The Art of Rails®

Edward Benson

WILEY

Wiley Publishing, Inc.

The Art of Rails®

Acknowledgments xi
Introduction xix

Chapter 1: Emergence(y) of the New Web 1
Chapter 2: The Rails Concept 21
Chapter 3: The Server as an Application 45
Chapter 4: Getting the Most from M, V, and C 59
Chapter 5: Beautiful Web APIs 89
Chapter 6: Resources and REST 115
Chapter 7: The Five Styles of AJAX 139
Chapter 8: Playing with Blocks 167
Chapter 9: Mixins and Monkey Patching 197
Chapter 10: Code That Writes Code (That Writes Code) 225
Chapter 11: How I Learned to Stop Worrying and Love the Schema 253
Chapter 12: Behavior-Driven Development and RSpec 275

Index 297

The Art of Rails®

Edward Benson

WILEY

Wiley Publishing, Inc.

The Art of Rails®

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2008 by Edward Benson

Published by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-0-470-18948-1

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data

Benson, Edward, 1983-
The art of Rails / Edward Benson.
p. cm.
Includes index.
ISBN 978-0-470-18948-1 (pbk.)
1. Web site development. 2. Ruby on rails (Electronic resource) 3. Ruby (Computer program language) I. Title.
TK5105.888.B4524 2008
005.1'17 — dc22

2008012006

For Grace

About the Author

Edward Benson is a Staff Scientist with BBN Technologies in Arlington, Virginia. Edward's work at BBN includes the design and implementation of agent-based logistics and data processing architectures and semantically-enabled data recording and processing environments (often called the ''Semantic Web''). He is a member of the IEEE and has published papers on both grid and agent computing techniques. Edward is an experienced web applications developer and a co-author of *Professional Rich Internet Applications*, also from Wrox. Edward received his B.S. in Computer Science *summa cum laude* from the University of Virginia.

Credits

Acquisitions Editor
Jenny Watson

Development Editors
Lori Cerreto
John Sleeva

Technical Editor
Dana Moore

Production Editor
Daniel Scribner

Copy Editor
Susan Christophersen

Editorial Manager
Mary Beth Wakefield

Production Manager
Tim Tate

Vice President and Executive Group Publisher
Richard Swadley

Vice President and Executive Publisher
Joseph B. Wikert

Project Coordinator, Cover
Lynsey Stanford

Proofreader
Jen Larsen, Word One

Indexer
Robert Swanson

Acknowledgments

My heartfelt thanks go out to the team at John Wiley & Sons — especially John Sleeva and Lori Cerreto — and to my colleague and technical editor Dana Moore. Your insight, feedback, and hard work have been paramount to making the book what it is. Thanks also to Carol Long at Wiley for believing in my ideas enough to convince me that I should write them down as a book proposal.

I could not have completed this book without the help and love of my fiancée, Grace. At times, writing can be a painstakingly slow and all-consuming process. Her never-ending encouragement pushed me to write each chapter with the enthusiasm that prompted me to begin the book in the first place.

Thank you to my parents and brother for their support; to my cousin Emily for her fantastic cartoon renderings of W. Web — they didn't make it into the book, so now we'll have to lobby Wiley together for *The Art of Rails, Animated Edition*; and to Robert Hiedemann for his grandfatherly advice and encouragement to make education a part of my life.

Thank you to the many friends at BBN Technologies who provided help and advice to make this book happen: Pete Pflugrath and Bud Sichler for being understanding of my time constraints and being flexible with my work schedule; Troy Self for providing feedback on early chapter drafts; Rob Battle for being a sounding board for ideas; and Doug Reid, Dave Kolas, Steve Allen, Jeremy Learner, Andrew Perez-Lopez, Tony Stein, Jonathan Nilsson, and Greg Joiner for providing their thoughts, humor, and feedback on ideas over the course of writing. (Steve's reaction to the title: '''The Art of Rails'? Who do you think you are, Donald Knuth?'')

