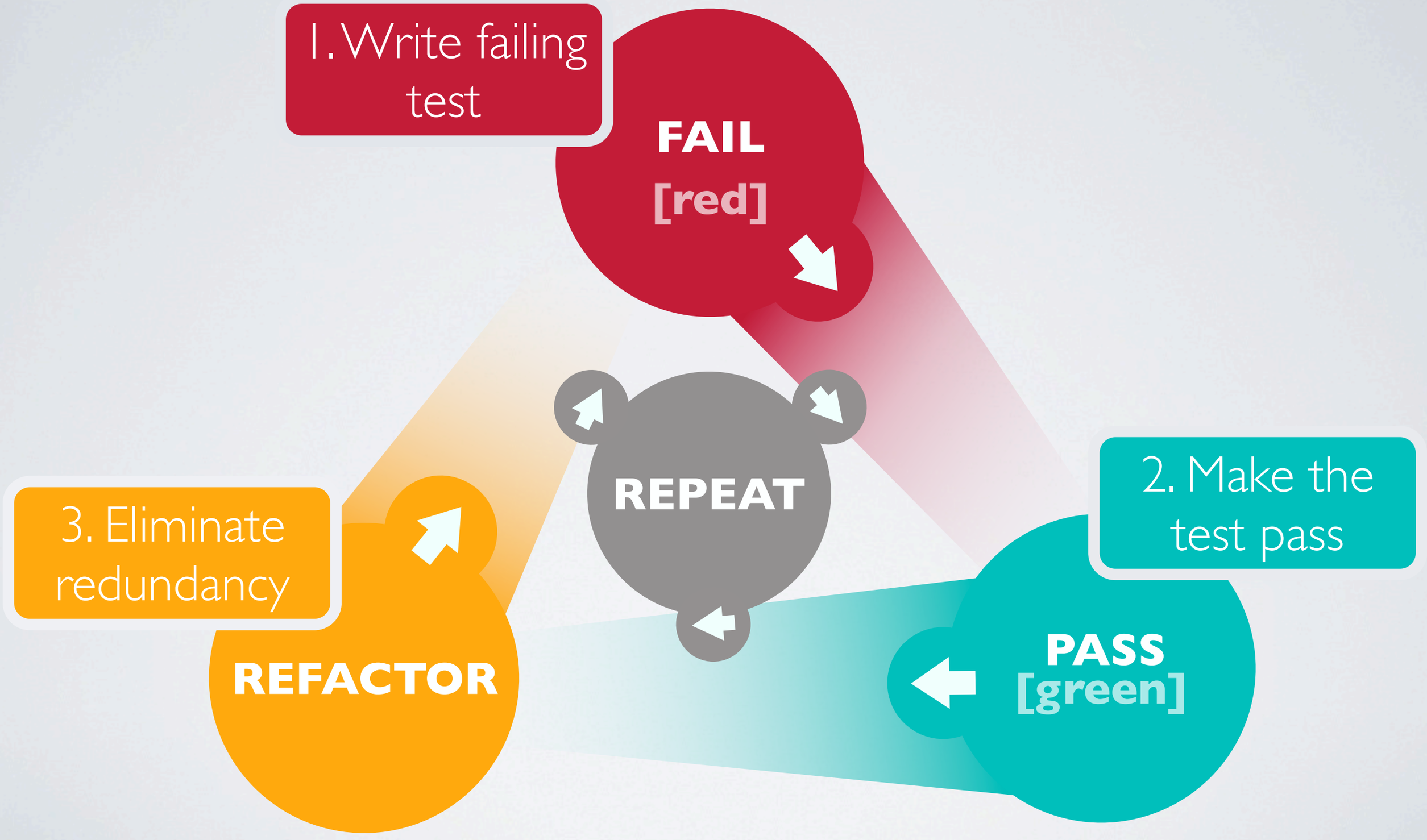
Test-Driven Development (TDD)

- Idee: Automatisierte Tests für Komponenten vor dem eigentlichen Code schreiben

- Ursprung der Idee im NASA-Projekt *Mercury* in den 1960er Jahren

- Größere Popularität als fester Bestandteil der agilen Methode *Extreme Programming (ca. 1995-2000)*

Red-Green-Refactor-Cycle



Bildquelle: http://www.agileapps.co.uk

Kopplung und Kohäsion

Kopplung

Abhängigkeit der Module untereinander

Kohäsion

Abbildung genau einer Verantwortlichkeit auf ein Modul

TDD wirkt sich positiv auf Kopplung und Kohäsion aus, da sich der Entwickler bereits im Vorfeld Gedanken über die Schnittstelle machen und mit dieser auch direkt selbst arbeiten muss.

