Behaviour Driven Development

Marc Jeanson

Redline Software Inc.

www.redlinesoftware.com

marc@redlinesoftware.com

Twitter @marcjeanson

Why Do Software Projects Fail?

Deliver late or over budget (often both!)

Deliver the wrong thing

Unstable in production

Costly to maintain


I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO I DO IT IN PRODUCTION
memegenerator.net

Fast Feedback Loops

TDD

&

BDD

Terminology

Terminology Overload

Unit Tests

Integration Tests

Acceptance Tests

Regression Tests

White-box Tests

Black-box Tests

Functional Tests

Non-functional Tests

ATDD

ADHD

...etc

...etc...

What is a Unit Test?

It's a test on a **single unit**, such as a class or method, **in isolation**.

Acceptance Tests

Ideally created by developers and stakeholders together

Tell us what the system needs to do in order for the stakeholders to find it acceptable

Much more useful than receiving a requirements document

Comparison

Acceptance Tests ensure you ***build the right thing***

Unit Tests ensure you ***build the thing right***

Refactoring without tests isn't refactoring...

...it's just changing stuff.

Laws of TDD

You are not allowed to write any production code unless it is to make a failing unit test pass.

You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.

You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd

Why TDD?

Avoid wasting time debugging

Improve code quality

Improve design

Regression tests as byproduct

Increased confidence

Behaviour Driven Development

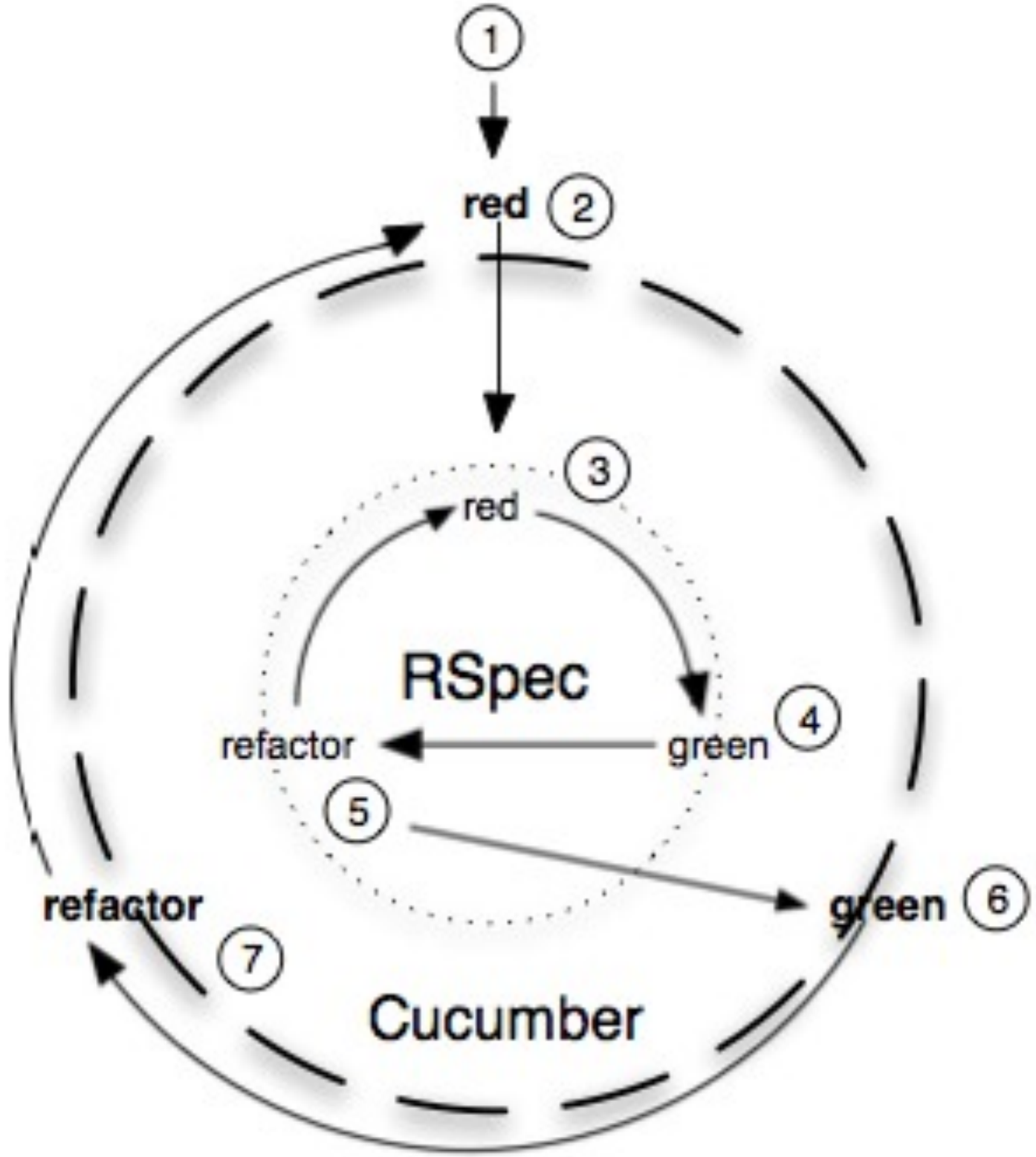
Think of the behaviour of the system instead of thinking of the tests