Several people on the Internet were kind enough to contribute their advice and code bits. Thank you to Rob Malda for his thoughts on the early days of web application development; Elaine Wherry of Meebo for her encouragement and feedback on the AJAX chapter; and Scott Raymond for allowing me to use his RSS 2.0 template for the XML Builder. Thank you, finally, to the many open source developers and Rails bloggers whose hard labors have advanced web development to the discipline that it is today.

Contents

Acknowledgments xi

Introduction xix

Chapter 1: Emergence(y) of the New Web 1

Rails, Art, and the New Web 2

Art and Engineering 3

The New Web 3

The Truth about Web Applications 5

Patient History: The World Wide Web 5

From Documents to Interfaces 8

The Decline of Semantics 9

Hello, Web Applications 12

Emergence of the New Web 19

Chapter 2: The Rails Concept 21

One Part Framework 22

The Configuration View 23

The Code View 25

The Process View 26

One Part Language 27

Models on Rails 28

Web-Specific Tasks 30

JavaScript 31

Plug-Ins 32

Two Parts Mindset 32

Web Sites Are MVC Applications 32

Web Applications Are Ecosystems 33

Convention over Configuration 34

A Little Assumption Goes a Long Way 34

Aesthetics Matter 36

Restraint Can Be Liberating 37

You Repeat Yourself Too Much 38

Testing Isn't a Choice 40
The Web Is a Set of Resources, Not Services 41
Summary 42
No Silver Bullet 43
Optimize Your Coding Before You Optimize Your Code 43

Chapter 3: The Server as an Application 45

Model-View-Controller: The Abbreviated Version 47
MVC and the Web 48
The MVC Design Process 50
The View Is Your Spec 51
Example: Social Networking for Cooks 51
Managing the Application Lifecycle 55
Think Agile, Not Engineered 55
Think Controlled, Not Organic 56
Beware of Open-Heart Surgery 56
Summary 57

Chapter 4: Getting the Most from M, V, and C 59

The Best API Documentation Is Free 60
The Model 61
Model Objects Should Understand Each Other 62
Use Exceptions to Make Code Cleaner 64
Mapping from Rows to Objects 67
Polymorphic Associations 68
The World Outside of ActiveRecord 70
The View 70
The Variable Problem 70
Rails-Style JavaScript 72
Partials as Atoms and Molecules 73
Picking the Proper Partials 74
The View Isn't Just HTML 77
The Controller 77
Reusable CRUD 78
Rails Scaffolding Is a Bunch of CRUD 79
Dealing with Two-Step Actions 80
Knowing When to Outsource 83
Knowing When to Refactor 87
Conclusion 88

Chapter 5: Beautiful Web APIs 89

Two Big Ideas for Web APIs 91
The New URL: Addressing Concepts, Not Files 91
The Application Is the API 93
Routing 93
Anatomy of the Web API Call 96
Overlaying the API 97
The respond_to Method 97
Writing a Non-HTML Result 99
Adding Custom MIME Types 103
Registering Types with Rails 104
Creating Your Own MIME Type 105
API-Metering, the Rails Way 105
Authenticating the User 105
The Metering Algorithm 107
Applying Metering via Filters 108
What about SOAP/XML-RPC Services? 109
Summary 112

Chapter 6: Resources and REST 115

A Web of Resources 116
Identifying Resources 117
Talking About Resources 117
Representing Resources 118
Representational State Transfer 118
HTTP: The Resource CRUD 119
Defining an Application in Terms of Resources 122
Communicating with the Client: Resources as Your API 126
Put Another Way: The Network Is the Computer 127
REST and Rails 128
Mapping Resources in the Router 129
But It's the Real World: Named Routes are Still Needed 130
Resource Scaffolds 133
Nested Resources 133
Singleton Resources versus Regular Resources 137
Summary 138

