Introduction

In late 2004, I was consulting at one of the big American auto makers, alongside a good friend of mine, Aslak Hellesoy.[1] It was a challenging assignment, chock full of difficult political situations, technical frustration, and crushing deadlines. Not your ordinary deadlines either; they were the type of deadline where the client would get fined a million dollars a day if we were late. The pressure was on!

In a moment of questionable judgment, the team agreed to base our continuous integration system on a pet project of Aslak's named DamageControl. It was a Ruby-based version of the venerable CruiseControl server produced by our employer, ThoughtWorks.

The problem was that DamageControl wasn't quite what you'd call a finished product. And like many other Ruby-related things, it just didn't work very well on Windows. Yet for some reason I can't quite remember today, we had to deploy it on an old Windows 2000 server that also hosted the StarTeam source repository (yikes!).

Aslak needed help—over the course of several weeks we pair-programmed extensively on both the application code of DamageControl and C-based internals of the Win32 process libraries for Ruby. At the time I had eight years of serious enterprise Java programming experience under my belt and a deep love of the brilliant IntelliJ IDE. I really cannot convey how much I hated Ruby at that point in my career.

So what changed? Well, for starters I eventually made it out of that stressful assignment alive, and took on a relatively easy assignment overseas out of the London office of ThoughtWorks. Within a month or so, Ruby caught my attention again, this time via considerable blogsphere excitement about an up-and-coming web framework named Ruby on Rails. I decided to give Ruby another chance. Perhaps it wasn't so bad after all? I quickly built an innovative social networking system for internal use at ThoughtWorks.

That first Rails experience, over the course of a few weeks in February 2005, was life-altering. All of the best practices I had learned over the years about building web apps had been distilled into a single framework, written in some of the most elegant and concise code that I had ever seen in my life. My interest in Java died a sudden death (although it took me almost another year to stop using IntelliJ). I began avidly blogging about Ruby and Rails and evangelizing it heavily both inside and out of ThoughtWorks. The rest, as they say, is history.

As I write this in 2007, the Rails business I pioneered at ThoughtWorks accounts for almost half of their global revenue, and they've established a large product division churning out Ruby-based commercial software. Among them is CruiseControl.rb, which I suspect is what Aslak wanted to build all along—it has the honor of being the official continuous integration server of the Ruby on Rails core team.

Ruby and Rails

Why do experienced enterprise folks like me fall in love with Ruby and Rails? Given a set of requirements to fulfill, the complexity of solutions created using Java and Microsoft technology is simply unacceptable. Excess complexity overwhelms individual understanding of the project and dramatically increases communications overhead for the team. The emphasis on following design patterns, as well as the obsession with performance, wears down the pure joy of application development with those platforms.

> There's no peer pressure to do anything in the Rails community. DHH (David Heinemeier Hansson) picked a language that made him happy. Rails was born from code that he felt was beautiful. That kind of set the tone for the Rails community. So much about Rails is subjective. People either "get it" or they don't. But there's no malice from those who do towards those who don't, just gentle encouragement.
> —Pat Maddox

Ruby is beautiful. Coding in Ruby is beautiful. Everyone I've known who makes the move into Ruby says they are happier than before. For this reason more than any other, Ruby and Rails are shaking up the status quo, especially in enterprise computing. Prior to getting involved with Rails, I was accustomed to working on projects based on fuzzy requirements bearing no relation to real-world needs. I was tired of mind-boggling arrays of competing frameworks to choose from and integrate, and I was tired of ugly code.

In contrast, Ruby is a beautiful, dynamic, high-level language. Ruby code is easier to read and write because it more closely maps to the problem domains we tackle, in a style that is closer to human language. The enhanced readability yields many benefits, both short-term and long-term, as code moves into production and must be understood by maintenance programmers.

My experience has shown me that programs written in Ruby have fewer lines of code than comparable programs in Java and C#. Smaller codebases are easier to maintain and long-term maintenance is widely cited as the biggest cost of successful software projects. Smaller codebases are also faster to debug when things go wrong, even without fancy debugging tools.

The Rise of Rails and Mainstream Acceptance

In ways similar to the Agile movement that helped birth it, Rails is all about catering to our needs as application developers—not as software engineers, and certainly not as computer scientists. By aggressively attacking unneeded complexity, Rails shines brightest in the people-oriented aspects of development that really matter to the ultimate success of our projects. We have fun when we're programming in Rails, and that makes us want to succeed!

