Chef TDD

(how I learned to stop worrying and love to test.

AWeber

Who Am I?

- Dan Tracy
- Software Engineer at AWeber Communications
- Primarily a Python developer
- Github - djt5019 <https://github.com/djt5019>
- twitter - @djt5019

AWeber

Wednesday, April 16, 14

My name is Dan Tracy and I am a software engineer over at AWeber Communications.

I am primarily a python dev on backend apps. Dev work always in TDD - reject untested pull requests

Became interested in Chef some time ago rougher tools, testing mentality not rooted

Ops had slow, manual, fragile tests

Introduced TDD - Hit some roadbumps
- Highlighted some Infra issues - look at skeletons in our closet
- Otherwise very successful

Why Test?

AWeber

Why should we even test in the first place?
What is being gained from testing?
Is it even worth our time?

Hell yes it is.

Wednesday, April 16, 14

When you begin writing tests you eventually develop a distaste for untested code.

Why Test?

- Minimizing risk

Minimizing risk we throw over the fence when we deploy our changes

Breaking out of the negative feedback loop of being too scared to make changes

Entering a positive feedback loop where we can iterate on our code quickly

Faster to market with new features

Why Test?

- Minimizing risk
- Confidence in what we write
- Confidence in what we write

Wednesday, April 16, 14

When we minimize risk we develop confidence in our codebase, if we have confidence in what we write we are inclined to deploy it faster.

Positive feedback loop when confident

Negative feedback loop when not confident

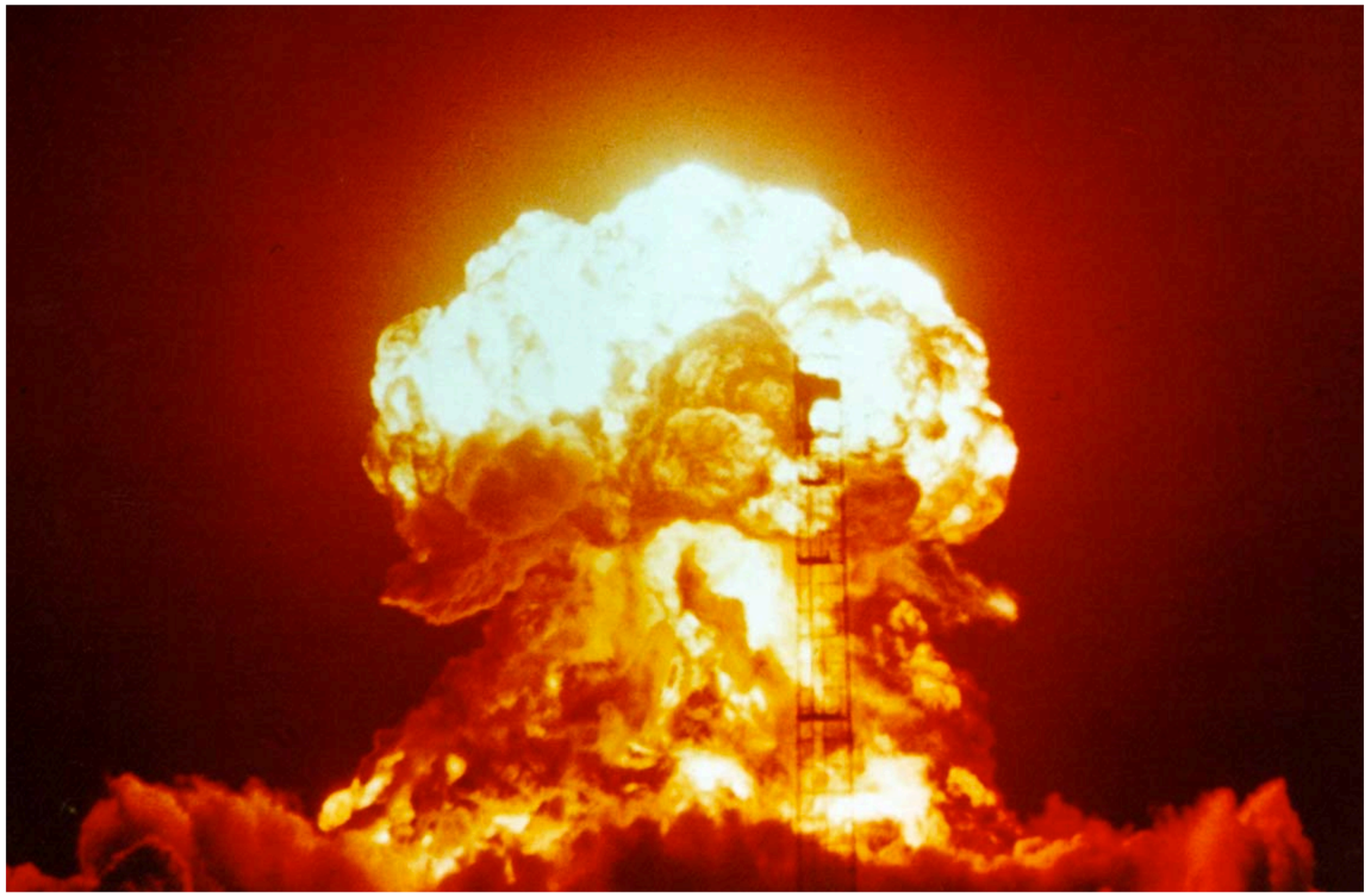
Why Test?