Chapter 7: The Five Styles of AJAX 139

The Big Secrets 141
AJAX Isn't Necessarily the Hard Part 141

AJAX Introduces Tough Design Issues 141
You Have Your Pick of JavaScript Frameworks, Even in Rails 143
The Five Styles of AJAX 144
Proxy Style 146
Partial Style 148
Puppet Style 149
Compiled-to-Web Style 151
In-Place Application Style 153
AJAX as Just Another API 155
Rails-Style AJAX 157
Partial-Style AJAX Controllers (and AJAX CRUD) 157
Puppet-Style AJAX Controllers (and RJS) 160
Elegant Degradation 162
Moving Backward from the Partial-Style 163
Moving Backward from Rich User Interfaces 164
Summary 165

Chapter 8: Playing with Blocks 167

The Block Mindset 169
Comparing Methods, Procs, and Blocks 173
Methods 173
Procs 177
Blocks 179
Moving Between Blocks and Procs 180
The Big Scope Experiment 181
Experiment 1: Blocks Are Affected by Changes in Their Source Environment 182
Experiment 2: Blocks Can Affect the Environment from Which They Came 184
Block Patterns, Blocks in Rails 186
Iteration 186
Aspect-Oriented Programming 187
Building Output in HTML and XML 192
Dual-Use Functions 194
The Callback 194
Summary 195

Chapter 9: Mixins and Monkey Patching 197

Mixins 199
Organizing Code into Modules 199
Methods in Modules 201
Mixing Modules into Classes 202
Mixins in Rails 205

Monkey Patching **210**
Eval: The Interpreter's Back Door 211
Eval's Two Siblings 213
Good Monkey Patching Technique 219
Summary **222**

Chapter 10: Code That Writes Code (That Writes Code) **225**

Dynamic Code and DSLs Revisited **227**
Code-Writing Macros **228**
Creating Methods on the Fly with define_method 228
define_method Example: The Pentagon and the Kremlin 229
Scope and define_method 230
Using define_method for Rails Macros 232
Macro Summary 234
Calling Methods That Don't Exist: Objects That Adapt to the Way You Use Them **234**
Some Basic Examples 236
Example: A Shortcut for Array.each 237
Beware of Catching Everything 240
method_missing Patterns 241
Implementing method_missing Patterns 242
Reflection **248**
Variables and Constants 249
Methods 250
Modules 251
Summary **252**

Chapter 11: How I Learned to Stop Worrying and Love the Schema **253**

Bringing the Database into the Picture: The LAMP Stack **254**
Thinking in Migrations **256**
Writing Migrations 257
Performing Schema Migrations 259
Team Schema Development **260**
Seeding Data for Production **262**
Small Datasets: Seed Migrations 262
Medium Datasets: Seed Fixtures 263
Large Datasets: Dumpfiles 264
When a Database Isn't Enough **266**
Model Object Hierarchies 266
Storing Lists, Hashes, and Other Fun Things 271
Custom Getters and Setters 272
Summary **273**

Chapter 12: Behavior-Driven Development and RSpec 275

Behavior-Driven Development 276
RSpec: BDD for Ruby and Rails 279
The Spec Development Cycle 279
Writing the Spec 280
Implementing Examples 281
Matchers 282
Custom Matchers 285
Before and After 287
An Example Trip through the Development Cycle 288
Part 1: Writing The Story 288
Part 2: Writing the Specs 288
Part 3: Initializing and Writing a Basic Test 290
Part 4: Writing Behavior Tests That Motivate Development 291
Part 5: Completing the Behavioral Test Implementations 293
But Wait, There's More 295
Summary 296

Index 297

Introduction

There is a certain state of mind, a certain transient condition that arises, where everything seems to resonate and effort becomes effortless. Athletes call it being in the zone, some others call it flow. Flow has nothing to do with triumph or accomplishment; it isn't the product of your labors. Flow is the merging of a watchmaker and his watch or an artist and her paints.

The dot-com bust was a confusing time for web development, but rising from the burst dreams of instant wealth, something strange and exciting happened. The web development community as a whole reached a kind of flow. In a world filled with duct-tape solutions and proprietary formats, suddenly web developers were clamoring for standards compliance, for elegance and simplicity. And it wasn't just to fend off browser compatibility issues, but because the code looked *beautiful*.

