Open in new tab

Test Driven Development

Lesson Duration: 90-120 minutes

Learning Objectives

- Know what a test is.
- Know what TDD is.
- Understand the benefits of writing tests first
- Be able to write unit tests, and make them pass

What is a Test?

Tests can be thought of in a few ways

- a piece of code that verifies that our application is working correctly
- an example that proves our application works as expected

e.g.

```
def test_one_plus_one
  expected_value = 2
  actual_value = 1 + 1
  assert_equal(expected_value, actual_value)
end
```

What Is This Test Verifying?

This test verifies that Ruby's `+` operator works as we expect it to. In the real world we know `1 + 1 == 2`, and that test verifies that Ruby works like the real world. i.e., using the example of `1 + 1 = 2` we check that our code meets expectations.

We wouldn't normally need a test for code we didn't write (in part because we assume the developers who wrote Ruby have tested it themselves) but it can be helpful as an example.

What Does It Mean to "Test-drive" Development?

TDD stands for Test-Driven Development.

At its simplest, it is just writing tests first, then making them pass.

If we want to be a bit more precise about it, TDD lets us design our application and think about what functions we need to write, and what effect we want them to have, before we write them.

We do this by writing a test about the function we *wish* we had, even though we know it (probably) won't pass. So if I want a function that adds one to any number, I'd start by pretending I already had that function, and write out how I wanted it to look.

```
def test_add_one_5_is_6
  expected_value = 6
  actual_value = add_one(5)
  assert_equal(expected_value, actual_value)
end
```

We continue to think about what functions we need, and what effect we want them to have after each passing test. In theory (and, as you'll hopefully see, in practice) this leads to a well-designed, easy to read, application.

Why Bother Test-Driving?

Let's say I ask you to write a function called `fizz_buzz`. That function will take in an `Integer` and:

- return "Fizz" if the number is divisible by 3 (e.g. `fizz_buzz(3) == "Fizz"`)
- return "Buzz" if the number is divisible by 5 (e.g. `fizz_buzz(5) == "Buzz"`)
- return "FizzBuzz" if the number is divisible by 3 **and** 5 (e.g. `fizz_buzz(15) == "FizzBuzz"`)
- return the input as a String otherwise (e.g. `fizz_buzz(7) == "7"`).

Where Do You Start?

Your instinct is probably to start thinking about dividing the input, or writing an `if` statement, or even a `case`. That's cool - that's because you're thinking about the solution.

So let's say you do that and you write your code. How do you know if your function works?

Unfortunately, you don't.

Maybe you write some tests after the code's been written, and that gives you some confidence. But now how do you know your tests are right?

They key point here is you've never seen your tests fail. So you don't know if they're good tests. More to the point, you don't even know if they work.

And that's the key to test-driven development. We write a test, we watch it fail, and then we make it pass. That way we have confidence in the tests, because we've seen them fail, and then made them pass, and we have confidence in the code because we've got a lot of passing tests that tell us exactly what the code does.

Okay, You've Convinced Me. How Do I TDD?

So let's go back to that "FizzBuzz" example. What's the simplest test we could write to start us off?

Something like:

```
# fizz_buzz_spec.rb
def test_fizz_buzz_3_returns_fizz
  assert_equal("Fizz", fizz_buzz(3))
end
```

This is weird: we don't have a `fizz_buzz` function, but in our test it looks like we do.

Oh well, let's run the test and see what happens!

It fails, obviously. But the important part is the error message, which tells us why it failed. As we already knew, the function `fizz_buzz` didn't exist. So now we can create it!

```
# fizz_buzz.rb
def fizz_buzz
end
```

Well that solves that problem. Shall we run it again?

> repeat the process, failing that test and fixing the error, until…

```
# fizz_buzz.rb
def fizz_buzz(number)
  return "Fizz"
end
```

The test passes!

So is our application done? Let's look at what we wanted it to do again.

Clearly we're far from done. But all our tests pass. So now? Write another test!

(…repeat until done…)

FAQs

> These are questions that come up during the lesson.

How do I know what test to write next?

This is a tough question. Sometimes it can seem random, especially when you watch someone else doing TDD. Really, it comes down to one question: does the code do what it's supposed to yet? So, in the FizzBuzz example, once we knew that `fizz_buzz(3) == "Fizz"` was `true` it made sense to me to start thinking about what `fizz_buzz(5)` would return. But you might have preferred to look at `fizz_buzz(1)` or `fizz_buzz(15)` or even `fizz_buzz(196212)`. As long as you can explain *why* that test makes sense then it's probably a reasonable test.

How do I know when I've written enough tests?

The more you test-drive your code, the easier it will be to know when you've written the right amount of tests. As a simple guide, though, if you can't write another failing test, then you're probably done.

When do I stop writing tests?

This is related to the above question, I guess. Really the best way to know is when you can't find another failing test to write. Then again, if you rush ahead and write too much code without testing, you might find it hard to write a failing test. That doesn't necessarily mean that your code is well-tested.

For now, I'd recommend not worrying too much about this. You'll get a feeling for it as you go on.

Do I always need to write tests?

You know what? Probably not.

I never write code without writing tests first. But I know lots of engineers who write tests after they write their code. Personally I find that really dull. It's much more exciting for me to have the constant validation of making tests pass throughout the day.

If you don't like writing tests first, that's okay. Maybe you'll learn to like it or maybe you won't. But keep in mind a lot of employers will ask about testing, and it'll be helpful to have a reasonable knowledge of how to write good tests.

Summary

Do we:

- Know what a test is? (Yes!)
- Know what TDD is? (Yes!)
- Understand the benefits of writing tests first? (Yes!) Are we:
- Able to write unit tests, and make them pass? (Yes!)

Published with GitHub Pages