The tools and technical infrastructure provided by Rails are comprehensive, encouraging us to focus on delivering business value. Ruby's Principle of Least Surprise is embodied in the simple and elegant design of the Rails. Best of all, Rails is completely free open-source software, which means that when all else fails, browsing the source code can yield answers to even the most difficult of problems.

David has occasionally mentioned that he is not particularly excited about Rails reaching mainstream acceptance, because the competitive edge enjoyed by early adopters would be diminished. Those early adopters have primarily been individuals and small groups of web designers and programmers, with legions of them coming out of the PHP world.

Enterprise Adoption

Call me an idealist if you like, but I believe that even enterprise developers at large and conservative corporations will act to become more effective and innovative at their jobs if they are given the tools and encouragement to do so. That's why it seems like they're jumping on the Rails bandwagon in ever-greater numbers with every year that passes.

Perhaps enterprise developers will ultimately be the most vocal and enthusiastic adopters of Ruby and Rails, because right now they are the ones who as a group stand to lose the most from the status quo. They're consistently the targets of mass layoffs and misguided outsourcing efforts, based on assumptions such as "specification is more important than implementation" and "implementation should be mechanical and trivial."

Is specification actually more important than the implementation? Not for most projects. Is implementation of all but the simplest projects trivial? Of course not! There are significant underlying reasons for the difficulties of software development, especially in enterprise environments:[2]

- Hard-to-understand legacy systems.
- Highly complex business domains such as investment banking.
- Stakeholders and business analysts who don't actually know what they want.
- Managers resistant to productivity because it shrinks their yearly budgets.
- End users who actively sabotage your project.
- Politics! Sticking your head out means worrying that it'll get chopped off.

As a consultant to Fortune 1000 companies, I lived and breathed those situations on an everyday basis for almost 10 years, eventually stumbling upon a powerful concept. There is a viable alternative to playing it safe, an alternative so powerful that it transcends politics and is guaranteed to bring you acclaim and open new doors of opportunity.

That alternative is being exceptional! It starts with productivity. I'm talking about becoming so obviously effective at your job that nobody will ever be able to scapegoat you, to the extent that it would be political suicide to try. I'm talking about cultivating practices that make your results stand out so brilliantly that they bring tears of joy to even the most cynical and hardened stakeholders of your projects. I mean regularly having time to polish your applications to a state of wonderfulness that consistently breeds passionate end users.

By simply being exceptional, you can be that individual (or team) that keeps clients happy and paying their invoices on time, or that survives layoffs year after year, because the decision-makers say: "Oh, there's no way we can afford to lose them."

Let me pause for a second. I wouldn't blame you for regarding my words with skepticism, but none of what I'm saying is idle hype. I'm describing my own life since moving to Ruby on Rails. This book is intended to help you make Ruby on Rails your secret (or not-so-secret) weapon for thriving in the treacherous world of software development.

Delivering Results

My contributors and I draw on our collective experience and industry knowledge to show you how to deliver practical results using Ruby on Rails on your projects, giving you the ammunition needed to justify your choice of technology and even defeat objections that will undoubtedly come your way. Since we know there are never any silver bullets, we'll also warn you about situations where choosing Rails would be a mistake.

Along the way, we'll analyze each of the components of Rails in depth and discuss how to extend them when the need arises. Ruby is an extremely flexible language, which means there are myriad ways to customize the behavior of Rails yourself. As you will learn, the Ruby way is all about giving you the freedom to find the optimal solution to the problem at hand.

As a reference work, this book functions as a guide to the Rails API and the wealth of Ruby idioms, design approaches, libraries, and plugins useful to the Ruby on Rails enterprise developer.

About Opinionated Software

Before going on, I should mention that part of what makes Rails exceptional is that it is opinionated software, written by opinionated programmers. Likewise, this is an opinionated book, written by opinionated writers.

Here are some of the opinions about development that influence this book. You don't have to agree with all of them—just be aware of their influence:

- Developer motivation and productivity trump all other factors for project success.
- The best way to keep motivated and productive is to focus on delivering business value.
- Performance means "executing as fast as possible, on a given set of resources."
- Scalability means "executing as fast as needed, on as many resources as needed."
- Performance is irrelevant if you can't scale.
- If you can scale cheaply, milking every ounce of performance from your processors should never be your first priority.
- Linking scalability to choice of development tools is a pervasive mistake in the industry and most software does not have extreme scalability requirements.
- Performance *is* related to choice of language and tools because higher-level languages are easier to write and understand. There is wide consensus that the performance problems in most applications are caused by poorly written application code.
- Convention over configuration is a better way to write software. Huge XML configuration files must be eliminated!
- Code portability, the ability to take code and run it on a different hardware platform, is not particularly important.

