

Extracted from:

The RSpec Book

Behaviour Driven Development with RSpec, Cucumber, and Friends

This PDF file contains pages extracted from The RSpec Book, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2009 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The
Pragmatic
Programmers

The RSpec Book

Behaviour Driven Development
with RSpec, Cucumber,
and Friends

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

Edited by Jacquelyn Carter

The Facets  of Ruby Series



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

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://www.pragprog.com>

Copyright © 2010 The Pragmatic Programmers LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.

ISBN-10: 1-934356-37-9

ISBN-13: 978-1-934356-37-1

Printed on acid-free paper.

B12.0 printing, December 3, 2009

Version: 2010-1-3

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 behaviour. 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 behaviour 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.

The first step of the Outside-In cycle is to produce a failing scenario, and to do that we'll need a step definition, but how should it be implemented? Rails applications contain many layers, from the model and the database all the way up to the web browser, and this leaves us with options and choices in how step definitions interact with an application.

We want scenarios to exercise a vertical slice through all of our code, but we also want them to run fast. The fastest scenarios are going to bypass HTTP, routing, and controllers and just talk directly to the models. The slowest ones are going to exercise everything through the web browser, giving us the fullest coverage, the most confidence, and, unfortunately, the least desire to run them on a regular basis!

So what's a pragmatic story teller to do? Naturally, *it depends*. 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 behaviour:

- *Direct Model Access*: Access ActiveRecord models directly, bypassing routing, controllers, and views. This is the fastest but least integrated style.
- *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 doesn't exercise JavaScript.
- *Automated Browser*: Access the entire Rails MVC stack in a real web browser by driving interactions with the Webrat API and its support for piggy-backing on Selenium. This style is fully integrated but is the slowest to run and can be challenging to maintain.

General Recommendations

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. When writing Cucumber scenarios, integration and speed are opposing forces. This conundrum is illustrated in Figure 20.1, on the following page. The balance of these forces that feels best will vary a bit from developer to developer, but we can provide some guidelines based on experience exploring a variety of approaches.

We recommend using Direct Model Access in Givens to prepare the state of the system, except for logging-in or other actions that set up browser session state. Use Simulated Browser with Webrat for Whens and Thens. This helps to 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.

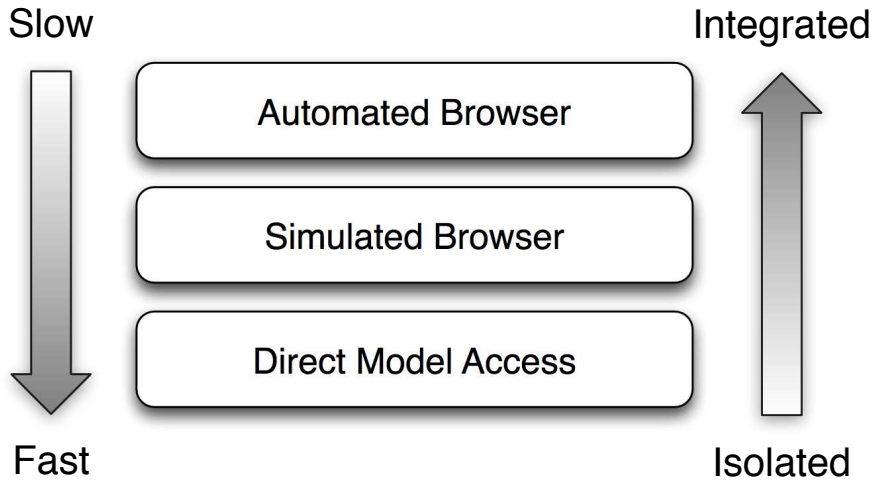


Figure 20.1: Comparing step definition styles

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*

Cucumber::Rails::World

Cucumber::Rails::World is the bridge between Cucumber and Rails. It provides the Rails integration testing methods within each scenario.

When Cucumber's Rails support is loaded by requiring `cucumber/rails/world` in `features/support/env.rb`, instances of `Cucumber::Rails::World` are configured to be the `World` for each scenario:

```
World do
  Cucumber::Rails::World.new
end
```

`Cucumber::Rails::World` inherits Rails' `ActionController::IntegrationTest`, and makes surprisingly few modifications to the superclass behaviour. Each scenario is run in a newly instantiated `Cucumber::Rails::World`. This gives us access to all of the functionality of Rails' Integration tests and RSpec's Rails-specific matchers, including simulating requests to the application and specifying behaviour with RSpec expectations.

In the default configuration, it runs each scenario in an isolated DB transaction. You can disable this by removing the following line from the `features/support/env.rb` generated by Cucumber:

```
Cucumber::Rails.use_transactional_fixtures
```

If you disable per-scenario transactions, you'll have to worry about records left over from one scenario affecting the results of the next. This often leads to inadvertent and subtle ordering dependencies in your scenario build. For these reasons, we *strongly* recommend using the transactional fixtures setting.*

*. Don't use `use_transactional_fixtures` with Selenium. We'll discuss this in detail in Chapter 22, *Automating the Browser with Webrat and Selenium*, on page 378

Browser with Webrat, on page 354 and Automated Browser in Chapter 22, *Automating the Browser with Webrat and Selenium*, on page 378.

20.2 Direct Model Access

Direct Model Access step definitions execute quickly and are immune to changes in the controller and view layers, but that speed and isolation comes at a price. DMA step definitions don't provide much assurance

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