TESTING

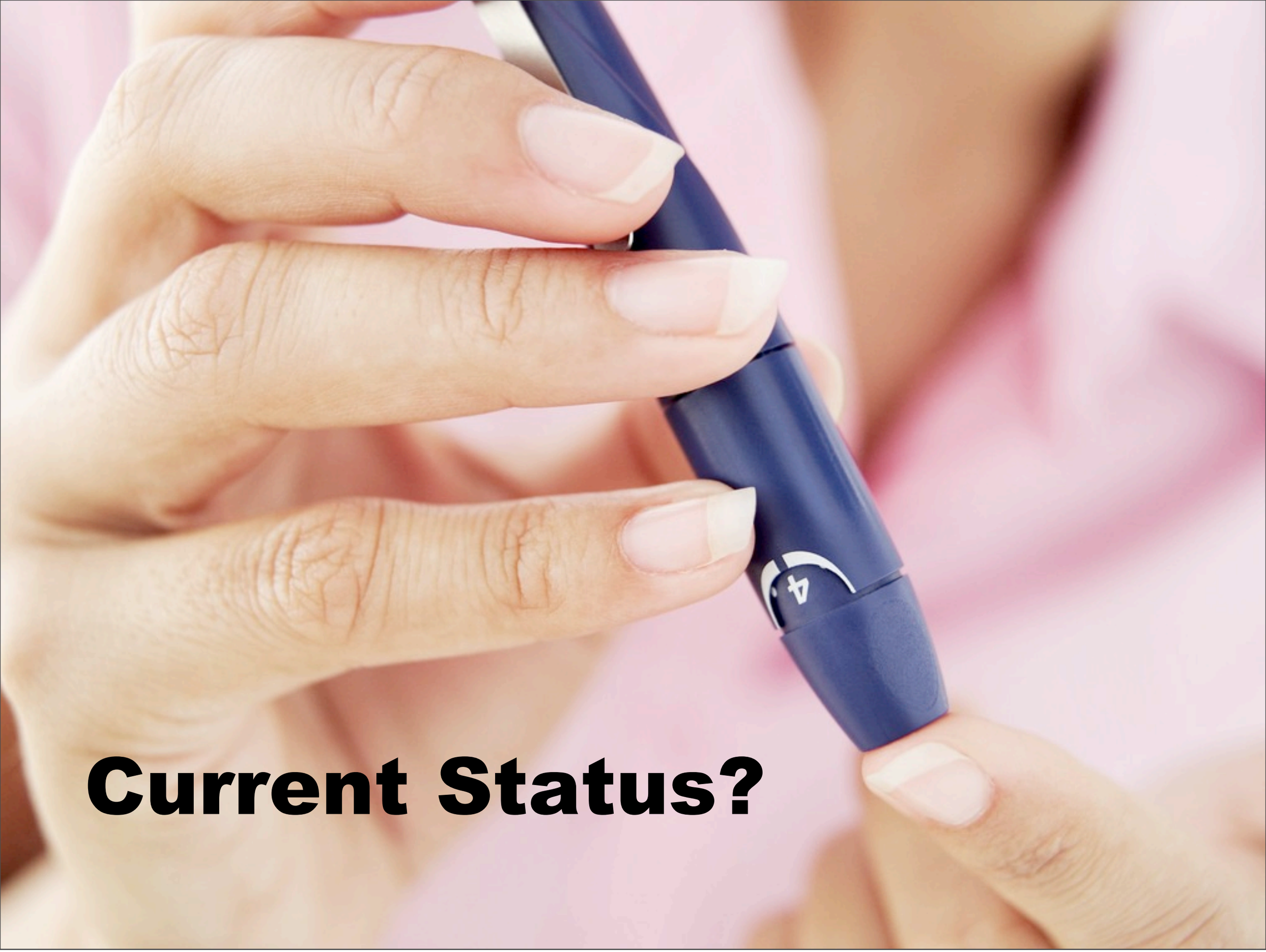
I FIND YOUR LACK OF TESTS DISTURBING.

DIYDESPAIR.COM

Jason Meridth


Without
Tests

Responsibility



Current Status?

