HANSER

Agile Webentwicklung mit Rails

Dave Thomas, David Heinemeier Hansson

ISBN 3-446-40486-4

Vorwort

Weitere Informationen oder Bestellungen unter
http://www.hanser.de/3-446-40486-4 sowie im Buchhandel

1 Einführung

Durch die Befreiung des Gehirns von allen unnützen Tätigkeiten
wird es durch eine gute Notation in die Lage versetzt,
sich auf wichtigere Problemfelder zu konzentrieren …

Alfred Norman Whitehead

Bei Ruby on Rails handelt es sich um ein Framework, das die Entwicklung, Verbreitung und Wartung von Web-Anwendungen vereinfacht.

Natürlich versprechen alle Web-Frameworks dasselbe. Was also ist an Rails anders? Diese Frage lässt sich auf verschiedene Weisen beantworten.

Zum einen kann man einen näheren Blick auf die Architektur werfen. Im Laufe der Zeit sind die meisten Entwickler für ernsthafte Web-Anwendungen zu einer Model-View-Controller-Architektur (*MVC*, Modell-Darstellung-Steuerung) übergegangen. Davon versprechen sie sich einen saubereren Aufbau ihrer Anwendungen. (Das MVC-Konzept wird Gegenstand des nächsten Kapitels sein.) Java-Frameworks wie *Tapestry* und *Struts* basieren auf der MVC-Architektur. Auch bei Rails handelt es sich um ein MVC-Framework. Wenn Sie in Rails entwickeln, hat jedes Codeteil seinen Platz, und alle Codeteile Ihrer Anwendung interagieren auf eine standardisierte Weise miteinander. Sie beginnen sozusagen mit dem bereits vorbereiteten Skelett einer Anwendung.

Man kann die Frage aber auch in Bezug auf die verwendete Programmiersprache beantworten. Rails-Anwendungen werden in Ruby geschrieben, einer modernen, objektorientierten Scriptsprache. Ruby ist kurz und präzise, ohne kryptisch zu sein – Ideen lassen sich in Ruby natürlich und sauber ausdrücken. Dies führt dazu, dass Programme leicht zu schreiben sind und – nicht weniger wichtig – auch Monate später noch leicht zu lesen sind.

Daneben eignet sich Ruby für einen Programmierstil, die Lisp-Programmierern vertraut ist, anderen Programmierern aber etwas exotisch vorkommen dürfte. Die Sprache macht es einem leicht, Methoden zu erzeugen, die sich wie Erweiterungen der Syntax verhalten. Es gibt Leute, die das Metaprogrammierung nennen; wir nennen es einfach nur nützlich. Unsere Programme werden dadurch kürzer und leichter lesbar. Außerdem können wir auf diese Weise Aufgaben direkt innerhalb des Codes erledigen, für die normalerweise externe Konfigurationsdateien verwendet würden. So lässt sich viel leichter erkennen, was genau geschieht. Der folgende Beispielcode definiert die Modell-Klasse für ein Projekt. Um die Details brauchen Sie sich jetzt noch nicht zu kümmern. Denken Sie stattdessen darüber nach, wie viel Information in diesen wenigen Codezeilen steckt.

```
class Project < ActiveRecord::Base
  belongs_to              :portfolio
  has_one                 :project_manager
  has_many                :milestones
  has_and_belongs_to_many :categories
  validates_presence_of   :name, :description
  validates_acceptance_of :non_disclosure_agreement
  validates_uniqueness_of :key
end
```

Wir können auch die dahinter liegende Philosophie betrachten. Das Design von Ruby basiert auf einigen wesentlichen Konzepten, namentlich *DRY* und *Konvention vor Konfiguration*. DRY steht für *Don't Repeat Yourself*, also „wiederhole dich nicht": Jedes bisschen Wissen in einem System sollte nur an einer einzigen Stelle zum Ausdruck gebracht werden. Rails macht sich die Leistungsfähigkeit von Ruby zu Nutze, um diesem Prinzip Leben einzuhauchen. In einer Rails-Anwendung werden Sie nur wenige Wiederholungen vorfinden: Sie sagen das, was Sie zu sagen haben, an nur einer Stelle – die zudem häufig von den Konventionen der MVC-Architektur vorgegeben wird – und schreiten dann weiter voran.

