Extracted from:

The RSpec Book

Behaviour-Driven Development with RSpec, Cucumber, and Friends

The Pragmatic Programmers

The RSpec Book

Behaviour-Driven Development with RSpec, Cucumber, and Friends

David Chelimsky
with Dave Astels, Zach Dennis, Aslak Hellesøy, Bryan Helmkamp, and Dan North

Foreword by Robert C. Martin (Uncle Bob)

Edited by Jacquelyn Carter

The Facets of Ruby Series

Chapter 20

Cucumber with Rails

Cucumber supports collaboration between project stakeholders and application developers, with the goal of developing a common understanding of requirements and providing a backdrop for discussion. The result of that collaboration is a set of plain-text descriptions of features and automated scenarios that application code must pass to be considered *done*. Once passing, the scenarios serve as regression tests as development continues.

As with any BDD project, we use Cucumber in a Rails project to describe application-level behavior. In this chapter, we'll look at how Cucumber integrates with Rails, exploring a variety of approaches to setting up context, triggering events, and specifying expected outcomes as we describe the features of our web application.

20.1 Step Definition Styles

Step definitions connect the natural-language steps in a plain-text feature file to Ruby code that interacts directly with the application. Since Cucumber helps us describe behavior in business terms, the steps shouldn't express technical details. `Given I'm logged in as an administrator` could apply to a CLI, client-side GUI, or web-based application. It's within the step *definitions* that the rubber meets the road and code is created to interact with the application.

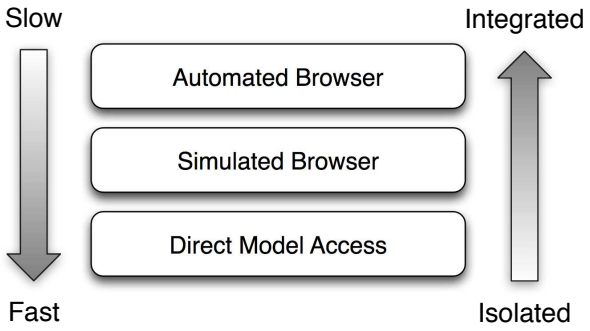

Figure 20.1: Comparing step definition styles

When building step definitions for a Rails application, we typically deal with three step definition styles for interacting with a web-based system in order to specify its behavior:

- *Automated Browser*: Access the entire Rails MVC stack in a real web browser by driving interactions with the Webrat API and its support for piggybacking on Selenium. This style is fully integrated but is the slowest to run and can be challenging to maintain.
- *Simulated Browser*: Access the entire MVC stack using Webrat, a DSL for interacting with web applications. This style provides a reliable level of integration while remaining fast enough for general use, but it doesn't exercise JavaScript.
- *Direct Model Access*: Access ActiveRecord models directly, bypassing routing, controllers, and views. This is the fastest but least integrated style.

When writing Cucumber scenarios, integration and speed are opposing forces, as illustrated in Figure 20.1. Fast is better than slow, of course, but integrated is better than isolated when we're looking for confidence that an app will work in the hands of users once it is shipped. So, what's the best approach to take?

Recommendations

We recommend using Simulated Browser with Webrat for Whens and Thens. This helps drive out the pieces that a user will interact with, providing confidence that the component parts are working well together

but still produces a suite that can be executed relatively quickly and without depending on a real web browser.

We generally recommend using direct model access in Givens, but there are a few exceptions. For anything that needs to set up browser session state, such as logging in, you should use Simulated Browser.

If there is any JavaScript or Ajax, add scenarios that use the Automated Browser approach in their Whens and Thens for the *happy path* and critical less common paths. The added value we get from doing this is exercising client-side code, so when no client code is necessary, there is no reason to use the browser.

Edge Cases

For features that produce many edge cases, it can be useful to drive a few through the Rails stack and the rest using just Direct Model Access for everything. This may seem more like a unit test, but keep in mind that scenarios are about communication. We want to make sure that we're writing the right code. If the customer asks for specific error messages depending on a variety of error conditions, then it's OK to go right to the model if that's the source of the message, as long as the relevant slice of the full stack is getting sufficient coverage from other scenarios.

In this chapter, we'll start with the simplest style, Direct Model Access, and walk through implementing a feature. Then we'll explore using Webrat for the Simulated Browser style in Chapter 21, *Simulating the Browser with Webrat*, on page 300 and Automated Browser in Chapter 22, *Automating the Browser with Webrat and Selenium*, on page 322.

20.2 Direct Model Access

Direct Model Access (DMA) step definitions execute quickly, but that speed and isolation comes at a price. They don't provide much assurance that the application works, and they are unlikely to catch bugs beyond those that should be caught by granular RSpec code examples that we'll be writing in a few chapters.

They do, however, facilitate conversation between the customer and developers and will catch regressions if the logic inside the models is broken in the future. In this way, DMA step definitions are useful for exercising fine-grained behaviors of a system, when driving all of them through the full stack would be too cumbersome.

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style, and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

The RSpec Book's Home Page
http://pragprog.com/titles/achbd
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragprog.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragprog.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
http://pragprog.com/news
Check out the latest pragmatic developments in the news.

Buy the Book

If you liked this PDF, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/achbd.

Contact Us

Phone Orders:	1-800-699-PROG (+1 919 847 3884)
Online Orders:	www.pragprog.com/catalog
Customer Service:	orders@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com