Through the fits and starts, the competing ideas, and the explosion of development frameworks that followed, an identity began to emerge. This identity is as much a philosophical statement about what the web could be as it is a technical statement about how to accomplish those goals. This identity is still emerging, and there are still many problems to be solved, but one thing is now certain: web application development has come of age as a rich discipline of programming that stands up on its own.

Ruby on Rails is just one part of this much larger story, but in many ways it is the symbol of this coming of age. Rails challenged the web development community to rethink what it meant to build a web application. It provided an entire application ecosystem when most developers were embedding their code inside HTML files. It made unit testing not only easy but also cool, and did so at a time when debugging web applications was, at best, a black art. It introduced a new generation of web developers to the ideas of meta-programming and domain-specific languages. And, most of all, it found the voice of the change that was taking place: that the web provides a natural and elegant architecture on which to write applications if only we can create the right metaphors to harness it.

What Is the Art of Rails?

Any programmer knows that an API is only half the story, and with Rails this is especially true. Good Rails development, and good web development, is much more about the design choices you make than the framework you have at your disposal. I wrote this book as an attempt to create the resource I wish I had after settling into Rails development — to pick up where the API leaves off and explain how to take good Rails code and turn it into beautiful Rails code: simple, effective, reusable, evolvable code.

This book is meant to take your Rails development to the next level, and in doing so, it cuts across a wide range of topics. Each of these topics is selected to highlight a particular part of the design and development process that can make the difference between just using the Rails framework and achieving a state of flow with the framework. Throughout the book, the focus is on *the way you code* rather than the mechanics of coding. The book is divided into clusters of chapters that represent the themes listed in the following sections.

Development Philosophy of the New Web

Chapters 1 and 2 discuss the changes in style and focus that have taken place since the web's inception. Chapter 1 presents a brief history of the evolution of web development, with a focus on interpreting that history as it relates to the changes that impact our lives as web developers today. Many aspects of the modern web application architecture were shaped by earlier programming styles that can still play invaluable roles in analyzing your design and code. This chapter gives names to some of these styles, such as code-first development and document-first development, to cast light on some of the design decisions that we are faced with today.

Chapter 2 presents Ruby on Rails as ''one part framework, one part language extension, and two parts state of mind.'' It picks apart Rails from each of these angles so that you can see how it all fits together mechanically, stylistically, and philosophically. When you are starting out with Rails, just understanding the mechanics of writing a Rails application is sufficient, but as you advance in your skill, a deeper understanding of how the framework fits together is necessary. This holistic presentation of the Rails architecture highlights some of the concerns that you should be factoring into your code as you become a more seasoned Rails developer.

Advanced Tricks and Patterns for MVC Development

Chapters 3 and 4 focus on getting the most from the MVC paradigm. Strict adherence to MVC is one of Ruby on Rails' most prominent contributions to web development, but the benefits you get from this code-organization structure can vary widely based on how you choose to organize the code within it. Chapter 3 discusses the MVC design process, including the steps for organizing your design work, a plan for decomposing functionality into the right objects, and guidance on refactoring your code.

Chapter 4 focuses on the implementation side of MVC with the goal of making your code as clear and concise as possible. It provides guidance on how to divide your implementation between the model and controller layers for maximum reusability and seamless error-handling, provides examples of aspect-oriented programming, and shows you how to decompose your HTML code so that you'll never have to repeat yourself, among other things.

Read-Write Web: APIs, Resources, and REST

Chapters 5 and 6 focus on the emerging web application architecture and what this means for APIs, resources, and REST (Representational State Transfer). Chapter 5 shows how to design web applications so that API access is overlaid on top of your web controllers from the very start, and it provides techniques for metering API access and managing multiple data formats. Chapter 6 introduces the idea of resources, one of the foundational metaphors for the future of web development, and presents the REST application architecture. REST both guides your design toward a simple and consistent style and centers your application's operations around a growing standard on the web that supports interoperability and sharing between web applications.

