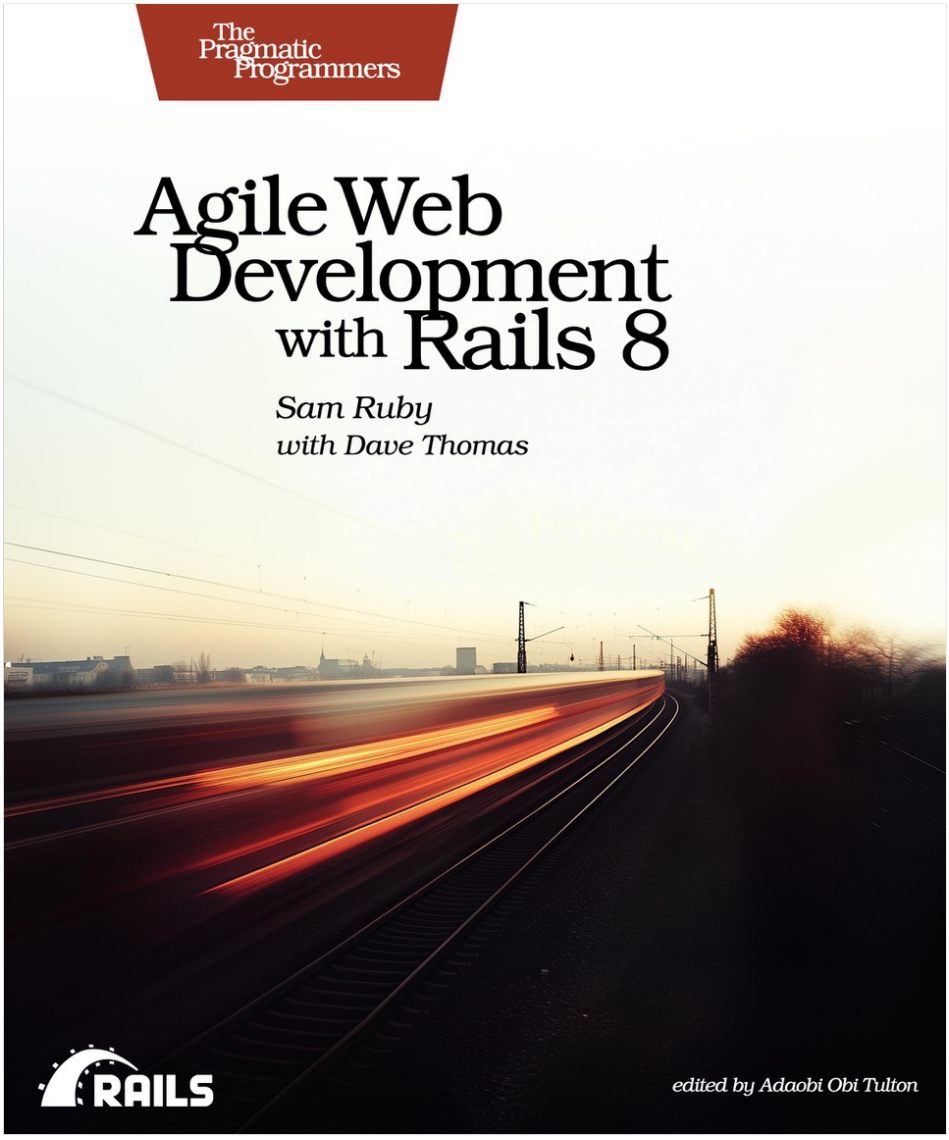

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Introduction

Ruby on Rails is a framework that makes it easier to develop, deploy, and maintain web applications. During the sixteen-plus years since its initial release, Rails went from being an unknown toy to a worldwide phenomenon. More importantly, it has become the framework of choice for the implementation of a wide range of applications.

Why is that?

Rails Simply Feels Right

A large number of developers were frustrated with the technologies they were using to create web applications. It didn't seem to matter whether they used Java, PHP, or .NET—there was a growing sense that their jobs were just too damn hard. And then, suddenly, along came Rails, and Rails was easier.