Driven by business value

Promotes “Outside-In Development”

YAGNI

BDD = TDD + automated acceptance testing



BDD Cycle from The RSpec Book

Gherkin

Gherkin

Lets you describe how the application should behave using a “business readable” DSL

Features (user stories) and scenarios are specified using Gherkin

Also doubles as documentation that stays “in-sync” with your codebase

Tools like Cucumber, SpecFlow, jBehave, etc use Gherkin to generate the test steps

Gherkin Syntax

Feature: [User Story]

Scenario: Title

Given [Context]

When [Action]

Then [Expected Outcome]

Gherkin Syntax

Scenario: Title

Given [Context]

And [More Context]

When [Action]

And [More context]

Then [Expected Outcome]

But [Unexpected Outcome]

Cucumber

Demo

Not just for Rails!

Java with JRuby

.NET using IronRuby

Erlang, Python, PHP, Flex...

https://github.com/cucumber/cucumber/wiki/

Alternatives

.NET: specflow.org

Java: jBehave.org

Javascript: Jasmine

Various: Fitnesse.org

Legacy Apps

How to get started?

Start by adding “black box” tests

➡ Using Cucumber or another acceptance testing framework

➡ Add them gradually, as you need them

Don’t add “white box” tests

Specification

vs

Characterization

Specification Tests

These are the tests we've been talking about already

Ideally they are written before the production code

They specify the code we wish we had

Characterization Tests

With legacy code, we want to add tests that describe the current behaviour of the application

These tests will be used to ensure that the correct behaviour is preserved in future changes

Recipe

1. Write a scenario that exercises some behaviour of your system
2. Add an assertion that you know will fail
3. Let the failure tell you what the behaviour is
4. Change the test so that it expects the behaviour that the code is producing
5. Repeat