Core of TDD

3

- **RED**: test fails
- **GREEN**: test passes
- **REFACTOR**
  - ↳ CLEAN Code + Tests

```
$ rails my_app
cd my_app
rake db:create
rake db:create RAILS_ENV=test
script/generate model User name:string
rake db:migrate
rake db:test:prepare
rake
```

```
Started
.
Finished in 0.07929800000000001 seconds.

1 tests, 1 assertions, 0 failures, 0 errors
```

generated test

my_app/test/unit/user_test.rb

```
require 'test_helper'

class UserTest <
ActiveSupport::TestCase
  # Replace this with your real tests.
  test "the truth" do
    assert true
  end
end
```

Write a failing test

my_app/test/unit/user_test.rb

```
require 'test_helper'

class UserTest < ActiveSupport::TestCase
  test "should require a name" do
    user = User.new
    assert_not_nil user.errors.on(:name)
  end
end
```

run tests - see failure

```
Loaded suite my_app/test/unit/user_test
Started
.
Finished in 0.055605 seconds.

 1) Failure:
test_should_require_a_name(PostTest) [/
test/unit/user_test.rb:7]:
<nil> expected to not be nil.

1 tests, 1 assertions, 1 failures, 0 errors
```

make it pass

my_app/test/unit/user_test.rb

```
class User < ActiveRecord::Base
  validates_presence_of :name
end
```

run tests - see passing

```
Loaded suite my_app/test/unit/user_test
Started
.
Finished in 0.049392 seconds.

1 tests, 1 assertions, 0 failures, 0 errors
```

Refactor

Shoulda

install Shoulda as Plugin

```
$ script/plugin install git://github.com/thoughtbot/shoulda
Initialized empty Git repository in my_app/vendor/plugins/shoulda/.git/
remote: Counting objects: 221, done.
remote: Compressing objects: 100% (183/183), done.
remote: Total 221 (delta 33), reused 179 (delta 17)
Receiving objects: 100% (221/221), 83.09 KiB, done.
Resolving deltas: 100% (33/33), done.
From git://github.com/thoughtbot/shoulda
 * branch            HEAD       -> FETCH_HEAD
```

unit

test::unit test

my_app/test/unit/user_test.rb

```ruby
require 'test_helper'

class UserTest < ActiveSupport::TestCase
  test "should require a name" do
    user = User.new
    assert_not_nil user.errors.on(:name)
  end
end
```

shoulda version

my_app/test/unit/user_test.rb

```
require 'test_helper'
require 'shoulda'

class UserTest < ActiveSupport::TestCase
  context "a user" do
    should_validate_presence_of :name
  end
end
```

run shoulda test

```
Loaded suite my_app/test/unit/user_test
Started
.
Finished in 0.066046 seconds.

1 tests, 1 assertions, 0 failures, 0 errors
```

custom macro

my_app/test/shoulda_macros/should_succeed.rb

```
class Test::Unit::TestCase
  def self.should_succeed
    should "succeed" do
      assert true
    end
  end
end
```

shoulda version using macro

my_app/test/unit/user_test.rb

```
require 'test_helper'
require 'shoulda'

class UserTest < ActiveSupport::TestCase
  context "a user" do
    should_validate_presence_of :name
    should_succeed
  end
end
```

run shoulda test with macro

```
Loaded suite my_app/test/unit/user_test
Started
..
Finished in 0.137248 seconds.

2 tests, 2 assertions, 0 failures, 0 errors
```

custom macro via extend

my_app/test/shoulda_macros/should_succeed.rb

```
module PresentationMacros
  def should_succeed
    should "succeed" do
      assert true
    end
  end
end

class Test::Unit::TestCase
  extend PresentationMacros
end
```

.../test/unit/image_test.rb

```ruby
require 'test_helper'

class ImageTest < Test::Unit::TestCase
  context "An image instance" do
    should_validate_presence_of :title, :summary
    should_belong_to :album
    should_have_many :thumbnails
  end
end
```

.../app/model/image.rb

```
class Image < ActiveRecord::Base
  belongs_to :album
  has_many :thumbnails, :foreign_key => 'parent_id'

  validates_presence_of :title, :summary

  ...#attachment_fu code
end
```