But easy on its own doesn't cut it. We're talking about professional developers writing real-world websites. They wanted to feel that the applications they were developing would stand the test of time—that they were designed and implemented using modern, professional techniques. So, these developers dug into Rails and discovered it wasn't just a tool for hacking out sites.

For example, *all* Rails applications are implemented using the model-view-controller (MVC) architecture. MVC isn't a new concept for web development—the earliest Java-based web frameworks (like Struts) base their design on it. But Rails takes MVC further: when you develop in Rails, you start with a working application, each piece of code has its place, and all the pieces of your application interact in a standard way.

Professional programmers write tests. And again, Rails delivers. All Rails applications have testing support baked right in. As you add functionality to the code, Rails automatically creates test stubs for that functionality. The framework makes it easy to test applications, and, as a result, Rails applications tend to get tested.

Rails applications are written in Ruby, a modern, object-oriented language. Ruby is concise without being unintelligibly terse. You can express ideas naturally and cleanly in Ruby code. This leads to programs that are easy to write and (just as important) easy to read months later.

Rails takes Ruby to the limit, extending it in novel ways that make our programming lives easier. Using Rails makes our programs shorter and more readable. It also allows us to perform tasks that would normally be done in external configuration files inside the codebase instead. This makes it far easier to see what's happening. The following code defines the model class for a project. Don't worry about the details for now. Instead, think about how much information is being expressed in a few lines of code:

```
class Project < ApplicationRecord
  belongs_to :portfolio

  has_one    :project_manager
  has_many   :milestones
  has_many   :deliverables, through: milestones

  validates  :name, :description, presence: true
  validates  :non_disclosure_agreement, acceptance: true
  validates  :short_name, uniqueness: true
end
```

A major philosophical underpinning of Rails that keeps code short and readable is the DRY principle, which stands for Don't Repeat Yourself (see *The Pragmatic Programmer, 20th Anniversary Edition [Hun19]*). Every piece of knowledge in a system should be expressed in one place. Rails uses the power of Ruby to bring that to life. You'll find little duplication in a Rails application; you say what you need to say in one place—a place often suggested by the conventions of the MVC architecture—and then move on. For programmers used to other web frameworks, where a simple change to the database schema could involve a dozen or more code changes, this was a revelation—and it still is.

From that principle, Rails is founded on the Rails Doctrine,[1] which is a set of nine pillars that explain why Rails works the way it does and how you can be most successful in using it. Not every pillar is relevant when just starting out with Rails, but one pillar in particular is most important: convention over configuration.

Convention over configuration means that Rails has sensible defaults for just about every aspect of knitting together your application. Follow the conventions, and you can write a Rails application using less code than a typical

1. http://rubyonrails.org/doctrine/

JavaScript application uses in JSON configuration. If you need to override the conventions, Rails makes that easy, too.

Developers coming to Rails find something else too. Rails doesn't merely play catch-up with the de facto web standards: it helps define them. And Rails makes it easy for developers to integrate features such as Hotwire, modern JavaScript frameworks, RESTful interfaces, and WebSockets into their code because support is built in. (And if you're not familiar with any of these terms, never fear—you'll learn what they mean as you proceed through the book.)

Rails was extracted from a real-world, commercial application. It turns out that the best way to create a framework is to find the central themes in a specific application and then package them in a generic foundation of code. When you're developing your Rails application, you're starting with half of a really good application already in place.

But there's something else to Rails—something that's hard to describe. Somehow, it feels right. Of course, you'll have to take our word for that until you write some Rails applications for yourself (which should be in the next forty-five minutes or so…). That's what this book is all about.

Rails Is Agile

The title of this book is *Agile Web Development with Rails 7*. You may be surprised to discover that we don't have explicit sections on applying agile practices *X*, *Y*, and *Z* to Rails coding. In fact, you won't find mention of many agile practices, such as Scrum or Extreme Programming, at all.

