Extracted from:

Effective Testing with RSpec 3

Build Ruby Apps with Confidence

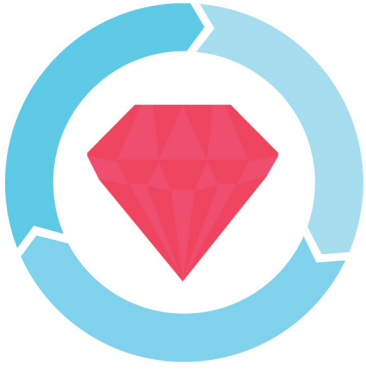
The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Effective Testing with RSpec 3

Build Ruby Apps with Confidence



Myron Marston
and Ian Dees
edited by Jacquelyn Carter

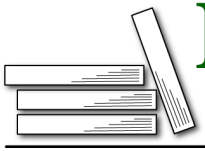
Foreword by Tom Stuart,
author of *Understanding Computation*

Effective Testing with RSpec 3

Build Ruby Apps with Confidence

Myron Marston
Ian Dees

The Pragmatic Bookshelf
Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Executive Editor: Susannah Davidson Pfalzer
Development Editor: Jacquelyn Carter
Indexing: Potomac Indexing, LLC
Copy Editor: Liz Welch
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-198-8
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—August 2017

Introduction

"Our tests are broken again!" "Why does the suite take so long to run?" "What value are we getting from these tests anyway?"

The years go by and the technologies change, but the complaints about automated tests are the same. Teams try to improve the code and end up fighting test failures. Slow test times drag down productivity. Poorly written tests do a bad job communicating, guiding the software design, or catching bugs.

No matter whether you're new to automated tests or have been using them for years, this book will help you write more effective tests. By *effective*, we mean tests that give you more value than the time spent writing them.

We'll be using the RSpec 3 framework to explore the art of writing tests. Every aspect of RSpec was designed to solve some problem that developers have encountered in the wild. With it, you can build Ruby apps with confidence.

How to Use This Book

With this book, you'll learn RSpec 3 in three phases:

- Part I: Introductory exercises to get you acquainted with RSpec
- Part II: A worked example spanning several chapters, so that you can see RSpec in action on a meaningfully sized project
- Parts III–V: A series of deep dives into specific aspects of RSpec, which will help you get the most out of RSpec

We wrote this book to be read cover to cover. Whatever your level of expertise, reading the chapters in order will give you the most value. However, if you're pressed for time and want to know where to look first, we can make a few suggestions.

If you're familiar with other test frameworks but new to RSpec, we recommend that you read the first two parts of the book, and then try RSpec out in one

of your own projects. As you do so, you'll likely have questions that you can consult specific deep-dive chapters for.

If you're a long-time user of RSpec, you can start with Parts III, IV, and V. These contain detailed recipes for situations you've likely encountered in the wild. Later on, you can return to the beginning of the book for a refresher on RSpec's philosophy.

Finally, if you use RSpec 3 every day, keep the deep-dive parts of this book nearby. You'll find them handy to refer to in specific situations—we do, and we've been using RSpec for years!

Code Snippets

We have provided code snippets throughout the book that show how RSpec is used in real-world situations. Most of these examples are intended for you to follow along with on your computer, particularly those in Part I and Part II.

A typical snippet will contain one or more lines of Ruby code meant for you to type into your text editor so that you can run them later. Here is an example:

00-introduction/01/type_me_in.rb

```
puts "You can type me in; it's okay!"
```

We'll show each code file a few lines at a time. If you need more context for any given snippet, you can click the filename banner (in the eBook) or open the book's source code (linked at the end of this chapter) to view the entire file at once.

Some code examples have no banner; these typically represent a session at your terminal, either in interactive Ruby (IRB) or in a shell like Bash. For IRB snippets, you'll run the `irb` terminal command and then type in just the parts after the green >> prompt:

```
>> %w[Type in just the bit after the prompt].join(' ')
=> "Type in just the bit after the prompt"
```

We'll represent shell sessions with a green `$` prompt instead. As with IRB sessions, you won't type in the prompt or the output lines, just the commands *after* the prompt:

```
$ echo 'RSpec is great!'
RSpec is great!
```

Later on in the book, we sometimes show isolated snippets from a larger project; these are *not* meant for you to run on your computer. If you're interested in running them on your own, you can download all the project files from the book's source code repository.

Most chapters have a "Your Turn" section with exercises for you to try. Don't skip these! Practicing on your own will ensure that each chapter builds on the skills you've honed over the course of the book.

RSpec and Behavior-Driven Development

RSpec bills itself as a behavior-driven development (BDD) test framework. We'd like to take a moment to talk about our use of that term, along with a related term, test-driven development (TDD).