Das zweite wesentliche Prinzip *Konvention vor Konfiguration* besagt, dass Rails für nahezu jeden Aspekt bei der Zusammenstellung Ihrer Anwendung über sinnvolle Vorgabewerte verfügt. Befolgen Sie die Konventionen, können Sie eine Rails-Anwendung schreiben, die mit weniger Code auskommt, als für die XML-Konfiguration einer typischen Web-Anwendung in Java benötigt wird. Wenn Sie sich einmal über die Konventionen hinwegsetzten müssen, macht Rails Ihnen auch das leicht.

Des Weiteren könnten wir die ganzen tollen Bestandteile ins Feld führen, die zu Rails gehören, etwa die integrierte Unterstützung von Web-Services, der Empfang eingehender E-Mails, AJAX (für in hohem Maße interaktive Web-Anwendungen), ein vollständiges Framework für Unit-Tests (einschließlich der transparenten Unterstützung von Pseudo-Objekten) sowie getrennte Umgebungen für Entwicklung, Tests und Produktion.

Wir könnten aber auch über die Code-Generatoren reden, die Rails mitbringt (oder die zusätzlich im Internet zur Verfügung stehen). Sie erzeugen das Code-Skelett in Ruby, das Sie dann nur noch mit der Anwendungslogik auszugestalten brauchen.

Schließlich unterscheidet sich Rails auch auf Grund seiner Herkunft: Rails ging aus einer real existierenden kommerziellen Anwendung hervor. Es hat sich gezeigt, dass sich ein Framework am besten erstellen lässt, wenn man die zentralen Themen einer bestimmten Anwendung ermittelt und diese dann in Form von allgemeinem Basiscode vorgefertigt anbietet. Wenn Sie nun damit beginnen, Ihre Rails-Anwendung zu entwickeln, befindet sich die Hälfte einer wirklich guten Anwendung bereits an Ort und Stelle.

Darüber hinaus verkörpert Rails noch etwas, das sich nur schwer beschreiben lässt. Irgendwie fühlt es sich einfach richtig an. Natürlich müssen Sie unseren Worten Glauben schenken, bis Sie selbst eine Rails-Anwendung geschrieben haben (was etwa in den nächsten 45 Minuten geschehen sollte…). Aber genau darum geht es in diesem Buch ja auch.

Daves Top-10-Gründe, warum man Rails mögen muss

1. Die Web-Entwicklung wird dadurch agil.
2. Ich kann Webseiten mit tollen Effekten erstellen, so wie es die coolen Kids tun.
3. Ich kann mich auf das Erstellen der Anwendung konzentrieren, ohne mich um das Grundgerüst kümmern zu müssen.
4. Meine Anwendungen bleiben wartbar, während sie weiter wachsen.
5. Ich kann die Anfragen meiner Kunden häufiger mit „ja“ beantworten.
6. Das Testen ist integriert (und einfach), also wird es auch genutzt.
7. Unmittelbares Feedback: Bearbeitung des Codes, Projekt aktualisieren, und schon erscheint die Änderung im Browser.
8. Metaprogrammierung bedeutet, dass ich auf einer wirklich hohen Stufe programmieren kann.
9. Die Code-Generatoren ermöglichen mir einen schnellen Beginn.
10. Kein XML!

1.1 Rails ist agil

Der Titel dieses Buches lautet *Agile Web-Entwicklung mit Rails*. Möglicherweise sind Sie nun etwas überrascht, dass Sie in diesem Buch keine Abschnitte finden, die sich ausdrücklich mit den agilen Praktiken X, Y und Z für die Rails-Programmierung beschäftigen.

Der Grund dafür ist sowohl einfach als auch subtil. Agilität ist Bestandteil der Struktur von Rails.

Lassen Sie uns einen Blick auf die Werte werfen, die das *Agile-Manifesto*[1] zum Ausdruck bringt. Sie werden in Form von vier Vorlieben beschrieben. Die agile Entwicklung bevorzugt folgende Vorgehensweisen:

- Individuen und Interaktionen kommen vor Prozessen und Werkzeugen.
- Funktionierende Software kommt vor umfassender Dokumentation.
- Zusammenarbeit mit dem Kunden kommt vor der Aushandlung von Verträgen.
- Reaktion auf Veränderungen kommt vor dem Befolgen eines Plans.

