Changing Your Mindset

Getting Started With Test-Driven Development

Patrick Reagan
Director, Application Development
patrick@viget.com

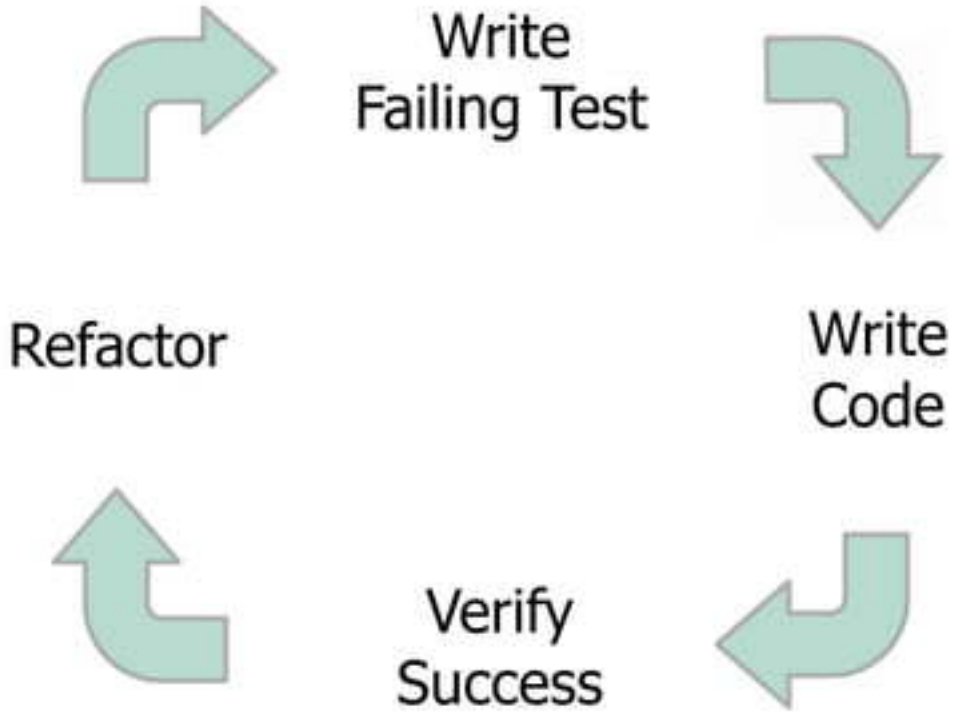


Overview

- Testing Process
- Test::Unit Introduction
- Code Example: GoogleRank
- Testing Pitfalls
- Coverage Analysis
- Next Steps

Test-Driven Development is the process of testing the behavior of non-existent objects

Testing Cycle



Why Test First?

- No untested code is written
- Start with higher code coverage
- Rapid feedback
- Build your regressions as you go

Anatomy of a (Test::Unit) Test Case

```ruby
require 'test/unit'

class StringTest < Test::Unit::TestCase

  def setup
    @string = 'ruby'
  end

  def test_gsub_should_replace_found_chars
    assert_equal 'rby', @string.gsub(/[aeiou]/i, '')
  end

  def test_match_with_matching_pattern_should_return_matchdata
    assert_instance_of MatchData, @string.match(/^ruby$/)
  end

end
```