AJAX Patterns

The wealth of full-featured JavaScript frameworks today means that the hard part of AJAX is no longer AJAX itself, but all the design issues that begin to arise after you have decided to go that route with your UI design. Chapter 7 presents five different AJAX design patterns that characterize different approaches to AJAX integration. It elaborates in depth two of these patterns — partial style and puppet style — that are particularly effective in Rails applications, and it shows how to integrate these styles of AJAX into your application without losing the simplicity and reusability of your design.

Advanced Ruby and Meta-programming

Much of the style of Ruby on Rails would not be possible without the Ruby language. Chapters 8, 9, and 10 focus on some of the wonderful advanced features of Ruby that make it so different from other languages. You will learn how to think and design in ''blocks'' and discover several design patterns that blocks make possible, such as adverb-based programming, creative APIs, and code wrappers. Chapter 9 dives into mixin-based development and monkey patching. You will learn how to change the implementation of an object after it has been loaded in memory and will see how to use this technique to refine the way the Rails framework behaves. Chapter 10 teaches you how to use message passing and the `method_missing` method to create introspective and dynamic APIs such as `ActiveRecord`.

Group Schema Development and Behavior-Driven Development

Chapters 11 and 12 address topics outside the ''application'' component of web applications. They show you how schema development and code testing can become integral driving factors in your design and development process. Chapter 11 discusses topics in data management, focusing primarily on `ActiveRecord` migrations and how to manage your migrations over the life span of a project and working with a large team of members. It also dives into other database-related challenges, such as techniques for seeding production data and encoding complex object models within relational schemas. Chapter 12 presents behavior-driven development (BDD) and a framework called RSpec that implements it. BDD is a reconsideration of test-driven development that is taking the Rails community by storm. You'll have to turn to the chapter to find out why!

Whom This Book Is For

This book is for any developer who has a basic understanding of Ruby on Rails and is looking to expand his or her skills to become a seasoned Rails designer. Ideally, you have written a few toy applications and have a general familiarity with the key features that Rails is known for — routing, models, views, controllers, associations, validations, layouts, and partials. This book provides short refreshers when these core concepts come up, but quickly moves on to higher-level discussions about how to best use these concepts for effective development.

Although this is a Ruby on Rails-centric book, many of the topics contained within are relevant to any developer who wishes to understand the techniques and design patterns that thrive on modern MVC-style web frameworks. As such, it is a good resource for developers wanting to learn the ''Rails style'' even if their target platform is something else. As has been said on the web more than a few times, learning Ruby on Rails is a great way to become a better PHP developer.

What's Up With the Stories?

Each chapter begins with a story about a fictional character named W. Web who gets caught up in an adventure that spans the book and ends online at both the book's companion web site (`www.wrox.com`) and at www.artofrails.com. Each chapter's segment of the story roughly aligns with the overall topics and themes of the chapter — some blatantly and some a bit more subtly.

I wanted to add a storyline to the book because I believe that a book so heavily concerned with design should be something that you can read cover to cover rather than something that just serves as a reference. Writing each installment of W. Web's adventure was a way I could remind myself of that goal at the start of each chapter. My other, much simpler motive was that it was fun. The most entertaining technical book I have ever read is *Why's (Poignant) Guide to Ruby* (http://poignantguide.net). Although this book lacks *Why's* crazy cartoons and chunky bacon, the stories were lots of fun to write and, I hope, will be fun for you to read.

Conventions

To help you get the most from the text and keep track of what's happening, I've used a number of conventions throughout the book.

> Boxes like this one hold important, not-to-be forgotten information that is directly relevant to the surrounding text.

Tips, hints, tricks and asides to the current discussion are offset and placed in italics like this.