Bug Fixing

Adding characterization tests to cover the entire app can take a long time

Starting with bug fixes can be a good approach

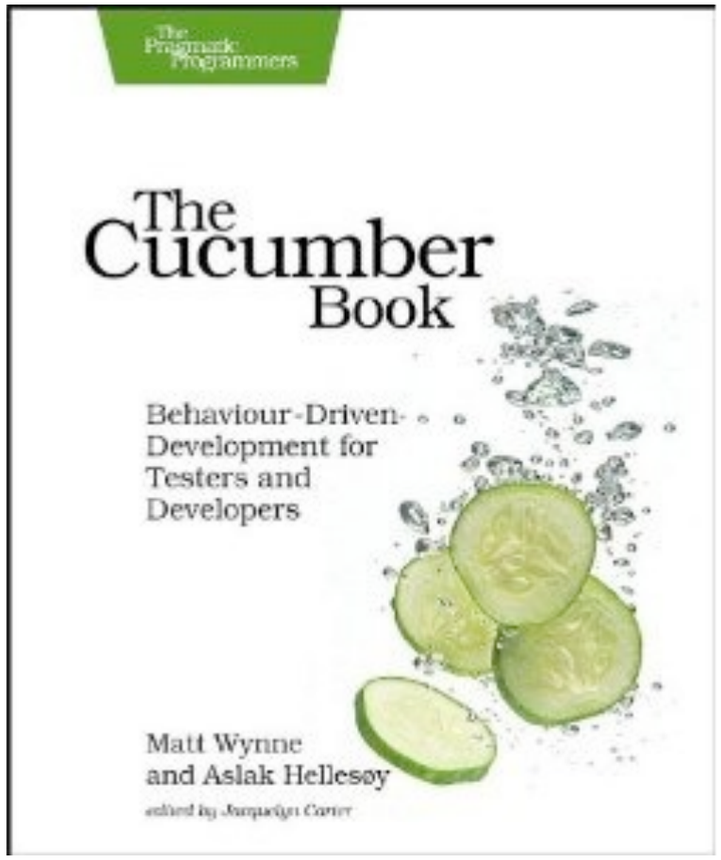
Bug Fix Recipe

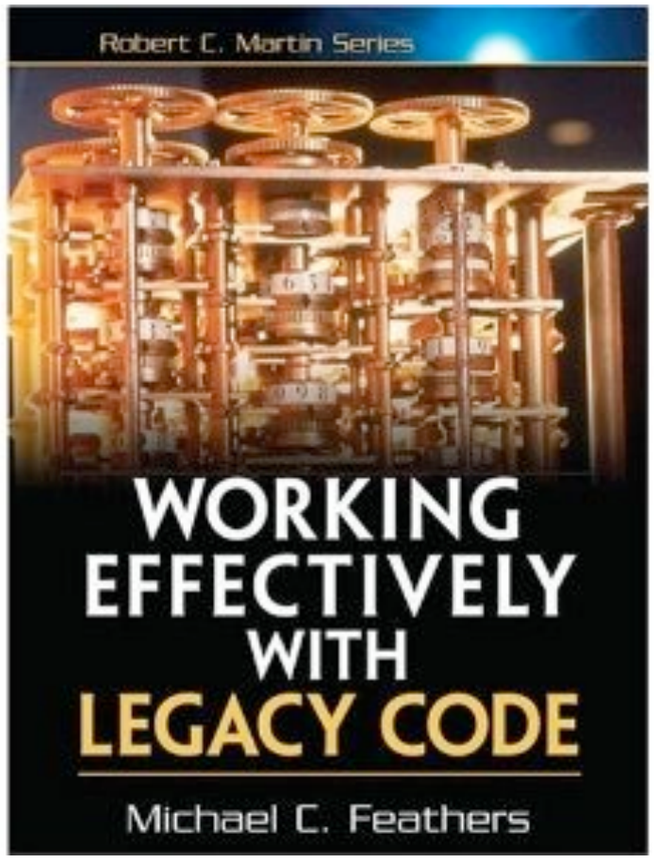
1. Translate the bug report into a scenario.
2. Run the scenario. It should fail the same way as your system currently is.
3. Investigate the busted code. Add additional characterization tests if needed.
4. Fix the code so the bug scenario passes.
5. Run all the tests to make sure you didn't add any regressions.

New Behaviour Recipe

1. Add any necessary characterization tests in the area that you will be changing
2. Add new scenarios to specify the new behaviour
3. Run the tests, something should fail. Examine the code to make it pass.
4. Add extra characterization tests to build up confidence to change the code if necessary
5. Add the code to make it pass
6. Repeat 3-6 until everything passes