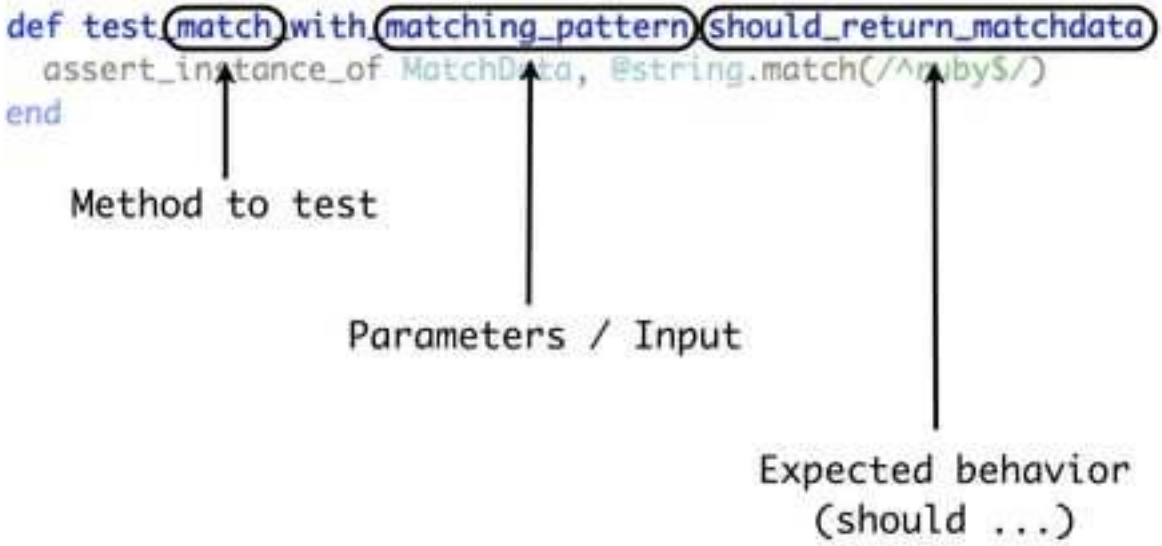
Test case

Test setup (run every test)

Test

Assertion

Naming Conventions



Basic Assertions

- assert

```
assert 'hello'.include?('hell')
```

- assert_equal / assert_not_equal

```
assert_equal 'rby', 'ruby'.gsub(/[aeiou]/i, '')
```

- assert_nil / assert_not_nil

```
assert_nil 'ruby'.match(/php/i)
```

- assert_match / assert_no_match

```
assert_match /^ruby/, 'ruby is awesome'
```

- assert_raise / assert_nothing_raised

```
assert_raise(IndexError) { 'hell'[4] = 'o' }
```

Code Example

Class that retrieves Google results

- **Retrieves only the top 10 URLs**
- URL encodes provided search terms
- Makes an HTTP connection to Google
- Raises an exception when unavailable

Start Small

```ruby
require 'test/unit'
require 'google_rank'

class GoogleRankTest < Test::Unit::TestCase
  def test_google_rank_should_exist
    assert_nothing_raised { GoogleRank }
  end
end
```

```
test_google_rank_should_exist(GoogleRankTest):
Exception raised: Class: <NameError>
Message: <"uninitialized constant GoogleRankTest::GoogleRank">

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

```ruby
class GoogleRank; end
```

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

Simplest Thing That Works

```
def test_parse_with_empty_document_should_return_empty_array
  assert_equal [], GoogleRank.parse('')
end
```

Class method

```
test_parse_with_empty_document_should_return_empty_array(GoogleRankTest):
NoMethodError: undefined method `parse' for GoogleRank:Class

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

```
class GoogleRank
  def parse(content)
    □
  end
end
```

Instance method

```
test_parse_with_empty_document_should_return_empty_array(GoogleRankTest):
NoMethodError: undefined method `parse' for GoogleRank:Class

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

Simplest Thing That Actually Works

```
class GoogleRank
  def self.parse(content)
    □
  end
end
```

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

Use a Canned Data Set

```ruby
def test_parse_with_sample_document_should_return_list_of_urls
  expected = [
    'http://blog.viget.com/',
    ...
    'http://www.techcfl.com/blog/?cat=2'
  ]

  assert_equal expected, GoogleRank.parse(File.read('data/google_search.html'))
end
```

Sample markup from Google search



```
test_parse_with_sample_document_should_return_list_of_urls
<["http://blog.viget.com/", ..... > expected but was <[]>.