As for styles in the text:

- I *highlight* new terms and important words when I introduce them.
- I show filenames, URLs, and various code elements within the text like so: `persistence.properties`.
- I present code in two different ways:

```
I use a monofont type with no highlighting for most code examples.
I use gray highlighting to emphasize code that's particularly important in the present
context.
```

Source Code

Good code is concise, sometimes startlingly so, and Rails allows you to program at your best. This book consciously attempts to steer clear of large code examples in favor of small, targeted segments of code to demonstrate a particular point or technique. This keeps the signal-to-noise ratio high, and keeps the focus on what is meaningful about the technique and when you might want to use it.

Any code examples long enough to be run in a stand-alone fashion can be found online for your convenience in the source code files that accompany the book. This code is available for download at http://www.wrox.com. When you're at the site, simply locate the book's title (either by using the Search box or by using one of the title lists) and click the Download Code link on the book's detail page.

Because many books have similar titles, you may find it easiest to search by ISBN. This book's ISBN is 978-0-470-18948-1.

After you download the code, just decompress it with your favorite compression tool. Alternatively, you can go to the main Wrox code download page at `http://www.wrox.com/dynamic/books/download.aspx` to see the code available for this book and all other Wrox books.

Errata

I make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error, such as a spelling mistake or faulty piece of code, I would be very grateful for your feedback. By sending in errata, you may save another reader hours of frustration and at the same time you will be helping me provide even higher-quality information.

To find the errata page for this book, go to `http://www.wrox.com` and locate the title using the Search box or one of the title lists. Then, on the book details page, click the Book Errata link. On this page, you can view all errata that has been submitted for this book and posted by Wrox editors.

If you don't spot ''your'' error on the Book Errata page, go to `www.wrox.com/contact/techsupport.shtml` and complete the form there to send me the error you have found. I'll check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

p2p.wrox.com

For author and peer discussion, join the P2P forums at `p2p.wrox.com`. The forums are a Web-based system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At `http://p2p.wrox.com` you will find a number of different forums that will help you not only as you read this book but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to `p2p.wrox.com` and click the Register link.
2. Read the terms of use and click Agree.
3. Complete the required information to join as well as any optional information you wish to provide and click Submit.
4. You will receive an e-mail with information describing how to verify your account and complete the joining process.

You can read messages in the forums without joining P2P, but in order to post your own messages, you must join.

After you join, you can post new messages and respond to messages other users post. You can read messages at any time on the Web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

1 Emergence(y) of the New Web

W. Web knew immediately that something was wrong. He had suffered stomach pangs before, but never like this. Stumbling out of the taxi cab and toward the hospital, he mopped the sweat from his brow and pushed his way through the sidewalk traffic.

Inside, everything was a dizzy blur flowing past him — nurses, patients, a police officer, and several computer technicians hitting a computer monitor and mumbling something about the Internet going down.

''I know, I know!'' Web thought as he struggled past them for the emergency patient entrance.

Luckily for W. Web, this particular hospital uses a triage system, and when you explain to the nurse at the front desk that you are the Internet, you get bumped to the front of the line. A lot is riding on your health.

As Web lay in his hospital gurney, passing other familiar technologies as the nurse pushed him down the hall, he realized that he had made the right decision to stop ignoring the pangs. It was going to be okay.

This book will make you a better web application developer. And if some of the pundits with crystal balls are to be believed, we're all on the path to becoming web application developers. More specifically, this book will make you a better Ruby on Rails developer. It assumes that you have written code using Ruby on Rails and now you are thirsty to understand how to design with Ruby on Rails and how to master the elements of Ruby that make it so successful.

The web is a strange medium for application development. Web applications can't run by themselves, they have little access to a machine's hardware and disk capabilities, and they require a menagerie of client and server software providing them with life support. Despite all this, as you probably already know or are beginning to learn, the Web is a wonderful and exciting place to be developing applications.

Programming for the Web is a blend of art and engineering. Its odd quirks and demands can be harnessed to create applications out of clean, concise, elegant code with minimal waste. This book will show you how.

Programming for the Web is also a task in which good design skills can be the most critical part of a project because of the lack of features such as compilation and type checking found in the desktop world.