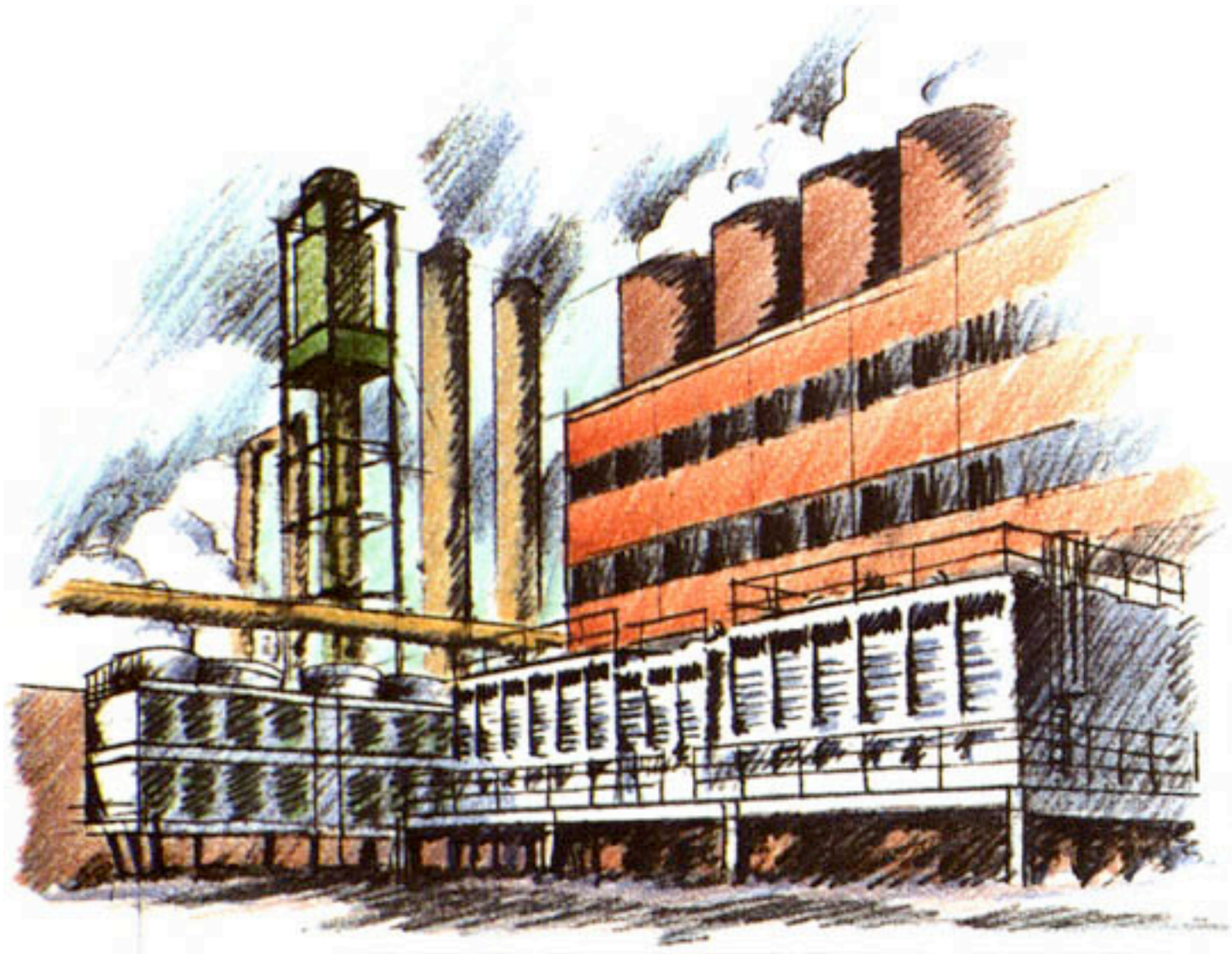
.../test/unit/album_test.rb

```ruby
require 'test_helper'

class AlbumTest < Test::Unit::TestCase
  context "An album instance" do
    should_validate_presence_of :name, :description
    should_have_many :images
  end
end
```

.../app/model/album.rb

```
class Album < ActiveRecord::Base
  has_many :images

  validates_presence_of :name, :description
end
```

factory_girl



.../test/factories.rb

```
Factory.define :image, :class => Image, :default_strategy => :build do |i|
  i.title "Test Title"
  i.summary "<h1>Test Summary</h1>"
  i.content_type "image/jpeg"
  i.size 10000
  i.filename "/path/to/image.jpeg"
end

Factory.define :album, :class => Album, :default_strategy => :build do |a|
  a.name "Test Name"
  a.description "<h1>Test Description</h1>"
  a.images do |image|
    [image.association(:image)]
  end
end
```

