Chef TDD

Or how I learned to stop worrying and love to test

Who Am I?

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

My name is Dan Tracy and I am a software engineer over at AWeber Communications. I am primarily a backend python dev but became interested in Chef some time ago. I don't exactly remember why I started messing with Chef but was quickly enticed by it. When I started out with Chef the testing landscape looked much different than what it does today. The tools were in a rougher shape and the testing mentality had not totally taken root just yet. Today testing is more in the forefront of our minds when developing code.

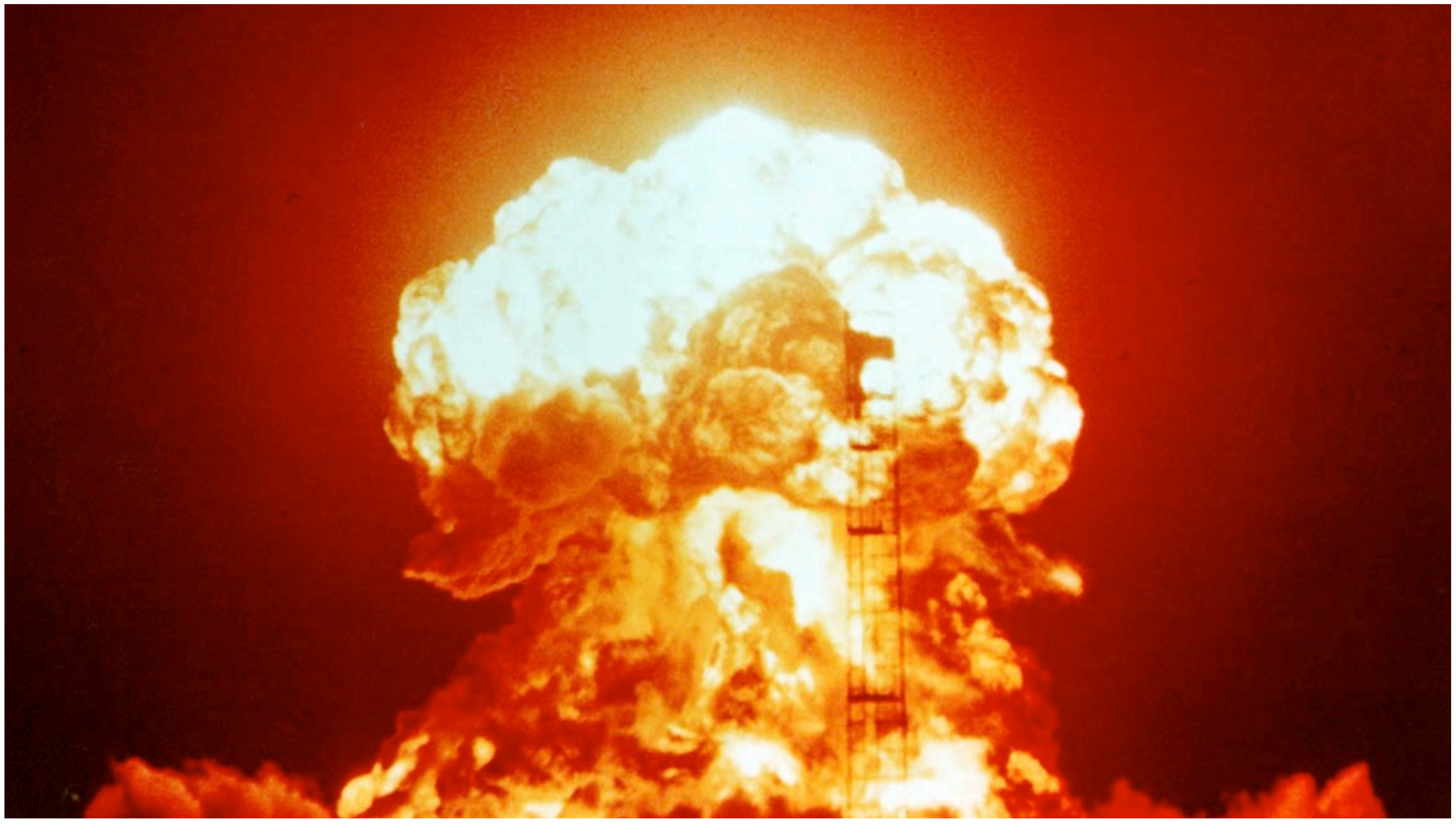
Why Test?