Over the years since Rails was introduced, the term *agile* has gone from being relatively unknown to being overhyped, to being treated as a formal set of practices, to receiving a well-deserved amount of pushback against formal practices that were never meant to be treated as gospel, to a return back to the original principles.

But it's more than that. The reason is both simple and subtle. Agility is part of the fabric of Rails.

Let's look at the values expressed in the Agile Manifesto (Dave Thomas was one of the seventeen authors of this document) as a set of four preferences:[2]

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation

2. http://agilemanifesto.org/

- Customer collaboration over contract negotiation
- Responding to change over following a plan

Rails is all about individuals and interactions. It involves no heavy toolsets, no complex configurations, and no elaborate processes, just small groups of developers, their favorite editors, and chunks of Ruby code. This leads to transparency; what the developers do is reflected immediately in what the customer sees. It's an intrinsically interactive process.

The Rails development process isn't driven by documents. You won't find 500-page specifications at the heart of a Rails project. Instead, you'll find a group of users and developers jointly exploring their need and the possible ways of answering that need. You'll find solutions that change as both the developers and the users become more experienced with the problems they're trying to solve. You'll find a framework that delivers working software early in the development cycle. This software might be rough around the edges, but it lets the users start to get a glimpse of what you'll be delivering.

In this way, Rails encourages customer collaboration. When customers see how quickly a Rails project can respond to change, they start to trust that the team can deliver what's required, not just what's been requested. Confrontations are replaced by "What if?" sessions.

The agile way of working that Rails encourages is tied to the idea of being able to respond to change. The strong, almost obsessive, way that Rails honors the DRY principle means that changes to Rails applications impact a lot less code than the same changes would in other frameworks. And since Rails applications are written in Ruby, where concepts can be expressed accurately and concisely, changes tend to be localized and easy to write. The deep emphasis on both unit and system testing, along with support for test fixtures and stubs during testing, gives developers the safety net they need when making those changes. With a good set of tests in place, changes are less nerve-racking.

Rather than constantly trying to link Rails processes to agile principles, we've decided to let the framework speak for itself. As you read through the tutorial chapters, try to imagine yourself developing web applications this way, working alongside your customers and jointly determining priorities and solutions to problems. Then, as you read the more advanced concepts that follow in Part III, see how the underlying structure of Rails can enable you to meet your customers' needs faster and with less ceremony.

One last point about agility and Rails—although it's probably unprofessional to mention this—think how much fun the coding will be!

Who This Book Is For

This book is for programmers looking to build and deploy web-based applications. This includes application programmers who are new to Rails (and perhaps even new to Ruby) as well as those who are familiar with the basics but want a more in-depth understanding of Rails.

We presume some familiarity with HTML, Cascading Style Sheets (CSS), and JavaScript—in other words, the ability to view source on web pages. You needn't be an expert on these subjects; the most you'll be expected to do is copy and paste material from the book, all of which can be downloaded.

The focus of this book is on the features and choices made by the Rails core team. More specifically, this book is for *users* of the Rails framework—people who tend to be more concerned about what Rails does, as opposed to how it does it or how to change Rails to suit their needs. Examples of topics not covered in this book include the following:

- Introduced in Rails 4, Turbolinks is a way to load pages more quickly by just loading markup.[3] If you want to know more about how Rails makes your pages load faster, follow that link. But should you instead be content with the knowledge that Rails makes pages load fast and not need to know more, that's OK too.
- Rails itself is highly hackable and extensible, but this book doesn't cover the concept of how to create your own Rails engine.[4] If that topic is of interest to you, we highly recommend *Crafting Rails 4 Applications [Val13]* as a follow-on to this book.
- The Rails team has chosen *not* to include plenty of features—such as user authentication—in the Rails framework itself. That doesn't mean that these features aren't important, but it generally does mean that no single solution is the obvious default for Rails users.

How to Read This Book

The first part of this book makes sure you're ready. By the time you're done with it, you'll have been introduced to Ruby (the language), you'll have been exposed to an overview of Rails, you'll have Ruby and Rails installed, and you'll have verified the installation with a simple example.