.../test/factories.rb

```
Factory.define :user, :class => User, :default_strategy => :build do |u|
  u.first_name "John"
  u.last_name "Doe"
  u.username "jdoe"
  u.password_confirmation "secret"
  u.password "secret"
  u.email "jdoe@domain.com"
end

Factory.define :section, :class => Section, :default_strategy => :build do |s|
  s.title "Test Title"
  s.content "<h1>Test Content</h1>"
end
```

.../test/test_helper.rb

```
ENV["RAILS_ENV"] = "test"
require File. \
  expand_path(File.dirname(__FILE__) + "/../
config/environment")
require 'test_help'
require 'factory_girl'
require 'shoulda'
require 'mocha'
require 'factories'
```

functional

```ruby
class AdminController < ApplicationController
  layout false

  def login
    if request.post?
      user = User.authenticate(params[:username], params[:password])
      if user
        session[:user_id] = user.id
        redirect_to(:controller => "sections", :action => "index" )
      else
        session[:user_id] = nil
        flash.now[:message] = "Invalid user/password combination"
      end
    end
  end

  def logout
    session[:user_id] = nil
    flash[:notice] = "Logged out"
    redirect_to(:action => "login" )
  end
end
```

```ruby
require 'test_helper'

class AdminControllerTest < ActionController::TestCase
  context "logging in" do
    context "with valid credentials" do
      setup do
        @user = Factory(:user, :id => 1234)
        User.stubs(:authenticate).returns(@user)
        post :login
      end

      should_set_session(:user_id) { @user.id }
      should_respond_with :found
      should_redirect_to("the sections screen") { "/sections" }
      should_not_set_the_flash
    end
  end
end
```

```ruby
context "logging in" do
  ...

  context "with invalid credentials" do
    setup do
      User.stubs(:authenticate).returns(nil)
      post :login
    end

    should_set_session(:user_id) { nil }
    should_respond_with :success
    should_render_template :login
    should_set_the_flash_to "Invalid user/password combination"
  end
end
```

```ruby
context "logging out" do
  setup do
    post :logout
  end

  # setup { post :logout }

  should_set_session(:user_id) { nil }
  should_respond_with :found
  should_set_the_flash_to "Please log in"
end
```

```ruby
class SectionsController < ApplicationController
  layout "manage"

  # GET /sections
  # GET /sections.xml
  def index
    @sections = Section.find(:all, :order => :title)

    respond_to do |format|
      format.html # index.html.erb
      format.xml  { render :xml => @sections }
    end
  end

  # GET /sections/1
  # GET /sections/1.xml
  def show
    @section = Section.find(params[:id])

    respond_to do |format|
      format.html # show.html.erb
      format.xml  { render :xml => @section }
    end
  end

  # GET /sections/1/edit
  def edit
    @section = Section.find(params[:id])
  end

  ...
end
```

```ruby
require 'test_helper'

class SectionsControllerTest < ActionController::TestCase
  context "before logging in" do
    context "on GET to :show" do
      setup { get :show }

      should_respond_with 302
      should_redirect_to("the login screen") { "/admin" }
      should_set_the_flash_to "Please log in"
    end

    context "on GET to :index" do
      setup { get :index }

      should_respond_with 302
      should_redirect_to("the login screen") { "/admin" }
      should_set_the_flash_to "Please log in"
    end
  end
end
```

```ruby
class SectionsControllerTest < ActionController::TestCase
  ...

  context "after logging in" do
    setup do
      @user = Factory(:user, :id => 12345)
      User.expects(:find_by_id).returns(@user)

      @section = Factory(:section, :id => 23456)
    end

    context "on GET to :index" do
      setup do
        Section.expects(:find). \
          with(:all, { :order => :title }).returns([@section])
        get :index
      end

      should_respond_with :success
      should_render_template :index
      should_not_set_the_flash
    end
  end
end
```

```ruby
class SectionsControllerTest < ActionController::TestCase
  ...

  context "after logging in" do
    setup do
      @user = Factory(:user, :id => 12345)
      User.expects(:find_by_id).returns(@user)

      @section = Factory(:section, :id => 23456)
    end

    ...

    context "on GET to :show" do
      setup do
        Section.expects(:find). \
          with(@section.id.to_s).returns(@section)
        get :show, :id => @section.id
      end

      should_assign_to :section
      should_respond_with :success
      should_render_template :show
      should_not_set_the_flash

      should "assign the id to the shown instance" do
         assert_equal @section.id, assigns(:section).id
      end
    end
  end
```

```ruby
class SectionsControllerTest < ActionController::TestCase
  ...

  context "after logging in" do
    setup do
      @user = Factory(:user, :id => 12345)
      User.expects(:find_by_id).returns(@user)

      @section = Factory(:section, :id => 23456)
    end

    ...

    context "on GET to :edit" do
      setup do
        Section.expects(:find).with(@section.id.to_s).returns(@section)
        get :edit, :id => @section.id
      end

      should_assign_to :section
      should_respond_with :success
      should_render_template :edit
      should_not_set_the_flash

      should "assign the id to the shown instance" do
         assert_equal @section.id, assigns(:section).id
      end
    end
  end
end
```

```
Loaded suite /my_app/test/functional/
sections_controller_test
Started
....................
Finished in 0.217798 seconds.

19 tests, 45 assertions, 0 failures, 0 errors
```