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

Use a Canned Data Set

```
class GoogleRank
  def self.parse(content)
    urls = []
    results = content.scan(/<h2 class=r><a href="(http:\/\/\S+)"/)
    (1..10).each {|i| urls << results[i - 1][0] }
    urls
  end
end
```

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

Running Regressions

```
test_parse_with_empty_document_should_return_empty_array(GoogleRankTest):
NoMethodError: undefined method `[]' for nil:NilClass

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

```
class GoogleRank
  def self.parse(content)
    urls = []
    results = content.scan(/<h2 class=r><a href="(http:\/\/\S+)"/)
    (1..10).each {|i| urls << results[i - 1][0] }
    urls
  end
end
```

results.nil? == true

Fix Regressions

```
class GoogleRank
  def self.parse(content)
    urls = []
    results = content.scan(/<h2 class=r><a href="(http:\/\/\S+)"/)
    (1..10).each {|i| urls << results[i - 1][0] }
    urls
  end
end
```

```
class GoogleRank
  def self.parse(content)
    content.scan(/<h2 class=r><a href="(http:\/\/\S+)"/).map {|match| match[0] }
  end
end
```

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

Code Example

Class that retrieves Google results

- Retrieves only the top 10 URLs
- **URL encodes provided search terms**
- Makes an HTTP connection to Google
- Raises an exception when unavailable

Retrieve Internal State

```
include ERB::Util

def test_new_should_encode_supplied_keywords
  keywords = 'rails development blog'
  gr = GoogleRank.new(keywords)
  assert_equal url_encode(keywords), gr.instance_variable_get(:@encoded_keywords)
end
```



Inspect instance variable

```
NameError: uninitialized constant GoogleRankTest::ERB
```

Retrieve Internal State

```
require 'erb'

class GoogleRank
  def self.parse(content)
    content.scan(/<h2 class=r><a href="(http:\/\/\S+)"/).map {|match| match[0] }
  end
end
```

```
test_new_should_encode_supplied_keywords(GoogleRankTest):
ArgumentError: wrong number of arguments (1 for 0)

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

Retrieve Internal State

```
class GoogleRank

  include ERB::Util

  def initialize(keywords)
    @encoded_keywords = url_encode(keywords)
  end

  def self.parse(content)
    content.scan(/<h2 class=r><a href="(http:\/\/\S+)"/).map {|match| match[0] }
  end
end
```

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

```
4 tests, 3 assertions, 0 failures, 0 errors
```

Refactor Test Duplication

```ruby
def test_new_should_encode_supplied_keywords
  keywords = 'rails development blog'
  gr = GoogleRank.new(keywords)
  assert_equal url_encode(keywords), gr.instance_variable_get(:@encoded_keywords)
end

def setup
  @keywords = 'rails development blog'
  @gr = GoogleRank.new(@keywords)
end

def test_new_should_encode_supplied_keywords
  assert_equal url_encode(@keywords), @gr.instance_variable_get(:@encoded_keywords)
end
```

```
4 tests, 3 assertions, 0 failures, 0 errors
```

Testing Encoding

```
def test_new_should_assign_url_with_base_and_encoded_keywords
  url = GoogleRank::BASE_URL + url_encode(@keywords)
  assert_equal url, @gr.instance_variable_get(:@url)
end
```

```
test_new_should_assign_url_with_base_and_encoded_keywords(GoogleRankTest):
NameError: uninitialized constant GoogleRank::BASE_URL

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

Testing Encoding

```
class GoogleRank

  BASE_URL = 'http://www.google.com/search?q='

  include ERB::Util

  def initialize(keywords)
    @encoded_keywords = url_encode(keywords)
    @url = "#{BASE_URL}#{@encoded_keywords}"
  end

  def self.parse(content)
    content.scan(/<h2 class=r><a href="(http:\/\/\S+)"/).map {|match| match[0] }
  end
end
```

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

```
5 tests, 4 assertions, 0 failures, 0 errors
```

Code Example

Class that retrieves Google results

- Retrieves only the top 10 URLs
- URL encodes provided search terms
- **Makes an HTTP connection to Google**
- Raises an exception when unavailable

"Mock objects are simulated objects that mimic the behavior of real objects in controlled ways"

- Wikipedia

Mocks and Stubs

- Remove external dependencies
- Create known state
- Focus tests on specific code paths

Mocha to Stub Net::HTTP

```ruby
require 'rubygems'
require 'mocha'

class GoogleRankTest < Test::Unit::TestCase

  ...

  def test_retrieve_content_should_set_content
    content = google_content()
    Net::HTTP.stubs(:get).returns(content)
    @gr.retrieve_content
    assert_equal content, @gr.instance_variable_get(:@content)
  end

end
```

Implementation will use Net::HTTP#get

```
test_retrieve_content_should_set_content(GoogleRankTest):
NoMethodError: undefined method `retrieve_content' for #<GoogleRank:0x104053c>