- Minimizing risk
- Confidence in what we write
- Ensure that we're pushing out good code

AWeber

Wednesday, April 16, 14

When we have tests we have a fairweather indicator that our changes didn't break anything else.



Don’t want this

This is what your untested code does at 3am.

Requirements for Testing

AWeber

Now you're convinced and ready to write some tests need to know requirements for writing tests

Test Requirements

- Tests should be **fast!**

Tests should be **fast.**

The word fast is relative to the types of tests that we're running.

There will be more unit tests than integration tests, and more integration tests than acceptance tests.

Negative feedback loop for slow tests

Test Requirements

- Tests should be **fast!**
- Tests should have a narrow scope

Tests should have a narrow scope.

Test bare minimum needed to satisfy your requirements.

Your function, LWRP, or code under test should be doing only one thing and doing it well.

Test Requirements

- Tests should be **fast!**
- Tests should have a narrow scope
- Tests should not depend on other tests

Wednesday, April 16, 14

Tests should not depend on other tests.

This is a big one.

Tests should not rely on state set by other tests.

You should be able to run your tests in random order with no surprises.

Coupling your tests together can make them slow, fragile, and error prone. Avoid coupled tests like the plague.

Kent Beck, Test Driven Development By Example (awesome book), big slow tests

Test Requirements

- Tests should be **fast!**
- Tests should have a narrow scope
- Tests should not depend on other tests
- Do not test implementation details, only behavior

Test the behavior of your code.

Your tests exist to ensure the behavior of your code under test acts in a manor that you deem correct.

Micromanaging the smaller details becomes very tedious and leads to fragile tests.

Mocks

Test Requirements

- Tests should be **fast!**
- Tests should have a narrow scope
- Tests should not depend on other tests
- Do not test implementation details, only behavior
- Be careful not to over-mock your tests!

Wednesday, April 16, 14

Testing implementation details usually involves mocks

Mocks are great but easily abused

Over mocking leads to you actually testing the mocks as opposed to the code you really wanted to test.

Use your mocks/stubs/doubles judiciously.

I usually mock code when communicating with a third party service. In Chef this usually means mocking out Searching or inline command within the guard statements in Chef providers. Chefspec does a great job of doing both of those things. You should check out some of their examples online.

Types of Tests

AWeber

Wednesday, April 16, 14

As I touched on earlier there are a couple of different kinds of tests. These different tests serve different purposes.

Types of Tests (Dev Side)

- Unit Testing - Single function testing

Wednesday, April 16, 14

In the dev world some of the common types of tests that we utilize are:

Simply put these tests should be lightning fast and test that your single function behaves as expected.