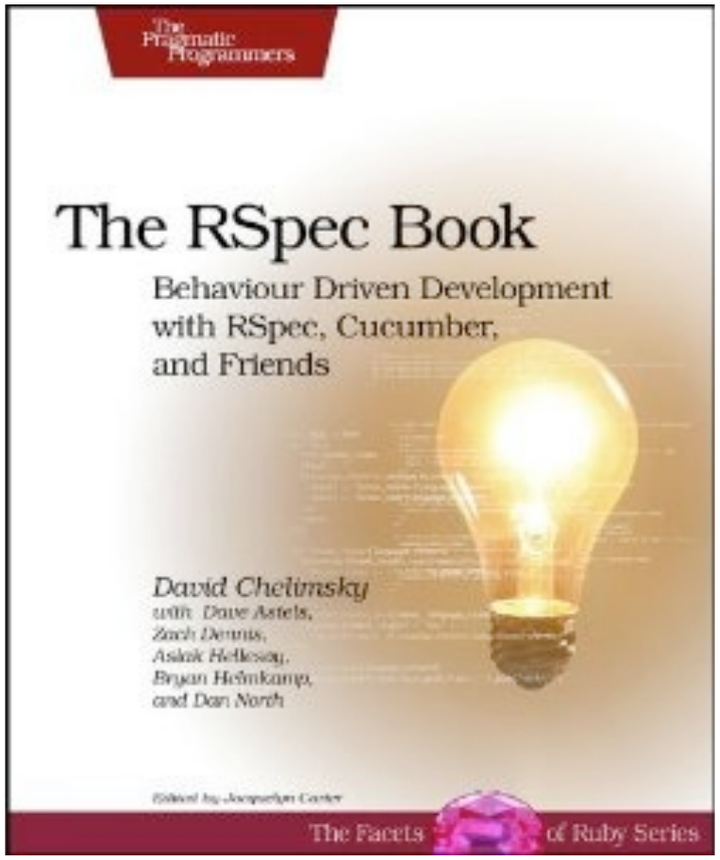
Resources

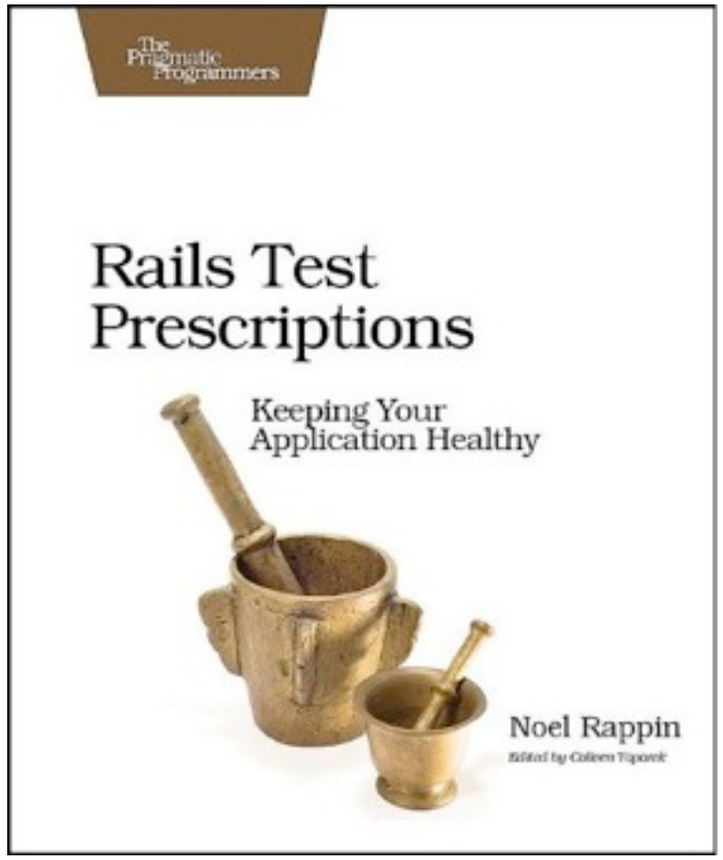




Buy these books...I stole most of the content in this section from them

Resources





http://destroyallsoftware.com
http://cleancoders.com

Questions?

Contact Info

Marc Jeanson

Redline Software Inc.

www.redlinesoftware.com

marc@redlinesoftware.com

Twitter: @marcjeanson

http://github.com/marcjeanson