Extracted from:

Continuous Testing

with Ruby, Rails, and JavaScript

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Continuous Testing

Ben Rady and Rod Coffin

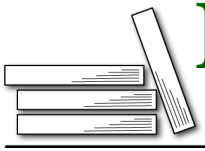
Edited by Jacquelyn Carter

Continuous Testing

with Ruby, Rails, and JavaScript

Ben Rady
Rod Coffin

The Pragmatic Bookshelf
Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Jacquelyn Carter (editor)
Potomac Indexing, LLC (indexer)
Kim Wimpsett (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-934356-70-8
Printed on acid-free paper.
Book version: P1.0—June 2011

As professional programmers, few things instill more despair in us than discovering a horrible production bug in something that worked perfectly fine last week. The only thing worse is when our customers discover it and inform us…angrily. Automated testing, and particularly test driven development, were the first steps that we took to try to eliminate this problem. Over the last ten years, these practices have served us well and helped us in our fight against defects. They've also opened the doors to a number of other techniques, some of which may be even more valuable. Practices such as evolutionary design and refactoring have helped us deliver more valuable software faster and with higher quality.

Despite the improvements, automated testing was not (and is not) a silver bullet. In many ways, it didn't eliminate the problems we were trying to exterminate but simply moved them somewhere else. We found most of our errors occurring while running regression or acceptance tests in QA environments or during lengthy continuous integration builds. While these failures were better than finding production bugs, they were still frustrating because they meant we had wasted our time creating something that demonstrably did not work correctly.

We quickly realized that the problem in both cases was the timeliness of the feedback we were getting. Bugs that happen in production can occur weeks (or months or years!) after the bug is introduced, when the reason for the change is just a faint memory. The programmer who caused it may no longer even be on the project. Failing tests that ran on a build server or in a QA environment told us about our mistakes long after we'd lost the problem context and the ability to quickly fix them. Even the time between writing a test and running it as part of a local build was enough for us to lose context and make fixing bugs harder. Only by shrinking the gap between the creation of a bug and its resolution could we preserve this context and turn fixing a bug into something that is quick and easy.

While looking for ways to shrink those feedback gaps, we discovered continuous testing and started applying it in our work. The results were compelling. Continuous testing has helped us to eliminate defects sooner and given us the confidence to deliver software at a

faster rate. We wrote this book to share these results with everyone else who has felt that pain. If you've ever felt fear in your heart while releasing new software into production, disappointment while reading the email that informs you of yet another failing acceptance test, or the joy that comes from writing software and having it work perfectly *the first time*, this book is for you.

Using continuous testing, we can immediately detect problems in code—before it's too late and before problems spread. It isn't magic but a clever combination of tests, tools, and techniques that tells us right away when there's a problem, not minutes, hours, or days from now but right now, when it's easiest to fix. This means we spend more of our time writing valuable software and less time slogging through code line by line and second-guessing our decisions.

Exploring the Chapters

This book is divided into two parts. The first part covers working in a pure Ruby environment, while the second discusses the application of continuous testing in a Rails environment. A good portion of the second part is devoted to continuous testing with JavaScript, a topic we believe deserves particular attention.

In Chapter 1, *Why Test Continuously?*, on page ?, we give you a bit of context. This chapter is particularly beneficial for those who don't have much experience writing automated tests. It also establishes some terminology we'll use throughout the book.

The next three chapters,
Chapter 2, *Creating Your Environment*, on page ?, Chapter 3, *Extending Your Environment*, on page ?, and Chapter 4, *Interacting with Your Code*, on page ?, show how to create, enhance, and use a continuous testing environment for a typical Ruby project. We'll discuss the qualities of an effective suite of tests and show how continuous testing helps increase the quality of our tests. We'll take a close look at a continuous test runner, Autotest, and see how it can be extended to provide additional behavior that is specific to our project and its needs. Finally, we'll discuss some of the more advanced techniques that continuous testing allows, including inline assertions and comparison of parallel execution paths.

In the second part of the book, we create a CT environment for a Rails app. In addition to addressing some of the unique problems that Rails brings into the picture, we also take a look at another continuous test runner, Watchr. As we'll see, Watchr isn't so much a CT runner but a tool for easily creating feedback loops in our project. We'll use Watchr to create a CT environment for JavaScript, which will allow us to write tests for our Rails views that run very quickly and without a browser.

At the very end, we've also included a little "bonus" chapter: an appendix on using JavaScript like a functional programming language. If your use of JavaScript has been limited to simple HTML manipulations and you've never had the opportunity to use it for more substantial programming, you might find this chapter very enlightening.

For the most part, we suggest that you read this book sequentially. If you're very familiar with automated testing and TDD, you can probably skim through the first chapter, but most of the ideas in this book build on each other. In particular, even if you're familiar with Autotest, pay attention to the sections in Chapter 2, *Creating Your Environment*, on page ? that discuss FIRE and the qualities of good test suites. These ideas will be essential as you read the later chapters.

Each chapter ends with a section entitled "Closing the Loop." In this section we offer a brief summary of the chapter and suggest some additional tasks or exercises you could undertake to increase your understanding of the topics presented in the chapter.

Terminology

We use the terms *test* and *spec* interchangeably throughout the book. In both cases, we're referring to a file that contains individual examples and assertions, regardless of the framework we happen to be working in.

We frequently use the term *production code* to refer to the code that is being tested by our specs. This is the code that will be running in production after we deploy. Some people call this the "code under test."

Who This Book Is For

Hopefully, testing your code continuously sounds like an attractive idea at this point. But you might be wondering if this book is really applicable to you and the kind of projects you work on. The good news is that the ideas we'll present are applicable across a wide range of languages, platforms, and projects. However, we do have a few expectations of you, dear reader. We're assuming the following things:

- You are comfortable reading and writing code.
- You have at least a cursory understanding of the benefits of automated testing.
- You can build tools for your own use.

Knowledge of Ruby, while very beneficial, isn't strictly required. If you're at all familiar with any object-oriented language, the Ruby examples will likely be readable enough that you will understand most of them. So if all of that sounds like you, we think you'll get quite a bit out of reading this book. We're hoping to challenge you, make you think, and question your habits.

Working the Examples

It's not strictly necessary to work through the examples in this book. Much of what we do with the examples is meant to spark ideas about what you should be doing in your own work rather than to provide written examples for you to copy. Nonetheless, working through some of the examples may increase your understanding, and if something we've done in the book would apply to a project that you're working on, certainly copying it verbatim may be the way to go.

To run the examples in this book, we suggest you use the following:

- A *nix operating system (Linux or MacOS, for example)
- Ruby 1.9.2
- Rails 3.0.4

In addition, you can find a list of the gems we used while running the examples in Appendix 2, *Gem Listing*, on page ?.

The examples may work in other environments (such as Windows) and with other versions of these tools, but this is the configuration that we used while writing the book.

Online Resources

The source for the examples is available at http://pragprog.com/titles/rcctr/source_code.

If you're having trouble installing Ruby, we suggest you try using the Ruby Version Manager (or RVM), available at: http://rvm.beginrescueend.com/.

If something isn't working or you have a question about the book, please let us know in the forums at http://forums.pragprog.com/forums/170.

Ben blogs at http://benrady.com, and you can find his Twitter stream at https://twitter.com/benrady.

Ben Rady
June 2011

Rod Coffin
June 2011