Types of Tests (Dev Side)

- Unit Testing - Single function testing
- Integration Testing - Interdependent modules testing

Integration tests are the tests the components that make up a user story.

They’re a step above unit tests and should test the interaction between various parts of your codebase.

These tests should run fairly fast and test the correctness of your code.

These ensure that your code behaves well with others. When writing code I usually start with these then solidify and generalize my design with unit tests

Types of Tests (Dev Side)

- Unit Testing - Single function testing
- Integration Testing - Interdependent modules testing
- Acceptance Testing - Black box/User Story testing

Acceptance tests which test the user story the code is supposed to satisfy.

Black Box, request goes in - what comes out?

API example

These tests can be slow and the scope of the test can be pretty big sometimes. Try to keep the scope of the tests as wide as it needs to be to satisfy a single user story.

Chef Analogues

- Unit Testing - Single recipe test assertions against Chef resource collection

Unit tests ensure that Chef resources are created as we expect via Chefspec. Fast tests

Not actually testing the “correctness” of the code, not the goal of unit tests.

Uncle Bob Martin - “The act of writing a unit test is more an act of design than of verification. It is also more an act of documentation than of verification. The act of writing a unit test closes a remarkable number of feedback loops, the least of which is the one pertaining to verification of function

These are used for ensuring refactoring doesn’t break anything.

Chef Analogues

- Unit Testing - Single recipe test assertions against Chef resource collection
- Integration Testing - Single node post-convergence assertions

Run against converged node via Test Kitchen and BATS

Test correctness of our recipe or recipes

Ensures everything plays nicely together and node is in a state we deem correct.

Be sure you don’t repeat the tests that you’ve written for your unit tests.

Chef Analogues

- Unit Testing - Single recipe test assertions against Chef resource collection
- Integration Testing - Single node post-convergence assertions
- Acceptance Testing - Assertions against your infrastructure or subset of it

Subset of your infrastructure. Given your new node does it interact with other nodes as you expect?

New node should play nicely.

Chef Testing Tools

- Unit Testing - Chefspec / Rspec (for plain Ruby code)
- Integration Testing - BATS and Test Kitchen
- Acceptance Testing - ??? (Open to suggestions!)

AWeber

Wednesday, April 16, 14

Chefspec/Rspec are great tools and easy to work with

Test Kitchen and BATS (Bash Automated Testing System) shell for tests. Actually really nice and clean.

In full disclosure we haven't come up with a great way to write automated acceptance tests out at AWeber that we're totally satisfied with so if you have any recommendations I would love to hear about them.

Enter TDD

AWeber

At it's simplest TDD is writing tests before writing code

Tests guide development of production code

Seemingly simple, but profound impact on code

Some of the benefits of TDD are...

Why TDD?

- Forces code to be testable

Wednesday, April 16, 14

Since we're developing our code test first we now place a strong emphasis on writing testable code.

If we're not writing testable code then our tests become a slow, horrible mess that no one will wants to run.

When tests become slow and fragile people won't want to run them which totally defeats their purpose.

Make your tests fast.

Why TDD?

- Forces code to be testable
- Promotes loose code coupling
- Looser code coupling forces you to think about your codes public interface as opposed to its underlying implementation.

Code become less interested in other implementation details of unrelated code.

Rely on public interfaces than implementation

Using functions or LWRPs instead of reproducing it's contents. LDAP User creation example

Why TDD?

- Forces code to be testable
- Promotes loose code coupling
- Confidence in production code

Gaining confidence in the code we push means that we can push code faster and with minimal risk of our code going nuclear leading to a nice little Pager storm.