Web applications aren't programs; they are ecosystems. For each separate task, a separate language is called upon: SQL for data persistence; Ruby and others for application logic; HTML for UI structure; CSS for UI appearance; and JavaScript for UI logic. Good design skills must extend past the knowledge of each individual area and incorporate the ability to coordinate all areas. On top of all that, the rise of web APIs and RESTful endpoints enable yet another ecosystem of web applications to communicate with each other and exchange services, adding another layer of abstraction that is built upon the ones below.

The Web is here to stay, and its potential will only grow as a development platform. As Internet access approaches utility-like status, as telephone and television did before it, the distinction between your hard drive and ''the cloud'' will blur to the point that the two are functionally indistinguishable. With the exception of maybe games and media editors, applications on the Web will be every bit as powerful as those once deployed on CDs and DVDs, but these new applications will be easier to code, faster to deploy, and will harness the combined intelligence of swarms of users to enrich their experience.

These changes are already taking place. In 2007, the primary process for both the Democratic and Republican parties included a presidential debate with a new twist: Questions for the candidates were asked via YouTube videos submitted by ordinary people through the Web. Web front ends for our banks, stocks, and bills are now considered requirements instead of features. It is no longer surprising to store data that we own, such as our documents and photos, to web applications such as Google Docs and Flickr — space leased for free in exchange for a bit of advertising. The Web is no longer just about fetching documents; instead, it has become a universal communications medium for both humans and software.

If you are here reading this page, then you see these changes taking place. The question that remains is how to best understand and take advantage of this new development medium.

This book aims to be a blend of design and programming. It takes a critical look at what makes the modern Web tick from the standpoint of Ruby on Rails. The chapters touch on a wide range of topics, from REST-based web design to domain-specific languages and behavior-driven development. All these topics represent the cutting edge of thought about web development and will become cornerstones of the web of applications that will flourish over the coming years.

At times throughout the book, the code will be sparse; elsewhere, it will be frequent. In all chapters, long code examples will be avoided in favor of small code examples interwoven with the text to demonstrate an idea. This is a book primarily about concepts, not syntax.

Rails, Art, and the New Web

No development framework understands the new Web better than Ruby on Rails. In a world of general-purpose languages applied to the Web through libraries and Apache modules, Ruby on Rails was the application framework to speak the Web language as its native language. Rails is both a programming framework and a style of development reflected by that framework.

Ruby on Rails embraces the latest thoughts in web design so thoroughly that in many cases it literally forces you to use them. Most other frameworks don't have this option — they have been around so long that entire industries built around them require legacy support. As a newcomer to the scene, Rails is in the unique position of being able to cherry pick both its features and the way that it exposes them to the developer, unifying the framework around these choices. Remember when Apple ditched the floppy drive? It's like that.

This our-way-or-the-highway approach is a bit brazen, but it has a wonderful effect: It yields a remarkably clean framework that makes writing high-quality code in very little time easy. Most of the ''housekeeping'' involved in writing a web application is done for you by the framework, and the rest of your coding tasks are assisted by a host of helper functions and code generators (both code-time and run-time). This means that good Rails developers can spend their time focusing on design-related issues rather than writing code, making sure that each line written is effective and appropriate.

But Ruby on Rails is still a tool, and as with any other tool, it can be misused. Tools that make a point of being simple to use often lull their owners into a false sense of security. The quick learning curve creates the illusion that there isn't anything else to it. Rails, and the Ruby language, are known for being concise, but tidy code doesn't come for free.

Art and Engineering

This book will teach you the finer points of designing and coding in the Ruby on Rails environment — the points that will transform Ruby on Rails from an easy-to-use web tool into a methodology of programming in which every design choice you make is purposeful. Ruby on Rails is a particularly good platform on which to practice this type of purposeful, artful programming because of the way it cuts out the fat in web development to leave only your design remaining.