Vor- und Nachteile von TDD

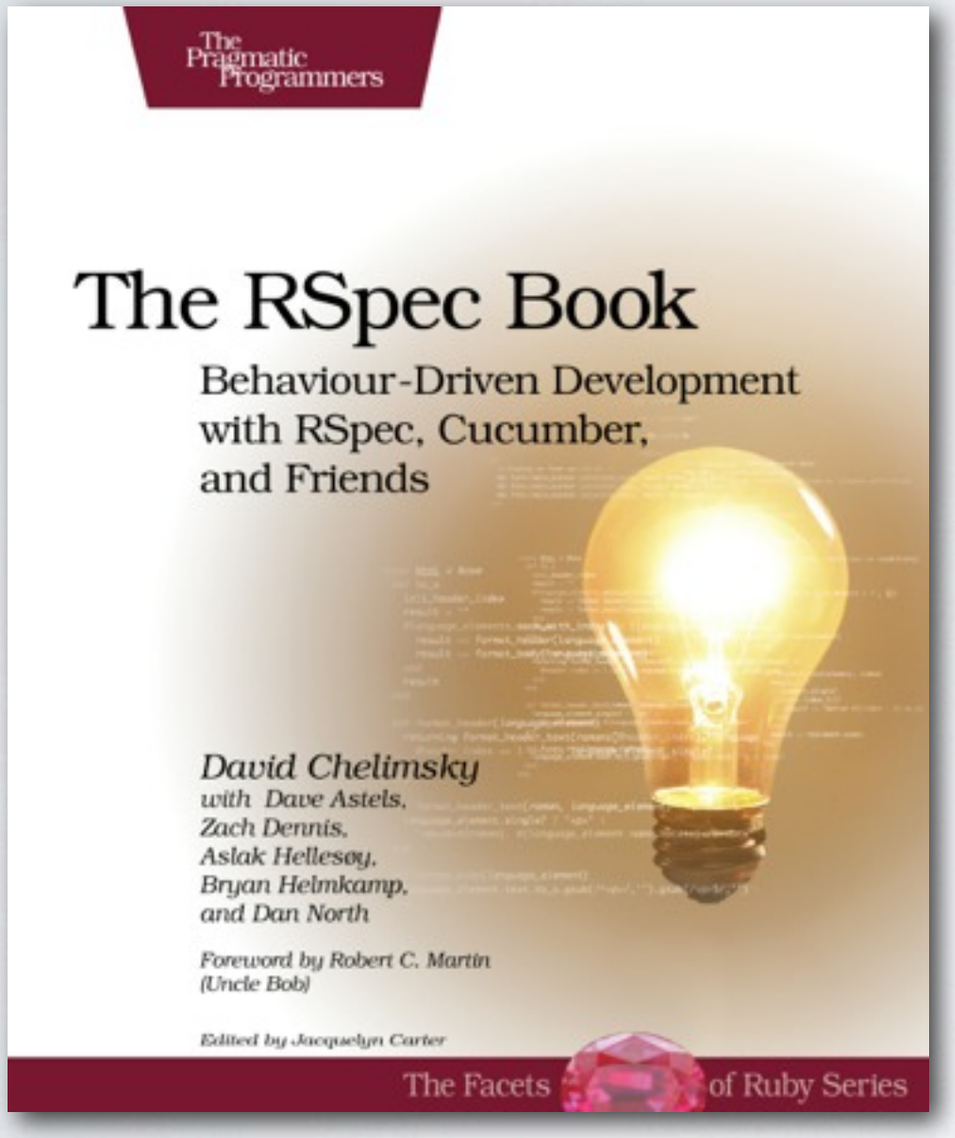
M. Siniallto and P. Abrahamson,
"A comparative case study on the impact of test-driven development on program design and test coverage",
In: *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pp. 275-284, IEEE, 2007

- 15% - 35% mehr Zeitaufwand
- Fehlerrate reduziert um bis zu 50%
- Verbesserte Testabdeckung
- Oft starke Verbesserung des Designs (Kopplung, Kohäsion)
- Effekt auf Kopplung und Kohäsion hängt stark von Vorerfahrung der Entwickler ab

Behaviour-Driven Development

Agile Software-entwicklung

Quellen/Literatur



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

Behaviour-Driven Development

- Ursprünglich: Neuausrichtung von TDD zum erleichterten Einstieg
- Heute: Eigenständiges Vorgehensmodell agiler Softwareentwicklung
- Eingesetzte agile Methoden: TDD, Domain-Driven Design, Acceptance Test-Driven Planning
- Entwicklung in Iterationen

Grundprinzipien

Behaviour-Driven Development is about implementing an application by describing its behaviour from the perspective of its stakeholders.

Enough is enough Up-front planning, analysis, and design all have a diminishing return. We shouldn't do less than we need to get started, but any more than that is wasted effort. This also applies to process automation. Have an automated build and deployment, but avoid trying to automate everything.

Grundprinzipien

Deliver stakeholder value If you are doing something that isn't either delivering value or increasing your ability to deliver value, stop doing it, and do something else instead.

It's all behaviour Whether at the code level, the application level, or beyond, we can use the same thinking and the same linguistic constructs to describe behavior at any level of granularity.