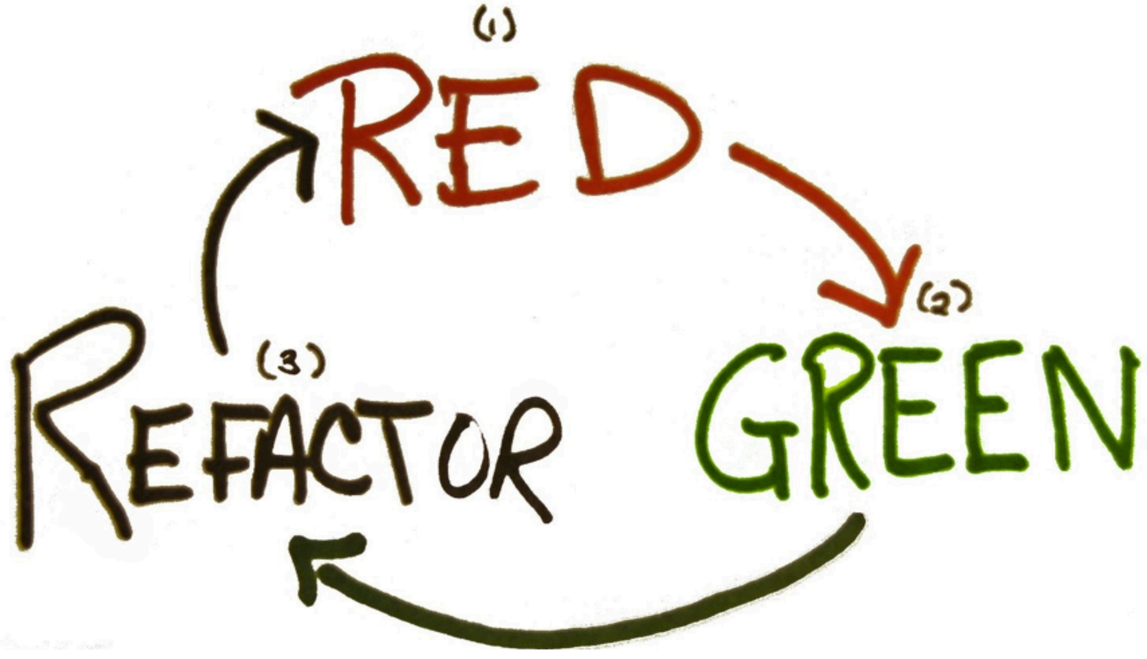
Positive feedback loop vs Negative feedback loop

Why TDD?

- Forces code to be testable
- Promotes loose code coupling
- Confidence in production code
- Introduces a quick development feedback loop...

Wednesday, April 16, 14

One of the major benefits of developing code is the introduction of a really nice feedback loop. The Red Green Refactor cycle.



Red Green Refactor Cycle

AWeber

Wednesday, April 16, 14

The red green refactor cycle.

Red - Write a failing test (only one). Motivates a new feature

Green - Make the failing test pass by writing production code
- Only write code needed to make test pass

Refactor - Clean up your mess

Writing a Test (Red/Green states)

AWeber

Walk through the Red/Green states

Refactoring is harder

```
@test "should start my supervisor process" {
    supervisorctl status | grep 'my-api'
}
```

Writing a failing integration test

AWeber

This is BATS

Supervisor program called ‘my-api’

Non-zero exit status == success

```
~> bundle exec kitchen verify
-----> Starting Kitchen (v1.2.1)
-----> Verifying <my-suite-ubuntu-precise>...
       Removing /tmp/busser/suites/bats
Uploading /tmp/busser/suites/bats/test_default.bats (mode=0644)
-----> Running bats test suite
 ✗ should start my supervisor process
   (in test file /tmp/busser/suites/bats/test_default.bats, line 2)

1 test, 1 failure
```

Failing Integration test output

AWeber

Make sure your test fails before writing code to make it pass!

```ruby
require 'chefspec'
require 'chefspec/berkshelf'


def enable_supervisor_service(resource_name)
  ChefSpec::Matchers::ResourceMatcher.new(:supervisor_service, :enable, resource_name)
end


describe 'test-cookbook::default' do
  subject(:chef_run) { ChefSpec::Runner.new.converge(described_recipe) }

  it { should enable_supervisor_service 'my-api' }

end
```

Writing a failing unit test

Chefspec Unit Test

Custom matcher needed to get test to pass (by any means needed)