Bei Rails geht es voll und ganz um Individuen und Interaktionen. Es gibt keine schwergewichtigen Werkzeugsammlungen, keine komplizierten Konfigurationen und keine ausufernden Prozesse. Es gibt nur kleine Gruppen von Entwicklern, deren bevorzugte Editoren und Fragmente mit Ruby-Code. Dies führt zu einer ungeahnten Transparenz – das Tun der Entwickler spiegelt sich unmittelbar in dem wider, was der Kunde zu sehen bekommt. Es handelt sich hierbei seinem Wesen nach um einen interaktiven Prozess.

Rails hat nichts gegen Dokumentation einzuwenden. Im Gegenteil, Rails macht es einem sehr leicht, eine HTML-Dokumentation für seinen gesamten Code zu erstellen. Aber der

---

1 `http://agilemanifesto.org/`. Dave Thomas war einer der 17 Autoren dieses Dokuments.

Entwicklungsprozess unter Rails wird nicht von der Dokumentation bestimmt. Sie werden kein Rails-Projekt finden, dem eine 500 Seiten umfassende Spezifikation zu Grunde liegt. Stattdessen werden Sie eine Gruppe von Benutzern und Entwicklern vorfinden, die gemeinsam ihren Bedarf untersuchen und mögliche Lösungswege in Augenschein nehmen. Sie werden auf Gesamtlösungen stoßen, die Änderungen unterliegen, während sowohl die Entwickler als auch die Benutzer allmählich mit den Aufgaben vertrauter werden, die sie zu lösen haben. Sie werden einen Rahmen vorfinden, der schon zu einem frühen Zeitpunkt im Entwicklungszyklus eine laufende Software liefert. Diese Software mag zwar noch Ecken und Kanten haben, aber sie ermöglicht den Benutzern einen ersten Einblick in das, was Sie liefern werden.

Auf diese Weise fördert Rails die Zusammenarbeit mit dem Kunden. Sobald die Kunden sehen, wie schnell ein Rails-Projekt auf Änderungen reagieren kann, vertrauen sie darauf, dass das Entwicklerteam ihnen liefern kann, was sie wirklich brauchen, und nicht einfach, was zu Beginn festgeschrieben wurde. An Stelle von Konfrontationen wird es Sitzungen geben, auf denen es um das *Was wäre wenn?* geht.

Dies alles ist untrennbar mit der Vorstellung verbunden, in der Lage zu sein, schnell auf Änderungen reagieren zu können. Die zwingende, fast schon besessene Art, in der Rails das DRY-Prinzip befolgt, führt dazu, dass sich Änderungen an einer Rails-Anwendung auf weitaus weniger Code auswirken, als es die gleichen Änderungen in anderen Frameworks täten. Und da Rails-Anwendungen in Ruby geschrieben werden, worin sich Konzepte kurz und präzise ausdrücken lassen, lassen sich Änderungen tendenziell lokal begrenzen und einfach schreiben. Die starke Betonung auf Unit- und Funktionstests im Zusammenspiel mit der Unterstützung von so genannten *Fixtures* (Test-Datenstrukturen) und Pseudo-Objekten bietet Entwicklern das Sicherheitsnetz, das sie bei der Durchführung derartiger Änderungen benötigen. Mit einer geeigneten Reihe von Tests in der Hand sind Änderungen weniger Nerven aufreibend.

Statt ständig zu versuchen, einen Zusammenhang zwischen den Rails-Prozessen und den agilen Grundsätzen herzustellen, haben wir uns entschieden, das Framework für sich selbst sprechen zu lassen. Während Sie den Tutorial-Teil durcharbeiten, versuchen Sie sich Ihre Vorgehensweise bei der Entwicklung von Web-Anwendungen wie folgt vorzustellen: Sie arbeiten mit Ihren Kunden zusammen und legen gemeinsam die Prioritäten und die Lösungsansätze für die Einzelprobleme fest. Wenn Sie dann das mehr in die Tiefe gehende Referenzmaterial im hinteren Teil studieren, lenken Sie Ihren Blick darauf, wie die Rails zu Grunde liegende Struktur es Ihnen ermöglicht, die Bedürfnisse Ihrer Kunden schneller und mit weniger Umschweifen zu erfüllen.

