Testing

TDD vs Debugging

Conventional	TDD
Write 10s of lines, run, hit bug: break out debugger	Write a few lines, with test first; know immediately if broken
Insert printf's to print variables while running repeatedly	Test short pieces of code using expectations
Stop in debugger, tweak/set variables to control code path	Use mocks and stubs to control code path
Dammit, I thought for sure I fixed it, now I have to do this all again	Re-run test automatically

Mocks

- Set up the mock with `double`
- Set up stubs for methods with `allow(<double>).to receive(:<method>).and_return(<return value>)`

Mock Train Wreck

- Have to pass mocks to mocks when chaining values
- Can get super complicated because you keep having to mock dependencies of initial mock

Fixtures

- Fixture $\triangleq$ statically preload some known data into database tables
- DB wiped and reloaded with fixtures before each spec
- Use cases:
  - Truly static data (eg: configuration info that never changes like API keys)
  - Easy to see all test data in one place
- Cons:
  - May introduce dependency on fixture data
- Usually put in `spec/fixtures`
- Are `.yml` files
- Eg:

```
# Fixture file.
milk_movie:
    id: 1
    title: Milk
    rating: R
    release_date: 2008-11-26
documentary_movie:
    id: 2
    title: Food, Inc.
    release_date: 2008-09-07
```

```
# Test file.
fixtures :movies
it 'finds movie by title' do
    movie = movies(:milk_movie)
    # etc...
end
```

Factories

- Factory ≙ create only what you need per-test
- Can use `FactoryBot` gem
- `Faker` gem is helpful for generating random fake data
- In `spec/factories`
- Are `.rb` files
- Eg:

```
# Factory file.
FctoryBot.define do
    factory :movie do
        title 'A Fake Title'
        rating 'PG'
        release_date { 2.years.ago }
    end
end

# Test file.
it 'should include rating and year' do
    # Can also use create to save it to db.
    movie = FactoryBot.build(:movie, :title => 'Milk')
    # etc...
end
```

Factories with Associations

```
# Factory file.
FactoryBot.define do
    factory :moviegoer do
        sequence(:email) { |n| "user#{n}@fakemail.com" }
        name { Faker::Name.name }
    end
    factory :review do
        potatoes 3
        description 'it was okay'
        # Create an association on instantiation.
        association :moviegoer
    end
end

# Test file.
review = create(:review)
review = create(:review, :movie => create(:movie, :rating => 'R'))
```

Stubbing the Internet

- Important for SOA
- Can stub at the level of the class
- Can stub at the level of http (`allow(Net::HTTP).to receive(:get).with('<full URI>'.and_return(<expected return>)`)
- Can use `webmock` gem
  - Can also use `VCR` gem which will initially make the request then use that response as a stub value
- For unit testing, stub nearby
  - Maximum isolation
  - Fast

- For integration testing, stub far away
  - Test as many interfaces as possible
  - Use things like `webmock`
  - Run against sandbox / stage

Amount of Testing

- 120-150% of actual code
- Often much higher for production systems
- Coverage measurement

Coverage

- Coverage types:
  - S0 := call every method
  - S1 := call every method from every call site
  - C0 := every line touched
  - C1 := every branch in both directions
  - C1+decision coverage := every subexpression in conditional
  - C2 := every path (difficult and disagreement on how valuable)
- Use to identity untested or undertested parts of code
- Need both integration and unit

Other testing Terms

- Mutation testing ≙ if introduce deliberate error in code, does some test break
- Fuzz testing ≙ throw random input at code
  - Find ~20% MS bugs, crash ~20% Unix utilities
  - Tests the app the way it wasn't meant to be used
- DU-coverage ≙ is every pair executed?
  - DU := define and use
- Black-box vs white-box / glass-box ≙ does the test know about the implementation?
  - White box ≙ trying to trick it / testing edge cases
  - Black box ≙ trying random values

TDD Summary

- Use red - green - refactor and always have working code
- Test one behavior at a time using seams
- Use it 'placeholders' or pending to node tests you know you'll need
- Read and understand coverage reports
- 'Defense in depth' ≙ do not rely too heavily on any one kind of test