Software development takes on an inherently artistic quality when the developer truly understands the environment he or she is working in and the tools that are available. Conversely, if you have ever watched a watercolor painter work, you know that art has a lot of engineering in it. Watercolor paintings are intricately designed in advance because each brush stroke can only add to, rather than replace, the color beneath it. Intricate protective masks are applied and layered with the precision of an Intel engineer layering the metal on a silicon wafer.

Ruby on Rails operates with this level of attention to the environment of web development — a level that extends beyond engineering and into art. This book attempts to address the higher-level web application design issues with this same level of attention. With a solid framework underneath and good design skills guiding your programming, your development will become both productive and fun, and these qualities will be reflected in the software that you write.

The New Web

The days of version numbers seemed over when Microsoft Windows suddenly jumped from version 3.11 to 95 overnight, and then advanced 1,905 product releases forward to 2000 in the span of just five years. So what a throwback it seemed when the masses collectively announced that the Web was versioned, too, and it had reached 2.0.

Web 1.0 describes the web as a digital version of the traditional publish-subscribe media model in which a few groups produce content while the majority of users passively consume it. Web 2.0 is a correction of this imbalance. Web 2.0 applications provide environments in which users can create and publish their own content without having to create and maintain web sites by themselves. Applications such as Blogger, Flickr, Digg, Wikipedia, YouTube, and Facebook turn over the bullhorn to their users, and in doing so have toppled traditional assumptions about the media industry.

Foreshadowed by the prevalence of APIs on the Web today, Web 3.0, as many are calling it, will bring a layer of automated reasoning and programmability to the Web. Semantic search engines will be able to answer questions such as ''which flights will get me to Seattle before lunchtime tomorrow'' instead of simply suggesting web sites associated with the topics ''flights,'' ''Seattle,'' and ''lunch.'' These engines will be able to sift through the Web as a data set, piecing together bits from multiple web sites using semantic markup to align the meaning of the data elements of each. This roadmap for the Web is shown in Figure 1-1.

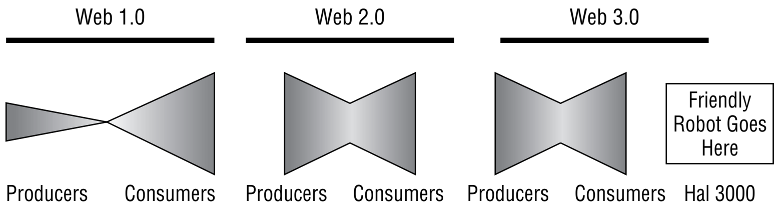

Figure 1-1

Another story is taking place beneath the media headlines and business models, and that is what this book is all about. A true renaissance of web development techniques is making the new capabilities of the Web possible. These advances are breaking long-held assumptions about the way web sites are developed and are introducing a completely new discipline of application development. In the world of web developers, each new ''version'' of the Web reflects a maturing of the art of web development as a discipline.

On the client side, Web 2.0 represented the push for refinement and tidying up of web formats, turning what once was a document markup language into a full-blown client-server application platform. This new platform was made possible because the web development community chose to place a high value on excellence in coding. XHTML and CSS validation icons were displayed as badges of honor at the bottoms of web sites, and the tutorials filling the Web focused on getting rid of the endless `TABLE` tags that clogged auto-generated HTML and on moving instead to simple, hand-crafted XHTML designs decorated entirely via CSS.

On the server side, the changes included new ideas about the ways frameworks should interact with developers, new interpretations of how URLs represent a web site's capabilities, and the incorporation of traditional application programming techniques into web development. In the chapters ahead, you will see how Ruby on Rails is at the forefront of many of these changes and how to incorporate them into your own development.

As the technologies of the Semantic Web are refined, Web 3.0 will be made possible by the introduction of resource-oriented development techniques in web development. The REST development style in Chapter 6 will teach you resource-oriented web design, which is the first step in this process. REST-based web design paves the way for formal modeling and reasoning systems to be directly integrated into our web applications, combining modern web design with the Semantic Web vision of an interlinking web of data and logic. So what is the ''New Web''? The New Web isn't one particular set of technologies or content