Tests Supervisor resource ‘my-api’ is created

BetterSpecs.org

```
~> rspec --no-drb --order random --format d --color

test-cookbook::default
  should enable supervisor_service "my-api" (FAILED - 1)

Failures:

  1) test-cookbook::default should enable supervisor_service "my-api"
     Failure/Error: it { should enable_supervisor_service 'my-api' }
       expected "supervisor_service[my-api]" with action :enable to be in Chef run. Other supervisor_service resources



     # ./spec/default_spec.rb:13:in `block (2 levels) in <top (required)>'

Finished in 4.55 seconds
1 example, 1 failure

Failed examples:

rspec ./spec/default_spec.rb:13 # test-cookbook::default should enable supervisor_service "my-api"

Randomized with seed 4839
```

Failing unit test output

AWeber

```
include_recipe 'python'
include_recipe 'supervisor'


supervisor_service 'my-api'
```

Making it pass

```
-----> Starting Kitchen (v1.2.1)
-----> Setting up <my-suite-ubuntu-precise>...
-----> Setting up Busser
       Creating BUSSER_ROOT in /tmp/busser
       Creating busser binstub
       Plugin bats already installed
       Finished setting up <my-suite-ubuntu-precise> (0m2.33s).
-----> Verifying <my-suite-ubuntu-precise>...
       Removing /tmp/busser/suites/bats
Uploading /tmp/busser/suites/bats/test_default.bats (mode=0644)
-----> Running bats test suite
 ✔ should start my supervisor process

1 test, 0 failures
       Finished verifying <my-suite-ubuntu-precise> (0m0.96s).
-----> Kitchen is finished. (0m4.69s)
```

Passing Integration test output

AWeber

```
~> rspec --no-drb --order random
.

Finished in 4.48 seconds
1 example, 0 failures

Randomized with seed 25887
```