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

Mocha to Stub Net::HTTP

```
class GoogleRank

  BASE_URL = 'http://www.google.com/search?q='

  include ERB::Util

  def initialize(keywords)
    @encoded_keywords = url_encode(keywords)
    @url = "#{BASE_URL}#{@encoded_keywords}"
  end

  def self.parse(content)
    content.scan(/<h2 class=r><a href="(http:\/\/\S+)"/).map {|match| match[0] }
  end

  def retrieve_content
    @content = Net::HTTP.get(URI.parse(@url))
  end
end
```

```
6 tests, 5 assertions, 0 failures, 0 errors
```

No Network? Oops!

```
$ irb --prompt simple
>> gr = GoogleRank.new('rails')
=> #<GoogleRank:0x54c89c @encoded_keywords="rails",
     @url="http://www.google.com/search?q=rails">
>> gr.retrieve_content
SocketError: getaddrinfo: No address associated with nodename
```

Replicate Real-Word Conditions

```
def test_retrieve_content_when_connection_fails_should_set_content_to_nil
  Net::HTTP.stubs(:get).raises(SocketError)
  @gr.retrieve_content
  assert_nil @gr.instance_variable_get(:@content)
end
```

Expected failure

```
test_retrieve_content_when_connection_fails_should_set_content_to_nil(GoogleRankTest):
SocketError: SocketError

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

Dealing With Failure

```
class GoogleRank

  ...

  def retrieve_content
    begin
      @content = Net::HTTP.get(URI.parse(@url))
    rescue SocketError
      @content = nil
    end
  end
end
```

```
7 tests, 6 assertions, 0 failures, 0 errors
```

Code Example

Class that retrieves Google results

- Retrieves only the top 10 URLs
- URL encodes provided search terms
- Makes an HTTP connection to Google
- **Raises an exception when unavailable**

Dealing With Even More Failure

```
def test_retrieve_content_when_connection_fails_should_raise_exception_and_set_content_to_nil
  Net::HTTP.stubs(:get_response).raises(SocketError)
  assert_raise(GoogleRank::ConnectionError) { @gr.retrieve_content }
  assert_nil @gr.instance_variable_get(:@content)
end

def test_retrieve_content_with_failure_should_raise_exception
  response = Net::HTTPInternalServerError.new('1.2', '500', 'Internal Server Error')
  Net::HTTP.stubs(:get_response).returns(response)
  assert_raise(GoogleRank::ConnectionError) { @gr.retrieve_content }
end
```

```
test_retrieve_content_when_connection_fails_should_raise_exception_and_set_content_to_nil(GoogleRankTest)
<GoogleRank::ConnectionError> exception expected but none was thrown.

test_retrieve_content_with_failure_should_raise_exception(GoogleRankTest)
<GoogleRank::ConnectionError> exception expected but was
Class: <NoMethodError> Message: <"undefined method `closed?' for nil:NilClass">

7 tests, 6 assertions, 2 failures, 0 errors
```

Handling Unsuccessful Requests

```
require 'erb'
require 'net/http'

class GoogleRank

  class ConnectionError < RuntimeError; end

  ...

  def retrieve_content
    begin
      response = Net::HTTP.get_response(URI.parse(@url))
      raise ConnectionError unless response.is_a?(Net::HTTPSuccess)
      @content = response.body
    rescue SocketError, ConnectionError
      @content = nil
      raise ConnectionError
    end
  end
end
```

```
8 tests, 8 assertions, 0 failures, 0 errors
```

Testing Pitfalls

Things to avoid

- Over-mocking
- Invalid mocks
- Testing library code
- "Assertion-heavy" tests
- Non-descriptive test names

Testing Pitfalls

```ruby
# Over-mocking
def test_parse_with_sample_document_should_return_list_of_urls
  content = google_content()
  content.stubs(:scan).returns([['http://blog.viget.com/']])
  gr = GoogleRank.new(@keywords)
  assert_equal ['http://blog.viget.com/'], gr.parse(content)
end