Shoulda 2.0 Cheat Sheet

AR Associations

should_have_and_belong_to_many	:schools
should_have_many	:posts
should_have_one	:profile
should_belong_to	:user

AR Validations

should_ensure_length_at_least	:name, 3
should_ensure_length_in_range	:password, (6..20)
should_ensure_length_is	:ssn, 9
should_ensure_value_in_range	:age, (0..100)
should_require_unique_attributes	:ssn
should_require_acceptance_of	:terms_of_use
should_only_allow_numeric_values_for	:years_of_experience
should_require_attributes	:name, :age
should_allow_values_for*	:phone, "(123) 999-3049", "201-0023"

AR Class Attributes

should_have_class_methods	:find, :destroy
should_have_instance_methods	:name=, :email
should_have_named_scope	Any named_scope options
should_have_readonly_attributes	:password, :is_admin
should_protect_attributes	:password, :ssn
should_have_db_column*	:post_id
should_have_index*	:url

Controllers

should_route	:post, '/posts', :controller => :posts, :action => :create
should_change*	"Post.count", :from => 0, :to => 1
should_change*	"Post.count", :by => 1
should_assign_to*	:post, :equals => "@post"
should_assign_to*	:post, :class => Post
should_filter_params	:password, :ssn
should_render_with_layout*	"special"
should_respond_with_content_type	:rss
should_return_from_session	:user_id, "@user.id"
should_set_the_flash_to*	/created/i
should_redirect_to	"users_url(@user)"
should_render_template	:new | "new.html.erb"
should_render_a_form	Asserts that view contains a form
should_respond_with	:success | :redirect

Added Assertions

assert_same_elements	[1, 2, 3], [3, 2, 1]
assert_contains*	['a', '1'], /a/
assert_good_value	User, :email, "bob@email.com"
assert_bad_value	User, :birthdate, "Older than you!"
assert_save	User.new(params[:user])
assert_valid	Post.new(params[:post])

```
assert_sent_email do |email|
    email.subject =~ /activated/ && email.to.include?('bob@email.com')
end
```

*Indicates that a negative form of the assertion exists, e.g. **should_not_allow_values_for**, **assert_does_not_contain**, etc.

should_eventually(name, options = {}, &blk)

```
require 'test_helper'
require 'shoulda'

class UserTest < ActiveSupport::TestCase
  context "a user" do
    should_validate_presence_of :name
    should_succeed

    should_eventually "test that the user is a l337 h4xor";
  end
end


  * DEFERRED: a user should test that the user is a l337 h4xor.
Loaded suite my_app/test/unit/user_test
Started
..
Finished in 0.069118 seconds.

2 tests, 2 assertions, 0 failures, 0 errors
```

mocha

mocks

specify expectation

stubs

stand-in objects that return whatever you give them

install mocha as Plugin

```
$ script/plugin install git://github.com/floehopper/mocha
Initialized empty Git repository in /my_app/vendor/plugins/mocha/.git/
remote: Counting objects: 199, done.
remote: Compressing objects: 100% (189/189), done.
remote: Total 199 (delta 35), reused 86 (delta 8)
Receiving objects: 100% (199/199), 90.82 KiB, done.
Resolving deltas: 100% (35/35), done.
From git://github.com/floehopper/mocha
 * branch            HEAD       -> FETCH_HEAD
```