3. https://github.com/turbolinks/turbolinks/blob/master/README.md
4. http://guides.rubyonrails.org/engines.html

The next part takes you through the concepts behind Rails via an extended example: we build a simple online store. It doesn't take you one by one through each component of Rails (such as "here's a chapter on models, here's a chapter on views," and so forth). These components are designed to work together, and each chapter in this section tackles a specific set of related tasks that involve a number of these components working together.

Most folks seem to enjoy building the application along with the book. If you don't want to do all that typing, you can cheat and download the source code (a compressed tar archive[5] or a zip file).[6]

Be careful if you ever choose to copy files directly from the download into your application: if the timestamps on the files are old, the server won't know that it needs to pick up these changes. You can update the timestamps using the `touch` command on either MacOS or Linux, or you can edit the file and save it. Alternatively, you can restart your Rails server.

Part III, Rails in Depth, on page ?, surveys the entire Rails ecosystem. This starts with the functions and facilities of Rails that you'll now be familiar with. It then covers a number of key dependencies that the Rails framework makes use of that contribute directly to the overall functionality that the Rails framework delivers. Finally, we survey a number of popular plugins that augment the Rails framework and make Rails an open ecosystem rather than merely a framework.

Along the way, you'll see various conventions we've adopted:

Live code

Most of the code snippets we show come from full-length, running examples that you can download.

To help you find your way, if a code listing can be found in the download, you'll see a bar before the snippet (like the one here):

rails80/demo1/app/controllers/say_controller.rb

```
class SayController < ApplicationController
  def hello
  end

  def goodbye
  end
end
```

5. https://media.pragprog.com/titles/rails80/code/rails80-code.tgz
6. https://media.pragprog.com/titles/rails80/code/rails80-code.zip

The bar contains the path to the code within the download. If you're reading the ebook version of this book and your ebook viewer supports hyperlinks, you can click the bar and the code should appear in a browser window. Some browsers may mistakenly try to interpret some of the HTML templates as HTML. If this happens, view the source of the page to see the real source code.

And in some cases involving the modification of an existing file where the lines to be changed may not be immediately obvious, you'll also see some helpful little triangles to the left of the lines that you'll need to change. Two such lines are indicated in the previous code.

David says

Every now and then you'll come across a "David says" sidebar. Here's where David Heinemeier Hansson gives you the real scoop on some particular aspect of Rails—rationales, tricks, recommendations, and more. Because he's the fellow who invented Rails, these are the sections to read if you want to become a Rails pro.

Joe asks

Joe, the mythical developer, sometimes pops up to ask questions about stuff we talk about in the text. We answer these questions as we go along.

This book isn't meant to be a reference manual for Rails. Our experience is that reference manuals aren't the way most people learn. Instead, we show most of the modules and many of their methods, either by example or narratively in the text, in the context of how these components are used and how they fit together.

Nor do we have hundreds of pages of API listings. There's a good reason for this: you get that documentation whenever you install Rails, and it's guaranteed to be more up-to-date than the material in this book. If you install Rails using RubyGems (which we recommend), start the gem documentation server (using the `gem server` command), and you can access all the Rails APIs by pointing your browser at http://localhost:8808.

In addition, you'll see that Rails helps you by producing responses that clearly identify any error found as well as traces that tell you not only the point at which the error was found but also how you got there. You'll see an example on page ?. If you need additional information, peek ahead to Iteration E2: Handling Errors, on page ?, to see how to insert logging statements.

If you get really stuck, plenty of online resources can help. In addition to the code listings mentioned, you can find more resources on the Pragmatic Bookshelf site page for this book, including links to the book forum and errata.[7] The resources listed on these pages are shared resources. Feel free to post not only questions and problems to the forum but also any suggestions and answers you may have to questions that others have posted.

Let's get started! The first steps are to install Ruby and Rails and to verify the installation with a simple demonstration.

7. https://pragprog.com/titles/rails80/agile-web-development-with-rails-72/