Noch eine letzte Anmerkung zum Thema Agilität und Rails: Auch wenn es vermutlich unprofessionell ist, dies zu erwähnen, stellen Sie sich vor, wie viel Spaß Sie beim Programmieren haben werden.

1.2 Eine kleine Orientierungshilfe

Dieses Buch ist nun doch etwas umfangreicher geworden, als ursprünglich geplant. Rückblickend ist uns klar, dass wir in unserer Begeisterung eigentlich zwei Bücher geschrieben haben: ein Tutorial und ein Nachschlagewerk für Rails.

Die ersten beiden Teile dieses Buchs liefern eine Einführung in die Konzepte hinter Rails sowie ein ausführliches Beispiel – wir werden einen einfachen Online-Shop erstellen. Damit sollten Sie beginnen, wenn Sie ein Gefühl für die Rails-Programmierung gewinnen möchten. In der Tat scheinen die meisten Leute Gefallen daran zu finden, die Anwendung zu erstellen, während sie das Buch durcharbeiten. Wenn Sie sich das Tippen ersparen wollen, steht es Ihnen frei zu mogeln und den Quellcode einfach herunterzuladen.[2]

Im dritten Teil des Buchs wird detailliert auf alle Funktionen und Einrichtungen von Rails eingegangen. Dort finden Sie heraus, wie die verschiedenen Komponenten von Rails verwendet werden und wie Sie Ihre Rails-Anwendung effizient und sicher einsetzen.

Im Laufe des Buches werden Sie immer mal wieder auf verschiedene Einlassungen folgender Art stoßen:

- **Live-Code**
  Die meisten Code-Fragmente stammen aus vollständigen, lauffähigen Beispielen, die auch zum Herunterladen bereit stehen. Wenn ein Listing zum Herunterladen bereit steht, wird es wie folgt durch einen grau hinterlegten Hinweis auf die entsprechende Datei gekennzeichnet:

  Datei 209

  ```
  class SayController < ApplicationController
  end
  ```

  Wechseln Sie zur Verweisliste in Anhang C.3 und schlagen Sie dort die entsprechende Nummer nach, um den Namen und Speicherort der Datei ausfindig zu machen, die das Code-Fragment enthält.

- **David meint ...**
  Hin und wieder werden Sie auf einen *David meint* …-Einschub stoßen, in denen David Heinemeier Hansson Ihnen ein paar echte Knüller zu einigen speziellen Aspekten von Rails präsentieren wird – Grundprinzipien, Tricks, Ratschläge und mehr. Da er derjenige ist, der Rails erfunden hat, sollten Sie diese Abschnitte unbedingt lesen, wenn Sie ein Rails-Profi werden wollen.

- **Joe fragt ...**
  Unser frei erfundener Entwickler Joe taucht von Zeit zu Zeit auf, um Fragen zum gerade behandelten Stoff zu stellen. Wir versuchen dann, diese Fragen im Laufe des Texts zu beantworten.

---

2 Von *http://www.pragmaticprogrammer.com/titles/rails/code.html*.

Auch wenn Sie Ruby kennen müssen, um Rails-Anwendungen programmieren zu können, haben wir die Erfahrung gemacht, dass viele Leute, die dieses Buch lesen, Ruby und Rails gleichzeitig lernen. In Anhang A finden Sie eine (sehr) kurze Einführung in die Programmiersprache Ruby. Wenn Sie Ruby noch nicht kennen oder wenn Sie Ihre Kenntnisse kurz auffrischen wollen, dann sollten Sie sich den Anhang A durchlesen, bevor Sie weiterlesen. In diesem Buch gibt es ziemlich viel Code!

Bei diesem Buch handelt es sich nicht um ein Referenzwerk für Rails. Wir führen zwar die meisten der Module und ihrer Methoden entweder anhand von Beispielen oder im Text eingeflochten vor, aber Sie finden hier keine Hunderten von Seiten mit API-Auflistungen. Dafür gibt es einen guten Grund: Sie erhalten diese Dokumentation ohnehin bei jeder Installation von Rails – und das garantiert in einem aktuelleren Zustand als das Material in diesem Buch. Wenn Sie Ruby mit Hilfe von RubyGems installieren (was wir nur empfehlen können), starten Sie einfach den Dokumentationsserver (durch Eingabe des Befehls `gem_server`), und schon haben Sie Zugriff auf alle Rails-APIs, indem Sie mit Ihrem Browser die Adresse `http://localhost:8808` besuchen.