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.

Why Test?

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



Don’t want this

This is what your untested code does at 3am

Enter TDD

AWeber

Simply put Test Driven Development is using tests to guide the development of production code. While the definition is pretty simplistic actually using it in practice has some profound impacts on our code. Some of the benefits of TDD are...

Why TDD?

- Requires you to develop code in a testable fashion
- Looser code coupling
- Gain confidence in production code through tests
- Introduces a quick development feedback loop, Red/Green/Refactor Cycle

Monday, April 7, 14

Some of the benefits that stem from using TDD are <bullet points>

* 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.

* Lose code coupling , this ties into the previous point. The benefit of this mean that components of code become less interested in other implementation details of unrelated code. Instead of becoming totally concerned with the underlying implementation of other pieces of code we can instead use the public interface of the code that we want to use. In the context of chef this means making use of an LWRP instead of reproducing it's contents. Looser code coupling forces you to think about your codes public interface as opposed to its underlying implementation.

* 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.

* 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

TDD 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!

AWeber

Monday, April 7, 14

* Tests should be **fast**. The word fast relative to the types of tests that we're running. Unit tests should be near instantaneous. Integration tests should run a couple of minutes. Acceptance tests will be the slowest of all your tests since they require various parts of your infrastructure to be up and running. There will be more unit tests than integration tests, and more integration tests than acceptance tests.

* Tests should have a narrow scope. What this means is that your test does the bare minimum needed to satisfy your requirements. Your function, LWRP, or code under test should be doing only one thing and doing it well.

* 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 makes them fragile and error prone. Avoid coupled tests like the plague.

* 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. The reason it leads to fragile tests is because you usually begin to rely on mocking out the internals of your code. This can lead to my next point... over mocking

* Over mocking leads to you actually testing the mocks as opposed to the code you really wanted to test. Mocks are great when needed, but they're not always needed. Use your mocks/stubs/doubles judiciously. I usually mock code when communicating with a third party service or shelling out. 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

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

Types of Tests

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

Monday, April 7, 14

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

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

* 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.

* Acceptance tests which test the entirety of the user story the code is supposed to satisfy. The way I've always done Acceptance testing is treating the code like a black box. For example, if I give the API some input I should get this output. 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
- Integration Testing - Single node post-convergence assertions
- Acceptance Testing - Assertions against your infrastructure or subset of it

The testing parallels with Chef look like this:

* Unit tests ensure that Chef resources are created as we expect. We use the awesome Chefspec library for this since it will go through the compilation step of the chef run but not really create the resources themselves. This makes these tests pretty fast.

* Integration tests. For our integration tests we use Test Kitchen and BATS. You can also use any of the other testing utilities offered by Test Kitchen. The goal here is that we're asserting on a fully converged VM. Be sure you don't repeat the tests that you've written for your unit tests.

* Acceptance test should test the a subset of your infrastructure to ensure your newly converged node plays nice with it. In full disclosure we haven't come up with a great way to write acceptance tests out at AWeber that we're totally satisfied with so if you have any reccomendations I would love to hear about them.

Chef Testing Tools

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

Writing a Test
(Red/Green states)

AWeber

Writing a failing integration test

Making the integration test pass

Writing a failing unit test

Making the unit test pass

Refactoring

AWeber

Why Refactor?

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

Why eliminate duplication?

- Duplication is evil!
- Don’t repeat yourself!
- Multiple places to maintain when requirements change
- Repetition eventually leads to inconsistency in the codebase

How do we refactor in Chef?

- Break repetition into functions
- Isolate implementation behavior into LWRP or Definition
- “Utility” cookbooks for similar behavior
- Library functions (and supporting Gems)

How don’t we refactor?

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

Tightening the feedback loop

AWeber

Guard

- Rspec Guard for Chefspec tests
- Kitchen Guard for Test Kitchen tests

Conclusion

Overview

- **DO** - Try out Chef TDD to see if it works for you
- **DO** - Write loosely coupled code
- **DON'T** - Commit copied and pasted code
- **DON'T** - Forget to refactor
- **DON'T -** Forget to **watch** your test fail first!

Questions?

AWeber