- It's better to solve a problem *well* even if the solution only runs on one platform. Portability is irrelevant if your project fails.
- Database portability, the ability to run the same code on different relational database systems is rarely important and is almost never achieved.
- Presentation is very important, even for small projects. If your application looks bad, everyone will assume it is written badly.
- Allowing technology to dictate the approach to solving a business problem is usually a bad idea; however, that advice shouldn't be used as an excuse to stick with inferior technology.
- The benefits of generalized application components are dubious. Individual projects usually have very particular business needs and wildly different infrastructure requirements, making parameterized reuse very difficult to achieve in practice.

Phew, that's a lot of opinions. But don't worry, *The Rails Way* is primarily a reference work, and this list is the only one of its kind in the book. Speaking of which….

About This Book

This book is not a tutorial or basic introduction to Ruby or Rails. It is meant as a day-to-day reference for the full-time Rails developer. At times we delve deep into the Rails codebase to illustrate why Rails behaves the way that it does, and present snippets of actual Rails code. The more confident reader might be able to get started in Rails using just this book, extensive online resources, and his wits, but there are other publications that are more introductory in nature and might be a wee bit more appropriate for beginners.

I am a fulltime Rails application developer and so is every contributor to this book. We do not spend our days writing books or training other people, although that is certainly something that we enjoy doing on the side.

I started writing this book mostly for myself, because I hate having to use online documentation, especially API docs, which need to be consulted over and over again. Since the API documentation is liberally licensed (just like the rest of Rails), there are a few sections of the book that reproduce parts of the API documentation. In practically all cases, the API documentation has been expanded and/or corrected, supplemented with additional examples and commentary drawn from practical experience.

Hopefully you are like me—I really like books that I can keep next to my keyboard, scribble notes in, and fill with bookmarks and dog-ears. When I'm coding, I want to be able to quickly refer to both API documentation, in-depth explanations, and relevant examples.

Book Structure

I attempted to give the material a natural structure while meeting the goal of being the best-possible Rails reference book. To that end, careful attention has been given to presenting holistic explanations of each subsystem of Rails, including detailed API information where appropriate. Every chapter is slightly different in scope, and I suspect that Rails is now too big a topic to cover the whole thing in depth in just one book.

Believe me, it has not been easy coming up with a structure that makes perfect sense for everyone. Particularly, I have noted surprise in some readers when they notice that `ActiveRecord` is not covered first. Rails is foremost a web framework and at least to me, the controller and routing implementation is the most unique, powerful, and effective feature, with `ActiveRecord` following a close second.

Therefore, the flow of the book is as follows:

- The Rails environment, initialization, configuration, and logging
- The Rails dispatcher, controllers, rendering, and routing
- REST, Resources, and Rails
- `ActiveRecord` basics, associations, validation, and advanced techniques
- `ActionView` templating, caching, and helpers
- Ajax, Prototype, and Scriptaculous JavaScript libraries, and RJS
- Session management, login, and authentication
- XML and `ActiveResource`
- Background processing and `ActionMailer`
- Testing and specs (including coverage of RSpec on Rails and Selenium)
- Installing, managing, and writing your own plugins
- Rails production deployment, configurations, and Capistrano

Sample Code and Listings

The domains chosen for the code samples should be familiar to almost all professional developers. They include time and expense tracking, regional data management, and blogging applications. I don't spend pages explaining the subtler nuances of the business logic for the samples or justify design decisions that don't have a direct relationship to the topic at hand. Following in the footsteps of my series colleague Hal Fulton and *The Ruby Way*, most of the snippets are not full code listings—only the relevant code is shown. Ellipses (…) denote parts of the code that have been eliminated for clarity.

Whenever a code listing is large and significant, and I suspect that you might want to use it verbatim in your own code, I supply a listing heading. There are not too many of those. The whole set of code listings will not add up to a complete working system, nor are there 30 pages of sample application code in an appendix. The code listings should serve as inspiration for your production-ready work, but keep in mind that it often lacks touches necessary in real-world work. For example, examples of controller code are often missing pagination and access control logic, because it would detract from the point being expressed.

Plugins

Whenever you find yourself writing code that feels like *plumbing*, by which I mean completely unrelated to the business domain of your application, you're probably doing too much work. I hope that you have this book at your side when you encounter that feeling. There is almost always some new part of the Rails API or a third-party plugin for doing exactly what you are trying to do.

As a matter of fact, part of what sets this book apart is that I never hesitate in calling out the availability of third-party plugins, and I even document the ones that I feel are most crucial for effective Rails work. In cases where a plugin is better than the built-in Rails functionality, we don't cover the built-in Rails functionality (pagination is an example).