Spezifikation der Funktionalität

Feature Set

Vage formulierte Features, die die Applikation später haben soll

Story

Genauere Spezifikation eines Teils eines Feature Sets im *Connextra Format:*

As a [stakeholder] I want [feature] so that [benefit]

Scenario

Kleinschrittig formulierter Anwendungsfall mit Akzeptanzkriterien

Beispiel

Feature Set: Creating assignment sheets

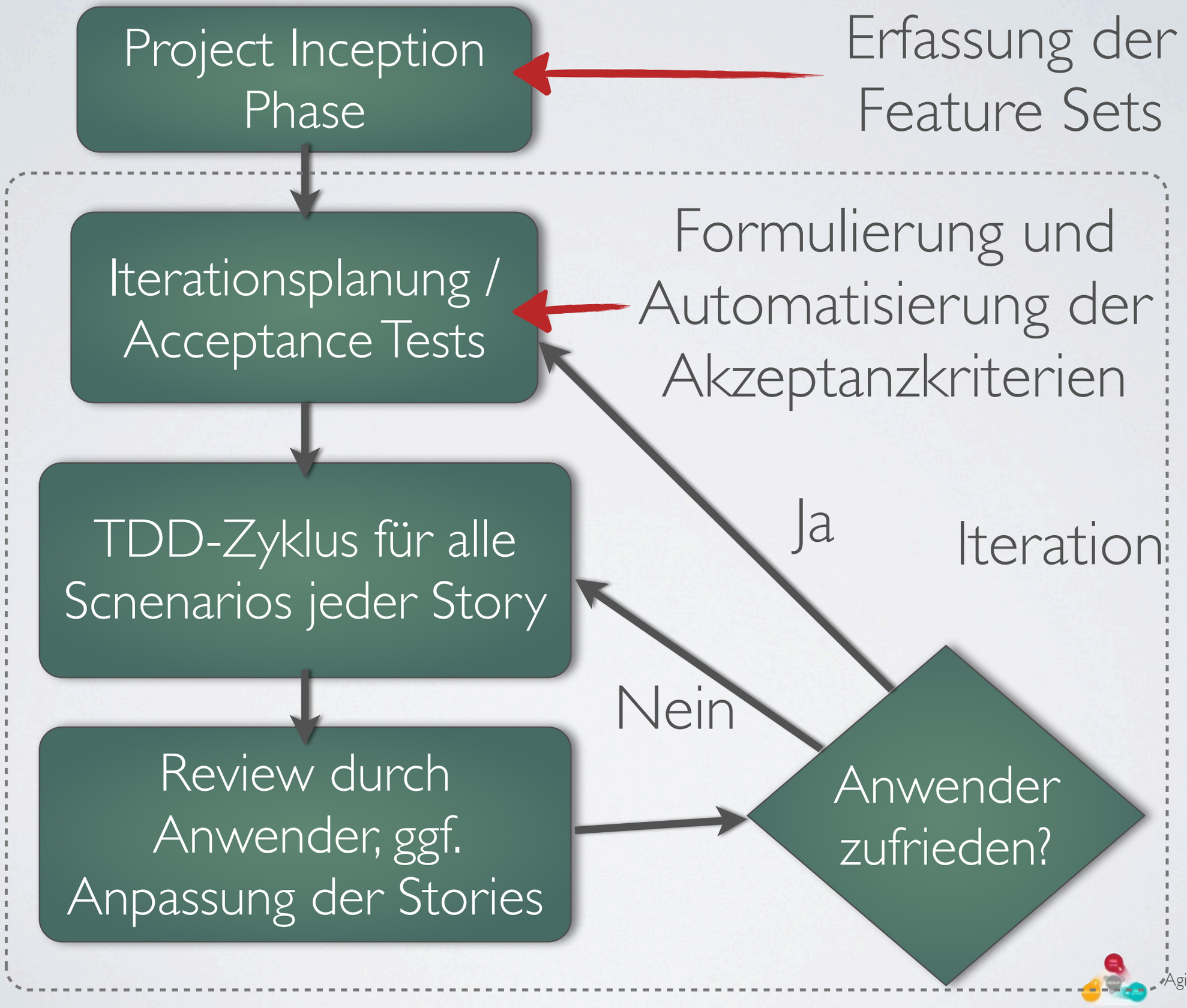
Story: Searching assignments

As a teaching assistant
I want to search for assignments with a search form
So that I can see easily which assignments have been used in the preceding years

Scenario: Performing a search

Given I am logged in as teaching assistant
When I go to the assignments page
And submit a search phrase into the search form
Then I want to see the assignments matching the search phrase

Ablauf


Project Inception Phase
Erfassung der Feature Sets
Iterationsplanung / Acceptance Tests
Formulierung und Automatisierung der Akzeptanzkriterien
TDD-Zyklus für alle Scnenarios jeder Story
Review durch Anwender, ggf. Anpassung der Stories
Anwender zufrieden?
Ja
Nein
Iteration

Zusammenfassung

- Fokus auf gutem Design und Änderungsfreundlichkeit
- Automatisierte Unit- und Akzeptanztests ermöglichen aggressives Refactoring
- Enge Zusammenarbeit mit dem Stakeholder sorgt für besseres Problemverständnis der Entwickler und führt zu Software, die den Erwartungen des Anwenders gerecht wird
- Festlegung eines gemeinsamen Vokabulars erleichtert die Kommunikation untereinander
- Keine unnötige Implementation nicht benötigter Features