Lets start testing

Chennai.rb meet, 18july2015

Lets start testing

The agenda:

- The Premise
- Rspec basics
- Testing models
- Factories with Factory-girl
- Stubbing
- Testing UI with Capybara
- Testing javascript

Speaker

Abhishek Yadav

@h6165

Freelance Ruby programmer

Volunteer at Chennai.rb

The premise

The Premise

- Here: Testing == Automated testing
- Why test ?
- What about non-automated tests ?
- Who writes the tests ?
- When to start testing ?
- TDD

The Premise: why test?

- Verification : *Does this feature work?*
- Regression spotting: *Something broke something else*
- Specification: *The machine should work this way*
- Documentation: *Test describes how it should behave*
- Developer feedback:
  - What I thought will not work
  - What I thought and what client wanted are different
- Code design: *Decoupling, cleaner interfaces, better OOP*

The Premise: Manual testing

- The default
- Easy to start – needs product knowledge and common sense
- Tester.price < Developer.price (generally)
- Can cover UI-aesthetics

The Premise: Semi automated testing

- Need human involvement, but parts are automated
- Examples:
  - Record and playback
  - Export and compare
- Quicker, less boring

The Premise:
Automated tests: why

- Automated
- Repeatable
- Pipeline-able
- [Rspec] Easy (after initial threshold)
- [Rspec] Fun

  +
- Benefits of TDD

The Premise: Who writes the tests

- The developer

  'Testers should write tests, not me' is an outdated excuse, and has no place in Ruby world

The Premise:
When to start testing

As early as possible

No tests are okay for:

Prototype level projects that may not live after 2 months

Manual QA is simple and cheap

The Premise:
When to start testing

Its too late signs**:**

Regressions.

Implementation slowness.

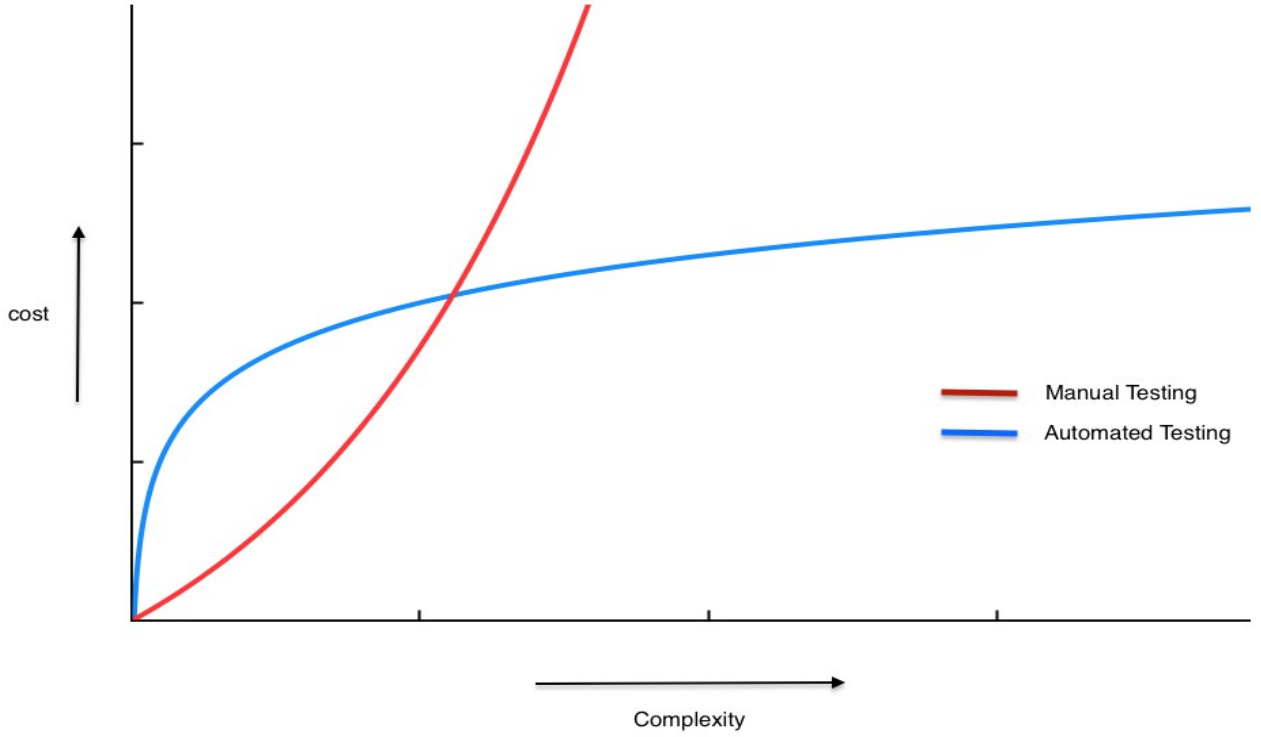
Poor quality in general

Developer stress

The Premise: When to start testing

At some point,

cost of manual testing exceeds the cost of automated testing



The premise: TDD

- Write test first, then the code
- Needs specification beforehand as examples
- Ideal fit when specs are available (software consulting with requirements)
- Leads to good code design