mock(name, &block) → mock object

mock(expected_methods = {}, &block) → mock object

mock(name, expected_methods = {}, &block) → mock object

```
def test_product
  product = mock('ipod_product', :manufacturer => 'ipod', :price => 100)
  assert_equal 'ipod', product.manufacturer
  assert_equal 100, product.price
  # an error will be raised unless both Product#manufacturer and
Product#price have been called
end
```

stub(name, &block) → mock object

stub(stubbed_methods = {}, &block) → mock object

stub(name, stubbed_methods = {}, &block) → mock object

```
def test_product
  product = stub('ipod_product', :manufacturer => 'ipod', :price => 100)
  assert_equal 'ipod', product.manufacturer
  assert_equal 100, product.price
  # an error will not be raised even if Product#manufacturer and
Product#price have not been called
end
```

stub_everything(name, &block) → mock object

stub_everything(stubbed_methods = {}, &block) → mock object

stub_everything(name, stubbed_methods = {}, &block) → mock object

```
def test_product
  product = stub_everything('ipod_product', :price => 100)
  assert_nil product.manufacturer
  assert_nil product.any_old_method
  assert_equal 100, product.price
end
```

expects(method_name) → expectation

```
object = mock()
object.expects(:method1)
object.method1
# no error raised

object = mock()
object.expects(:method1)
# error raised, because method1 not called exactly once
```

expects(method_names) → last expectation

```
object = mock()
object.expects(:method1 => :result1, :method2 => :result2)

# exactly equivalent to

object = mock()
object.expects(:method1).returns(:result1)
object.expects(:method2).returns(:result2)
```

stubs(method_name) → expectation

```
object = mock()
object.stubs(:method1)
object.method1
object.method1
# no error raised
```

stubs(method_names) → last expectation

```
object = mock()
object.stubs(:method1 => :result1, :method2 => :result2)

# exactly equivalent to

object = mock()
object.stubs(:method1).returns(:result1)
object.stubs(:method2).returns(:result2)
```

expects(symbol) → expectation

```
product = Product.new
product.expects(:save).returns(true)
assert_equal true, product.save
```

method_exists?(method, include_public_methods = true)

```
product = Product.new
assert_equal true,
product.method_exists?(:save)

assert_equal false,
product.method_exists?(:howdy)
```

stubs(symbol) → expectation

```
product = Product.new
product.stubs(:save).returns(true)
assert_equal true, product.save
```

```ruby
Mocha::Configuration.prevent(:stubbing_method_unnecessarily)

class ExampleTest < Test::Unit::TestCase
  def test_example
    example = mock('example')
    example.stubs(:unused_stub)
    # => Mocha::StubbingError: stubbing method unnecessarily:
    # =>   #<Mock:example>.unused_stub(any_parameters)
  end
end
```

at_least(minimum_number_of_times) → expectation

```
object = mock()
object.expects(:expected_method).at_least(2)
3.times { object.expected_method }
# => verify succeeds
```

```
object = mock()
object.expects(:expected_method).at_least(2)
object.expected_method
# => verify fails
```

at_least_once() → expectation

```
object = mock()
object.expects(:expected_method).at_least_once
object.expected_method
# => verify succeeds
```

```
object = mock()
object.expects(:expected_method).at_least_once
# => verify fails
```

at_most(maximum_number_of_times) → expectation

```
object = mock()
object.expects(:expected_method).at_most(2)
2.times { object.expected_method }
# => verify succeeds
```

```
object = mock()
object.expects(:expected_method).at_most(2)
3.times { object.expected_method }
# => verify fails
```

at_most_once() → expectation

```
object = mock()
object.expects(:expected_method).at_most_once
object.expected_method
# => verify succeeds

object = mock()
object.expects(:expected_method).at_most_once
2.times { object.expected_method }
# => verify fails
```

in_sequence(*sequences) → expectation

```
breakfast = sequence('breakfast')

egg = mock('egg')
egg.expects(:crack).in_sequence(breakfast)
egg.expects(:fry).in_sequence(breakfast)
egg.expects(:eat).in_sequence(breakfast)
```

multiple_yields(*parameter_groups) → expectation

```
object = mock()
object.expects(:expected_method). \
  multiple_yields(['result_1', 'result_2'], ['result_3'])
yielded_values = []
object.expected_method { |*values| yielded_values << values }
yielded_values # => [['result_1', 'result_2'], ['result_3]]
```