1.2.1 Versionen von Rails

Dieses Buch dokumentiert Rails in der Version 1.0, die seit Mitte 2005 zur Verfügung steht. Allerdings war dieser wichtige Meilenstein noch nicht erreicht, als die Vorbereitungen zu diesem Buch begannen. Um fristgerecht erscheinen zu können, beziehen sich die Beschreibungen der APIs in diesem Buch auf Rails 1.0. Der Code in diesem Buch wurde hingegen mit der Release 0.13 von Rails getestet, dem letzten Release vor der Version 1.0.

1.3 Danksagung

Dieses Buch entpuppte sich als aufwändiges Unterfangen. Ohne die enorme Unterstützung aus der Ruby- und Rails-Community wäre es nie zu Stande gekommen. Es ist schwierig, jeden aufzuführen, der zum Gelingen beigetragen hat – also, falls Du geholfen hast und Dein Name ist hier nicht genannt, haben wir das einfach übersehen.

Dieses Buch hatte eine unglaubliche Gruppe von Rezensenten – sie haben über 6 Megabyte nur an Kommentaren erzeugt. Unser herzlicher Dank geht daher an:

Alan Francis, Amy Hoy, Andreas Schwarz, Ben Galbraith, Bill Katz, Carl Dearmin, Chad Fowler, Curt Micol, David Rupp, David Vincelli, Dion Almaer, Duane Johnson, Erik Hatcher, Glenn Vanderburg, Gunther Schmidl, Henri ter Steeg, James Duncan Davidson, Johannes Brodwall, John Harechmak, John Johnson, Justin Forder, Justin Gehtland, Kim Shrier, Krishna Dole, Leon Breedt, Marcel Molina Jr., Michael Koziarski, Mike Clark, Miles K. Forrest, Raymond Brigleb, Robert Rasmussen, Ryan Lowe, Sam Stephenson, Scott Barron, Stefan Arentz, Steven Baker, Stian Grytøyr, Tait Stevens, Thomas Fuchs, Tom Moertel und Will Schenk.

Rails befand sich gerade im Entstehen, als dieses Buch zusammengetragen wurde. Daher haben die Leute aus der Kerntruppe von Rails viele Stunden damit verbracht, Daves Fragen mit viel Verständnis zu beantworten. (Sie verbrachten auch viel Zeit damit, mich zu quälen, indem sie die gerade erst fertig dokumentierten Stellen wieder änderten – aber das ist ein anderes Thema.) Ein aufrichtiges Dankeschön an:

> Jamis Buck (minam), Jeremy Kemper (bitsweat), Marcel Molina Jr., (noradio), Nicholas Seckar (Ulysses), Sam Stephenson (sam), Scott Barron (htonl), Thomas Fuchs (madrobby) und Tobias Lütke (xal).

Justin Forder leistete hervorragende Arbeit bei der Verbesserung von Daves farblosen Stylesheets für die Anwendung *Depot*.

Tausende von Leuten nahmen an dem Beta-Programm für die englische Fassung dieses Buchs teil. Ihnen allen vielen Dank dafür, dass sie diese Gelegenheit wahrgenommen haben. Hunderte von diesen Leuten nahmen sich die Zeit, Kommentare und Fehlerberichte zum Gelesenen einzureichen. Dieses Buch hat dadurch nur gewonnen.

Zu guter Letzt möchten wir den Leuten danken, die spezielle Kapitel zum Buch beisteuerten: Leon Breedt, Mike Clark, Thomas Fuchs und Andreas Schwarz.

Von Dave Thomas

Meine Familie hat mich die letzten acht Monate nicht zu Gesicht bekommen. Für ihre Geduld, Unterstützung und Liebe bin ich auf immer dankbar. Danke, Juliet, Zachary und Henry.

Von David Heinemeier Hansson

Marianne: Für die Geduld in schier endlosen Nächten, die ich an Rails saß.