An average developer might see his productivity double with Rails, but I've seen serious Rails developers achieve gains that are much, much higher. That's because we follow the Don't Repeat Yourself (DRY) principle religiously, of which Don't Reinvent The Wheel (DRTW) is a close corollary. Reimplementing something when an existing implementation is *good enough* is an unnecessary waste of time that nevertheless can be very tempting, since it's such a joy to program in Ruby.

Ruby on Rails is actually a vast ecosystem of core code, official plugins, and third-party plugins. That ecosystem has been exploding rapidly and provides all the raw technology you need to build even the most complicated enterprise-class web applications. My goal is to equip you with enough knowledge that you'll be able to avoid continuously reinventing the wheel.

Recommended Reading and Resources

Readers may find it useful to read this book while referring to some of the excellent reference titles listed in this section.

Most Ruby programmers always have their copy of the "Pickaxe" book nearby, *Programming Ruby* (ISBN: 0-9745140-5-5), because it is a good language reference. Readers interested in really understanding all of the nuances of Ruby programming should acquire *The Ruby Way, Second Edition* (ISBN: 0-6723288-4-4).

I highly recommend *Peepcode Screencasts*, in-depth video presentations on a variety of Rails subjects by the inimitable Geoffrey Grosenbach, available at http://peepcode.com.

Regarding David Heinemeier Hansson a.k.a. DHH

I had the pleasure of establishing a friendship with David Heinemeier Hansson, creator of Rails, in early 2005, before Rails hit the mainstream and he became an *International Web 2.0 Superstar*. My friendship with David is a big factor in why I'm writing this book today. David's opinions and public statements shape the Rails world, which means he gets quoted a lot when we discuss the nature of Rails and how to use it effectively.

David has told me on a couple of occasions that he hates the "DHH" moniker that people tend to use instead of his long and difficult-to-spell full name. For that reason, in this book I try to always refer to him as "David" instead of the ever-tempting "DHH." When you encounter references to "David" without further qualification, I'm referring to the one-and-only David Heinemeier Hansson.

Rails is by and large still a small community, and in some cases I reference core team members and Rails celebrities by name. A perfect example is the prodigious core-team member, Rick Olson, whose many useful plugins had me mentioning him over and over again throughout the text.

Goals

As stated, I hope to make this your primary working reference for Ruby on Rails. I don't really expect too many people to read it through end to end unless they're expanding their basic knowledge of the Rails framework. Whatever the case may be, over time I hope this book gives you as an application developer/programmer greater confidence in making design and implementation decisions while working on your day-to-day tasks. After spending time with this book, your understanding of the fundamental concepts of Rails coupled with hands-on experience should leave you feeling comfortable working on real-world Rails projects, with real-world demands.

If you are in an architectural or development lead role, this book is not targeted to you, but should make you feel more comfortable discussing the pros and cons of Ruby on Rails adoption and ways to extend Rails to meet the particular needs of the project under your direction.

Finally, if you are a development manager, you should find the practical perspective of the book and our coverage of testing and tools especially interesting, and hopefully get some insight into why your developers are so excited about Ruby and Rails.

Prerequisites

The reader is assumed to have the following knowledge:

- Basic Ruby syntax and language constructs such as blocks
- Solid grasp of object-oriented principles and design patterns
- Basic understanding of relational databases and SQL
- Familiarity with how Rails applications are laid out and function
- Basic understanding of network protocols such as HTTP and SMTP
- Basic understanding of XML documents and web services
- Familiarity with transactional concepts such as ACID properties

As noted in the section "Book Structure," this book does not progress from easy material in the front to harder material in the back. Some chapters do start out with fundamental, almost introductory material, and push on to more advanced coverage. There are definitely sections of the text that experienced Rails developer will gloss over. However, I believe that there is new knowledge and inspiration in every chapter, for all skill levels.

Required Technology

A late-model Apple *MacBookPro* with 4GB RAM, running OSX 10.4. Just kidding, of course. Linux is pretty good for Rails development also. Microsoft Windows—well, let me just put it this way—your mileage may vary. I'm being nice and diplomatic in saying that. We specifically *do not* discuss Rails development on Microsoft platforms in this book.[3] To my knowledge, most working Rails professionals develop and deploy on non-Microsoft platforms.

References

1. Aslak is a well-known guy in Java open-source circles, primarily for writing XDoclet.

2. I'm not saying startups are much easier, but they usually have less dramatic problems.

3. For that information, try the Softies on Rails blog at http://softiesonrails.com.