multiple_yields(*parameter_groups) → expectation

```
object = mock()
object.stubs(:expected_method). \
  multiple_yields([1, 2], [3]).then. \
  multiple_yields([4], [5, 6])
yielded_values_from_first_invocation = []
yielded_values_from_second_invocation = []
# first invocation
object.expected_method do |*values|
  yielded_values_from_first_invocation << values
end
# second invocation
object.expected_method do |*values|
  yielded_values_from_second_invocation << values
end
yielded_values_from_first_invocation # => [[1, 2],
[3]]
yielded_values_from_second_invocation # => [[4], [5,
6]]
```

never() → expectation

```
object = mock()
object.expects(:expected_method).never
object.expected_method
# => verify fails
```

```
object = mock()
object.expects(:expected_method).never
# => verify succeeds
```

once() → expectation

```
object = mock()
object.expects(:expected_method).once
object.expected_method
# => verify succeeds

object = mock()
object.expects(:expected_method).once
object.expected_method
object.expected_method
# => verify fails

object = mock()
object.expects(:expected_method).once
# => verify fails
```

raises(exception = RuntimeError, message = nil) → expectation

```
object = mock()
object.expects(:expected_method).raises(Exception, 'message')
object.expected_method # => raises exception of class Exception
and with message 'message'
```

```
object = mock()
object.stubs(:expected_method).raises(Exception1).then.raises(Exception2)
object.expected_method # => raises exception of class Exception1
object.expected_method # => raises exception of class Exception2
```

```
object = mock()
object.stubs(:expected_method).raises(Exception).then.returns(2, 3)
object.expected_method # => raises exception of class Exception1
object.expected_method # => 2
object.expected_method # => 3
```

returns(value) → expectation
returns(*values) → expectation

```
object = mock()
object.stubs(:stubbed_method).returns('result')
object.stubbed_method # => 'result'
object.stubbed_method # => 'result'
```

```
object = mock()
object.stubs(:stubbed_method).returns(1, 2)
object.stubbed_method # => 1
object.stubbed_method # => 2
```

```
object = mock()
object.stubs(:expected_method).returns(1, 2).then.returns(3)
object.expected_method # => 1
object.expected_method # => 2
object.expected_method # => 3
```

returns(value) → expectation
returns(*values) → expectation

```
object = mock()
object.stubs(:expected_method).returns(1, 2).then.raises(Exception)
object.expected_method # => 1
object.expected_method # => 2
object.expected_method # => raises exception of class Exception1
```

```
object = mock()
object.stubs(:expected_method).returns([1, 2])
x, y = object.expected_method
x # => 1
y # => 2
```

then() → expectation

```
object = mock()
object.stubs(:expected_method). \
  returns(1, 2).then. \
  raises(Exception).then.returns(4)
object.expected_method # => 1
object.expected_method # => 2
object.expected_method # => raises exception of class Exception
object.expected_method # => 4
```

then(state_machine.is(state)) → expectation

```
power = states('power').starts_as('off')

radio = mock('radio')
radio.expects(:switch_on).then(power.is('on'))
radio.expects(:select_channel).with('BBC Radio 4').when(power.is('on'))
radio.expects(:adjust_volume).with(+5).when(power.is('on'))
radio.expects(:select_channel).with('BBC World Service').when(power.is('on'))
radio.expects(:adjust_volume).with(-5).when(power.is('on'))
radio.expects(:switch_off).then(power.is('off'))
```

times(range) → expectation

```
object = mock()
object.expects(:expected_method).times(3)
3.times { object.expected_method }
# => verify succeeds

object = mock()
object.expects(:expected_method).times(3)
2.times { object.expected_method }
# => verify fails

object = mock()
object.expects(:expected_method).times(2..4)
3.times { object.expected_method }
# => verify succeeds

object = mock()
object.expects(:expected_method).times(2..4)
object.expected_method
# => verify fails
```

twice() → expectation

```
object = mock()
object.expects(:expected_method).twice
object.expected_method
object.expected_method
# => verify succeeds

object = mock()
object.expects(:expected_method).twice
object.expected_method
object.expected_method
object.expected_method
# => verify fails

object = mock()
object.expects(:expected_method).twice
object.expected_method
# => verify fails
```