The premise: TDD

- This talk isn't about TDD
- Its about testing, using a popular tool: rspec, generally applicable to Rails web application projects
- This may be a stepping stone towards TDD

Rspec basics

Rspec Basics

- It is a test framework and a test runner
- We write test-cases using Rspec recommended syntax
- For verification, we use Rspec provided assertion methods
- We create data on our own (or using factories or fixtures) – Rspec has no involvement there

Rspec basics

- Test-case syntax: **it** method:

```
it 'should give 5 for 2 and 3' do
    # Invoke the code (calculator here)
    # Verify the result
end
```

Rspec basics

The syntax: **describe** methods

- Encloses the test cases
- Names the thing under test

```
describe 'Addition-calculator' do
        it 'gives 5 for 2 and 3' do
        end

        It 'gives 3 for 3 and 0' do
        end
end
```

Rspec basics

- The syntax: verification: expect-to-eq

expect(result-value).**to eq**(expected-value)

Example:

```
Rspec.describe 'MyCalculator' do
  it 'should give 5 for 2 and 3' do
      sum = MyCalculator.add(2, 3)
      expect(sum).to eq(5)
  end
end
```

Rspec basics

- The syntax: Thats it.
- There are many more methods, concepts and conveniences
- But to get started, this much is just fine

Rspec basics

- Running it:
  - The code under test should have been loaded before the describe block starts
  - **rspec test_file.rb**

Demo: 1_spec.rb

Rspec basics

- Notes about setup:
  - Proper setup requires a Gemfile with rspec in it
  - Then **bundle install ---binstubs**
  - Then **bin/rspec –init** creates spec_helper and spec/
  - Tests are kept in a spec directory
  - All test files should require the spec_helper on top

Rspec with Rails

Rspec with Rails

- What can we test -
  - Models
  - Controllers
  - Views ?
  - Rake Tasks ?
  - Javascript
  - Overall UI ?

Rspec with Rails

- What can we test, IMO -
  - Models Yes
  - Controllers Almost
  - Views Maybe
  - Rake Tasks Maybe
  - Javascript Yes
  - Browser UI Yes
  - API Yes

Note: lib/ is also Yes

Rspec with Rails

Rspec with Rails

Rules of the game:

- Rails uses MiniTest by default – choose one
- Rails has separate ENV and database for testing

- Rspec tests are in spec/ folder
- Rspec config is in spec/spec_helper and spec/rails_helper files
- spec/rails_helper is required in all test files

Rspec with Rails

Rules of the game:

- Tests run:
  - rake *for all*
  - bin/rspec spec *for all*
  - bin/rspec spec/models *for tests in models folder*
  - bin/rspec spec/models/user_spec.rb *for User model*
  - bin/rspec spec/models/user_spec.rb:15 *for test on line 15*

Rspec with Rails

- Rule of the game
    - Test-db should be empty when test is not running
    - Every test-case must create its data in test-db, and wipe it out when done. (Rspec does this for us)
    - Every test-case must be independent. It should not share any data/variables with another test. So we can run tests in any order. (Rspec does that too)

Rspec with Rails

- Rule of the game
    - Since Models are tightly tied to ActiveRecord, model tests are not always unit test (Though this doesn't matter much)
    - We try to avoid database interaction in tests, as much as possible

Rspec with Rails

- Rule of the game:
    - We run the tests on developer computers or separate test servers (called CI server too)
    - We DO NOT run tests on staging or production
    - [Again] We DO NOT maintain any data in test-db

Rspec with Rails

- Demo:
  - blog: post_spec

Rspec with Rails

- Post demo:
  - Creating data can get difficult
  - References to other classes, what to do with them?

Factories with Factory girl

Factories with Factory girl

- Factories are one way to create data for tests
- Data creation can become difficult when there are validations
- There are other Factory gems:
  - Machinist
  - Fabrication

Factories with Factory-girl

Rails default is Fixtures. It is:

- Data expressed in YAML files,
- Loaded in the DB at the start of test suite
- much faster than Factories
- Are global
- Must be carefully maintained with database migrations

Factories with Factory-girl

```
factory :registered_user, class: User do
        sequence(:username){|n| "registered-#{n}" }
        first_name 'example-fn'
        last_name 'example-ln'
        date_of_birth 35.years.ago
        gender 'Male'
        phone_number { "1234567#{rand(9)}89" }
        sequence(:email) { |n| "registered#{n}@example.com" }
        password 'example-password'
        password_confirmation {|u| u.password }
        registration_state 'registered'
        company { create(:company)  }
        after(:create) do |user, e|
            user.add_role(:general_user)
        end
end
```

To be continued