Passing unit test output

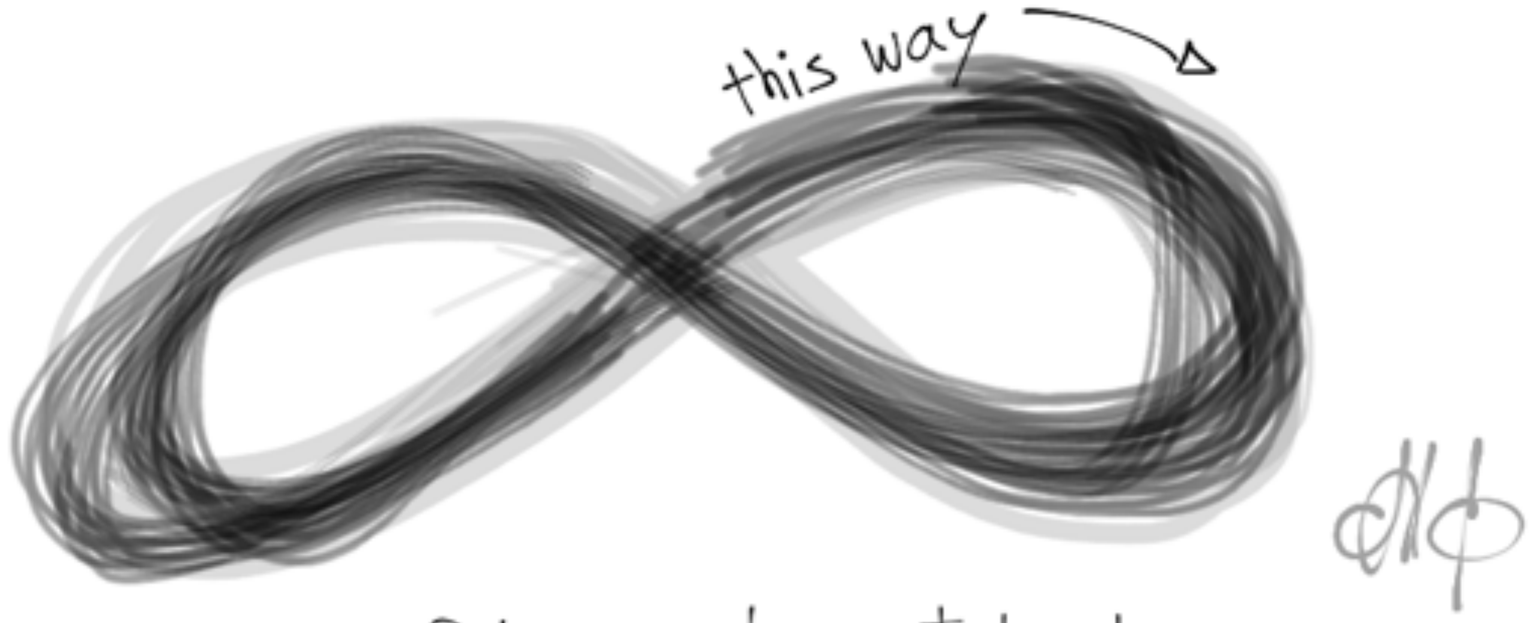
AWeber

Awesome!

We're GREEN!

Refactoring

AWeber



Always Be Refactoring

Refactoring is not a terminal state!

Why have an entirely separate state for refactoring?

Why Refactor?

- Cleaning up after your compromises getting tests to pass

Liberties can be taken to get to the green state. Now it's time to pay for your crimes.

Why Refactor?

- Cleaning up after your compromises getting tests to pass
- Paying down technical debt

Refactoring can eliminate technical debt elsewhere in the codebase

Why Refactor?

- Cleaning up after your compromises getting tests to pass
- Paying down technical debt
- Makes software easier to understand

AWeber

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.”

Why Refactor?

- Cleaning up after your compromises getting tests to pass
- Paying down technical debt
- Makes software easier to understand
- Eliminates duplication

Wednesday, April 16, 14

More on this later

How to Refactor

AWeber

How do we refactor in Chef?

- Isolate implementation behavior into LWRP, Function, etc

Hide implementation details behind a nice interface either an LWRP or function

How do we refactor in Chef?

- Isolate implementation behavior into LWRP, Function, etc
- “Utility” cookbooks for similar behavior

Isolate larger pieces of related functionality behind a separate cookbook.

Cookbooks with no recipes and only LWRPs and Functions are perfectly fine.

How do we refactor in Chef?

- Isolate implementation behavior into LWRP, Function, etc
- “Utility” cookbooks for similar behavior
- Library functions (and supporting Gems)

How not to Refactor

AWeber

How don’t we refactor?

- Do not introduce new functionality while refactoring.

Need a failing test first, go back to the red state

How don’t we refactor?

- Do not introduce new functionality while refactoring.
- Do not leave your tests in a broken state

Need to be in the green state before going back to red

How don’t we refactor?

- Do not introduce new functionality while refactoring.
- Do not leave your tests in a broken state
- Don’t the skip the refactoring step!

Eliminating Duplication

AWeber

Why eliminate duplication?

- Duplication is evil!

Why eliminate duplication?

- Duplication is evil!
- Keep your code DRY!

Why eliminate duplication?

- Duplication is evil!
- Keep your code DRY!
- Multiple places to maintain when requirements change

Why eliminate duplication?

- Duplication is evil!
- Keep your code DRY!
- Multiple places to maintain when requirements change
- Repetition eventually leads to inconsistency in the codebase

Back to the Red State

AWeber

Tightening the feedback loop

AWeber

Guard

- Rspec Guard for Chefspec tests

- Kitchen Guard for Test Kitchen tests

TL;DR

- Try out TDD to see if it works for you
- **DON'T** Commit copied and pasted code
- **DON'T** Forget to refactor your code **and** your tests
- **DON'T** Forget to see your test fail first!

Questions?

AWeber

- Photo Credits
- http://en.wikipedia.org/wiki/File:Operation_Upshot-Knothole_-_Badger_001.jpg
- http://agileinaflash.blogspot.com/2009/02/red-green-refactor.html
- http://twentytwowords.com/guns-in-movies-replaced-with-thumbs-ups-15-pictures/
- http://stfalcon.com/blog/post/phpstorm-refactoring
- http://guardgem.org/