when(state_machine.is(state)) → exception

```
power = states('power').starts_as('off')

radio = mock('radio')
radio.expects(:switch_on).then(power.is('on'))
radio.expects(:select_channel).with('BBC Radio 4').when(power.is('on'))
radio.expects(:adjust_volume).with(+5).when(power.is('on'))
radio.expects(:select_channel).with('BBC World Service').when(power.is('on'))
radio.expects(:adjust_volume).with(-5).when(power.is('on'))
radio.expects(:switch_off).then(power.is('off'))
```

with(*expected_parameters, &matching_block) → expectation

```
object = mock()
object.expects(:expected_method).with(:param1, :param2)
object.expected_method(:param1, :param2)
# => verify succeeds

object = mock()
object.expects(:expected_method).with(:param1, :param2)
object.expected_method(:param3)
# => verify fails


object = mock()
object.expects(:expected_method).with() { |value| value % 4 == 0 }
object.expected_method(16)
# => verify succeeds

object = mock()
object.expects(:expected_method).with() { |value| value % 4 == 0 }
object.expected_method(17)
# => verify fails
```

yields(*parameters) → expectation

```
object = mock()
object.expects(:expected_method).yields('result')
yielded_value = nil
object.expected_method { |value| yielded_value = value }
yielded_value # => 'result'
```

```
object = mock()
object.stubs(:expected_method).yields(1).then.yields(2)
yielded_values_from_first_invocation = []
yielded_values_from_second_invocation = []
# first invocation
object.expected_method do |value|
  yielded_values_from_first_invocation << value
end
# second invocation
object.expected_method do |value|
  yielded_values_from_second_invocation << value
end
yielded_values_from_first_invocation # => [1]
yielded_values_from_second_invocation # => [2]
```

Stubbing a non-existent method

```
Mocha::Configuration.prevent(:stubbing_non_existent_method)

class Example
end

class ExampleTest < Test::Unit::TestCase
  def test_example
    example = Example.new
    example.stubs(:method_that_doesnt_exist)
    # => Mocha::StubbingError: stubbing non-existent method:
    # =>   #<Example:0x593760>.method_that_doesnt_exist
  end
end
```

Stubbing a method unnecessarily

```
Mocha::Configuration.prevent(:stubbing_method_unnecessarily)

class ExampleTest < Test::Unit::TestCase
  def test_example
    example = mock('example')
    example.stubs(:unused_stub)
    # => Mocha::StubbingError: stubbing method unnecessarily:
    # =>   #<Mock:example>.unused_stub(any_parameters)
  end
end
```

Stubbing Method on Non-Mock Object

```
Mocha::Configuration.prevent(:stubbing_method_on_non_mock_object)

class Example
  def example_method; end
end

class ExampleTest < Test::Unit::TestCase
  def test_example
    example = Example.new
    example.stubs(:example_method)
    # => Mocha::StubbingError: stubbing method on non-mock object:
    # =>   #<Example:0x593620>.example_method
  end
end
```

Stubbing a non-public method

```
Mocha::Configuration.prevent(:stubbing_non_public_method)

class Example
  def internal_method; end
  private :internal_method
end

class ExampleTest < Test::Unit::TestCase
  def test_example
    example = Example.new
    example.stubs(:internal_method)
    # => Mocha::StubbingError: stubbing non-public method:
    # =>   #<Example:0x593530>.internal_method
  end
end
```

Gotchas

```
[WARNING] Using @user as the subject. Future versions of Shoulda
will require an explicit
subject using the subject class method.

Add this after your setup to avoid this warning: subject { @user }
```

The problem with using SomeObject.stubs is that it's almost the same as using SomeObject.expects, except if it's no longer necessary it doesn't cause a test to fail. This can lead to tests that unnecessarily stub methods as the application's implementation changes. And, the more methods that require stubbing the less the test can concisely convey intent.

~ Jay C. Field (http://blog.jayfields.com/2007/04/ruby-mocks-and-stubs-using-mocha.html)

```
Mocha::Configuration.prevent(:stubbing_non_existent_method)

class Example
end

class ExampleTest < Test::Unit::TestCase
  def test_example
    example = Example.new
    example.stubs(:method_that_doesnt_exist)
    # => Mocha::StubbingError: stubbing non-existent method:
    # =>   #<Example:0x593760>.method_that_doesnt_exist
  end
end
```

links

http://thoughtbot.com/projects/shoulda

http://dev.thoughtbot.com/shoulda/

http://rdoc.info/projects/thoughtbot/shoulda

http://mocha.rubyforge.org/

http://mocha.rubyforge.org/examples/misc.html

questions