8

Testing with IronRuby

* Learn about RSpec
* Learn about Cucumber
* Learn about CLR interop rules
* Learn about Caricature

At the start of this book we mentioned testing as a very important part of the development process, but the book has been rather thin on testing. That is about to change we’re going to have an exploration of testing both Ruby objects and CLR objects in this chapter. When I got first introduced to the concept of unit testing a few years ago I had to overcome a great deal of resistance. As soon as I got the hang of it and developed an application with a test-first approach I was sold on the concept.

As I’ve mentioned before testing gives me a warm fuzzy feeling and allows me to refactor in confidence. It’s also a good communication tool for collaborating with other members in your team as they immediately have a documented form of how the code is supposed to work. There is an entire book dedicated to testing with RSpec and Cucumber and the subject of BDD on itself is large enough to write one or more books on. We will just look at using these tools in the context of pure ruby and ruby to CLR interop.

At first sight testing stuff with ruby should be pretty easy because of its dynamic nature you have a great deal of control on how your objects look when you start testing them. However when you start dealing with objects defined with a CLR language the CLR interop rules kick in which may be confusing. At the very least you can get some surprising situations if you don’t follow those rules.

Anyway in this chapter we’ll look at using RSpec with IronRuby to test all kinds of objects. We’ll also look at Caricature a mocking library that bridges the gap between CLR and ruby objects. We’ll also briefly touch on using cucumber as a User Acceptance Test (UAT) tool but first we’re going to have a very brief history on how we got to Behavior Driven Development (BDD) through Test Driven Development (TDD).

8.1 Clearing up some of the naming confusion.

As you may be aware TDD has its roots in the eXtreme Programming (XP) methodology which is very related to Agile development. Agile development has a bunch of different disciplines and a bunch methodologies related to it like SCRUM, Lean etc. All of these methodologies have automated tests, unit tests etc as a core value. That is if you don’t do automated testing as a part of your agile development process, you’re not doing agile but some form of buzzword appeasement. Anyway lets take a look at TDD and what lead to BDD.

8.1.1 Test Driven Development

First of all the name of Test Driven Development is pretty misleading as it implies that you are going to actually test your application. While that is certainly possible it’s very time consuming and flow breaking to have to think if of all kinds of corner cases and in many companies there are dedicated testers for that.

So if it isn’t a tool for testing what is it then? It’s a design tool that comes with its own mantra: red/green/refactor. If you’re following the test driven part then you’re writing down the result of what your code is supposed to do before actually writing the code. If you run your test then it turns up as a failure, which in many test runners is colored red. Next you’re going to write the simplest bit of code that could make you satisfy the test. If you were to run your test then it would result in a success, which is often colored green. Now you have the chance to refactor out duplication, which takes you into the refactor phase. After refactoring you can run the test again and it should still pass. At this point you’re ready to start the mantra over with the next little bit of functionality.

This looks really good on paper and has been used for a while. There is one major flaw, not exactly a flaw more a suboptimal consequence. This type of testing puts a lot of weight on knowing the internals of an object and is often tied to the internals of an object resulting in a brittle test suite. Brittle test suites are hard to maintain and often lead to dropping the test part of the development process. The TDD community recognized this flaw and TDD evolved into BDD.

8.1.2 Behavior Driven Development

The key distinction between TDD and BDD is the way they handle the internals. In BDD it’s all about describing the behavior of an object not the way it has been implemented. When I first read about BDD it completely made sense to me and I didn’t feel the pressure anymore of having to test every little thing in my application. Instead I started looking at my objects from the point of view of the consumer of that class and only started testing that, after all that’s what I care about, and the internals are very likely to change in the future.

The TDD and BDD community, which consists of largely the same people, chose Ruby or SmallTalk to experiment with new ways of testing and new ways of approaching the testing story. The result of them choosing Ruby as a language to experiment is that the Ruby community is at the forefront of developments in this area and has a couple of really great tools at its disposal for doing BDD right. The tool we’ll look at shortly is RSpec, but before we do that a short word about Acceptance Test Driven Planning (ATDP).

8.1.3 Acceptance Test Driven Planning

In the TDD world people speak of Acceptance Test Driven Development and the primary tool used for this is FIT. BDD redubs the concept and utilizes it as a planning tool. The idea is that you provide scenarios for your application for the different roles of users of your system. These scenarios can be used to verify the overall state of the application.

In Ruby with Cucumber you can actually use these scenarios, called features, whilst putting together the backlog stories for your iteration you can use those stories to elaborate on later and write the tests that verify your application rather than the objects of your application. Next we’ll use cucumber to describe Hangman.

8.2 Cucumber

8.2 Introducing RSpec

8.3 CLR rules for IronRuby

8.4 Introducing Caricature

8.5 Testing Ruby objects

8.6 Testing CLR objects

8.7 Summary