# Invalid mock - no 'content' method on HTTPResponse object
def test_retrieve_content_with_success_should_return_retrieved_content
  response = Net::HTTPSuccess.new('1.2', '200', 'OK')
  response.stubs(:content).returns(@content)
  Net::HTTP.stubs(:get_response).returns(response)

  gr = GoogleRank.new(@keywords)
  assert_equal @content, gr.retrieve_content
end
```

Should test parsing

HTTPResponse#content does not exist

Testing Pitfalls

```ruby
# Testing library code (e.g. attr_accessor :content)
def test_content_accessor_exists
  gr = GoogleRank.new(@keywords)
  gr.content = 'foo'
  assert_equal 'foo', gr.content
end

# Non-descriptive, assertion-heavy test
def test_new
  keywords = 'rails development blog'
  encoded_keywords = url_encode(keywords)
  url = GoogleRank::BASE_URL + encoded_keywords

  gr = GoogleRank.new(keywords)
  assert_equal encoded_keywords, gr.instance_variable_get(:@encoded_keywords)
  assert_equal url, gr.instance_variable_get(:@url)
end
```

Use secondary test to verify

Expected behavior?

Coverage Analysis

- Shows 'tested' code
- Indication of when to stop
- Can present a false impression

Coverage With RCov

Name	Total lines	Lines of code	Total coverage	Code coverage
google_rank.rb	22	16	100.0%	100.0%

```
 1 require 'erb'
 2 require 'net/http'
 3
 4 class GoogleRank
 5
 6   BASE_URL = 'http://www.google.com/search?q='
 7
 8   include ERB::Util
 9
10   def initialize(keywords)
11     @encoded_keywords = url_encode(keywords)
12     @url = "#{BASE_URL}#{@encoded_keywords}"
13   end
14
15   def self.parse(content)
16     content.scan(/<h2 class=r><a href="(http:\/\/\S+)"/).map {|match| match[0] }
17   end
18
19   def retrieve_content
20     @content = Net::HTTP.get(URI.parse(@url))
21   end
22 end
```

Exception-prone

Next Steps?

- Rake - Rake::TestTask
- Mock expectations as tests
- Testing Macros

Rake::TestTask Example

```
require 'rake'
require 'rake/testtask'

task :default => 'test:all'

namespace :test do
  Rake::TestTask.new(:all) do |t|
    t.libs << 'test'
    t.test_files = FileList['*_test.rb']
    t.verbose = false
  end
end
```

```
$ rake -T
rake test:all  # Run tests for all
```

Expectations as Tests

```ruby
require 'test/unit'
require 'rubygems'
require 'mocha'

class GoogleRankExpectationTest < Test::Unit::TestCase
  class GoogleRankExpectationTest::GoogleRank; end

  def load_class
    class_source = File.join(File.dirname(__FILE__), 'google_rank.rb')
    yield
    self.class.class_eval(File.read(class_source))
  end

  def test_class_should_call_attr_accessor
    load_class { GoogleRank.expects(:attr_accessor).with(:content) }
  end
end
```

attr_accessor :content called once

Testing Macros

```
class PollControllerTest < Test::Unit::TestCase
  should_get(:index).and_assign(:poll) { |var| var.is_a? Poll }.and_render(:index)
  should_get(:woo).and_render.and_set_flash(:woo, 'woo')
  should_post(:woo).and_redirect_to(:answers_url).and_set_flash(:notice)
end

class AnswerTest < Test::Unit::TestCase
  should_require_numeric :participant_id, :choice_id
  should_require :participant_id, :choice_id
  should_belong_to :participant, :choice
end
```

Resources

Test::Unit

- http://ruby-doc.org/stdlib

Mocha

- http://mocha.rubyforge.org

RCov

- http://eigenclass.org/hiki/rcov

Rake

- http://rake.rubyforge.org

How to Test Validations (Expectations as Tests)

- http://relevancellc.com/2007/7/16/how-to-test-validations-part-4

Testing Macros

- http://www.extendviget.com

Slides

http://www.slideshare.net/viget

Blog

http://blog.viget.com

http://www.sneaq.net

Contact

patrick@viget.com