Without TDD, you might check your program's behavior by running it manually or by writing a one-off test harness. In situations where you intend to scrap the program shortly afterward, these approaches are all right. But when long-term maintenance is a priority, TDD provides important benefits.

With TDD, you write each test case just before implementing the next bit of behavior. When you have well-written tests, you wind up with more maintainable code. You can make changes with the confidence that your test suite will let you know if you've broken something.

The term TDD is a bit of a misnomer, though. Despite the fact that it has the word "test" in the name, TDD isn't just about your tests. It's about the way they enable fearless improvements to your design. For this reason, Dan North coined the term behavior-driven development in 2006 to encapsulate the most important parts of TDD.[1]

BDD brings the emphasis to where it's supposed to be: your code's *behavior*. The community stresses the importance of expressiveness in your tests, something that we'll be talking about a lot in this book. BDD is also about treating your software requirements with the same kind of care, since they're yet another expression of behavior. It's about involving all of your stakeholders in writing acceptance tests.

1. https://dannorth.net/introducing-bdd/

As a test framework, RSpec fits into a BDD workflow quite well. RSpec helps you "get the words right" and specify exactly what you mean in your tests. You can easily practice the outside-in approach favored in BDD, where you start with acceptance tests and move inward to unit tests.[2] At every level, your expressive tests will guide your software design.

However, RSpec and BDD are not synonymous. You don't have to practice BDD to use RSpec, nor use RSpec to practice BDD. And much of BDD is outside the scope of RSpec; we won't be talking in this book about stakeholder involvement, for instance.

Who We Are

Myron Marston started using RSpec in 2009 and began contributing to it in 2010. He's been its principal maintainer since late 2012. Here are just of the few major improvements he's made to RSpec:

- Composable matchers, which express exactly the pass/fail criteria you need
- `rspec --bisect`, which finds the minimal set of test cases to reproduce a failure
- Integrating RSpec's assertions and mocking libraries with the Minitest framework that ships with Ruby
- The `--only-failures` and `--next-failure` options that let you rerun just your failing tests so that you can fix bugs more quickly

With the insider knowledge Myron provides in this book, you'll learn all of these techniques and more. By the end, you'll be able to get free of just about any problems you run into with your test suite.

Ian Dees stumbled on an old beta of RSpec in 2006. It was just what he needed to build the automated acceptance tests for an embedded touchscreen device. Since then, he's used and taught RSpec for testing everything from tiny microcontrollers to full-featured desktop and web apps.

Who You Are

We hope this book is useful to a wide range of developers, from people who are just getting started with RSpec to those who have written thousands of tests with it. That said, we have made a few assumptions in order to keep the book from getting too bogged down with introductory material.

2. https://dannorth.net/whats-in-a-story/

First, we assume you're familiar with Ruby. You don't need to be an expert. We stick to the basics of classes, methods, and blocks for the most part. We will be directing you to install several Ruby gems, so it'll be useful to be familiar with that process as well. If you're new to Ruby, we recommend you first learn the language a bit using resources like Zed Shaw's *Learn Ruby the Hard Way* eBook or the Ruby tutorials at exercism.io.[3,4]

Although you'll be building a web service over the course of several chapters, we don't assume that you're already a web developer. Lots of folks use RSpec to test command-line apps, GUI apps, and so on. We'll explain a few web development concepts as they come up during the discussion.

When we have content that's meant for a specific audience—such as people coming from an older version of RSpec or folks who are new to web development—we'll put that content in a sidebar.

A Note on Versions

The libraries we're using in this book, both the ones from the RSpec framework and other dependencies like Sinatra and Sequel, are designed to be backward-compatible across minor version upgrades. The code examples you see here should work just fine in future versions of these libraries—at least until their next *major* versions.

While we've tested this code on multiple Ruby versions as far back as Ruby 2.2, you'll have the best experience if you follow along with the exact same versions we call out in the text: Ruby 2.4, RSpec 3.6, and so on. With the same versions we use, you should get output that closely mirrors what we show in the book.

Online Resources

This book has a website.[5] There, you'll find links to source code, discussion forums, and errata. We've also set up GitHub repositories containing all the examples in the book, plus a version of the project you'll build in *Building an App With RSpec 3*.[6]

3. https://learnrubythehardway.org
4. http://exercism.io/languages/ruby/about
5. https://pragprog.com/book/rspec3/effective-testing-with-rspec-3
6. https://github.com/rspec-3-book

For more information about RSpec, you can turn to the official site and the full developer documentation.[7,8]

Myron Marston
Lead Maintainer of RSpec
myron.marston@gmail.com
Seattle, WA, August 2017

Ian Dees
Senior Software Engineer, New Relic
undees@gmail.com
Portland, OR, August 2017

7. http://rspec.info
8. http://rspec.info/documentation/