

Introduction to Testing

Konstantinos Karasavvas

CITY College

December 1, 2013

Table of contents

- 1 Testing Software
- 2 Why Testing?
- 3 Unit Testing
- 4 TDD/BDD
- 5 RSpec
- 6 Rack Testing
- 7 Selenium

Testing Methods

- Static testing
 - reviews or walkthroughs
- Dynamic testing
 - automatic code testing
- White box testing
 - internal workings of software code
- Black box testing
 - functionality of software code

Testing Levels

- Unit testing
 - test individual units of code (and composed units)
 - procedural: typically functions
 - Object-oriented: typically classes
 - white box testing
- Integration testing
 - tests subsystem communications through their interfaces
 - black box testing from the subsystem's perspective
 - white box testing from the system's perspective
 - can used to verify functional, performance, and reliability requirements
- System testing
 - tests complete system
 - black box testing
 - can used to verify functional, performance, and reliability requirements

Why Testing?

- Better code quality
 - the code could have bugs
 - the tests could have bugs
 - but chances for both to have bugs is low
- Increased confidence
 - via regression testing
 - Tests ensure that everything is tested every time
- Allows quicker/bigger changes to code
 - If the old tests work then no need to check if something broke
 - depends on testing coverage
- Good unit tests help you define and document the intended functionality better

Unit Testing, Example

```
1 # File: simple_number.rb
2
3 class SimpleNumber
4
5   def initialize(num)
6     raise unless num.is_a?(Numeric)
7     @x = num
8   end
9
10  def add(y)
11    @x + y
12  end
13
14  def multiply(y)
15    @x * y
16  end
17
18 end
```

Unit Testing, Example, cont.

```
1 # File: tc_simple_number.rb
2
3 require_relative "simple_number"
4 require "test/unit"
5
6 class TestSimpleNumber < Test::Unit::TestCase
7
8   def test_simple
9     assert_equal(4, SimpleNumber.new(2).add(2) )
10    assert_equal(6, SimpleNumber.new(2).multiply(3) )
11  end
12
13 end
```

Unit Testing, Example, cont.

```
1 $ ruby tc_simple_number.rb
2 Loaded suite tc_simple_number
3 Started
4 .
5 Finished in 0.002695 seconds.
6
7 1 tests, 2 assertions, 0 failures, 0 errors
```


Unit Testing, Example, cont.

```
1 # File: tc_simple_number2.rb
2
3 require_relative "simple_number"
4 require "test/unit"
5
6 class TestSimpleNumber < Test::Unit::TestCase
7
8   def test_simple
9     assert_equal(4, SimpleNumber.new(2).add(2) )
10    assert_equal(4, SimpleNumber.new(2).multiply(2) )
11  end
12
13  def test_typecheck
14    assert_raise( RuntimeError ) { SimpleNumber.new('a') }
15  end
16
17  def test_failure
18    assert_equal(3, SimpleNumber.new(2).add(2), "Adding doesn't ←
19      work" )
20  end
21 end
```

Unit Testing, Example, cont.

```
1 $ ruby tc_simple_number2.rb
2 Loaded suite tc_simple_number2
3 Started
4 ...
5 Finished in 0.038617 seconds.
6
7 1) Failure:
8 test_failure(TestSimpleNumber) [tc_simple_number2.rb:16]:
9 Adding doesn't work.
10 <3> expected but was
11 <4>.
12
13 3 tests, 4 assertions, 1 failures, 0 errors
```

Unit Testing, Example, cont.

```
1 # File: tc_simple_number3.rb
2
3 require "../simple_number"
4 require "test/unit"
5
6 class TestSimpleNumber < Test::Unit::TestCase
7
8   def setup
9     @num = SimpleNumber.new(2)
10   end
11
12   def teardown
13   end
14
15   def test_simple
16     assert_equal(4, @num.add(2) )
17   end
18
19   def test_simple2
20     assert_equal(4, @num.multiply(2) )
21   end
22
23 end
```

Test-driven development (TDD)

- Software development process
 - first the developer writes the automated test – will fail
 - then he/she writes the minimum amount of code to pass the test
 - finally he/she refactors the new code to acceptable standards
- Forces developers to think about their code before writing it
- Must think of the ways his code (objects) interact with other code (objects/mock objects)
- Forces developers to think about interactions and interfaces before writing code

Mock Objects

- Used to *simulate* object's behaviour for testing code which
 - depends on an external resource (WS, filesystem, DB, mail server, ...)
 - would involve a large amount of non-reusable setup and fixture data
 - relies on features which are particularly computationally expensive
 - depends on unwritten objects

Specifications and BDD

- Human readable specifications that direct and validate code
 - Unit testing at a more abstract level
 - Tests behaviour
- Behaviour Driven Development combines aspects of
 - Acceptance Test Driven Planning
 - Domain Driven Design
 - Test Driven Development
- Ruby's RSpec

Domain Specific Language for describing the expected behaviour of a system with executable examples.

- instead of assertions it uses describers
- ...you describe a class, method, etc.
- ...and then you state expectations

RSpec

```
1 $ gem install rspec
```

RSpec

```
1 $ gem install rspec
```

- rspec_example
 - spec
 - lib

RSpec

```
1 $ gem install rspec
```

- rspec_example

- spec
- lib

```
1 # add_spec.rb
2 describe "add function" do
3   it "adds two numbers" do
4     add(2,3).should == 5
5   end
6 end
```

RSpec

```
1 $ gem install rspec
```

- rspec_example

- spec
- lib

```
1 # add_spec.rb
2 describe "add function" do
3   it "adds two numbers" do
4     add(2,3).should == 5
5   end
6 end
```

```
1 $ rspec
```

RSpec, cont.

```
1 Failures:
2
3 1) add function adds two numbers
4     Failure/Error: add(2,3).should == 5
5     NoMethodError:
6       undefined method 'add' for #<RSpec::Core::ExampleGroup::↵
7         Nested_1:0x0000000193fe30>
8       # ./spec/add_spec.rb:5:in 'block (2 levels) in <top (↵
9         required)>'
10
11 Finished in 0.00077 seconds
12 1 example, 1 failure
```

RSpec, cont.

```
1 # add.rb
2 def add a, b
3   a + b
4 end
```

RSpec, cont.

```
1 # add.rb
2 def add a, b
3   a + b
4 end
```

```
1 # add_spec.rb
2 require_relative "../lib/add.rb"
3
4 describe "add function" do
5   it "adds two numbers" do
6     add(2,3).should == 5
7   end
8 end
```

RSpec, cont.

```
1 # add.rb
2 def add a, b
3   a + b
4 end
```

```
1 # add_spec.rb
2 require_relative "../lib/add.rb"
3
4 describe "add function" do
5   it "adds two numbers" do
6     add(2,3).should == 5
7   end
8 end
```

```
1 $ rspec
2 .
3
4 Finished in 0.00076 seconds
5 1 example, 0 failures
```

Rack Testing

```
1 $ gem install rack-test
```

Rack Testing

```
1 $ gem install rack-test
```

```
1 # my_app.rb
2
3 class MyApp < Sinatra::Application
4
5   get "/" do
6     "Hello World!"
7   end
8
9 end
```


Rack Testing

```
1 $ gem install rack-test
```

```
1 # my_app.rb
2
3 class MyApp < Sinatra::Application
4
5   get "/" do
6     "Hello World!"
7   end
8
9 end
```

```
1 require "sinatra"
2 require "./my_app.rb"
3
4 run MyApp.new!
```

Rack Testing, cont.

```
1 require "sinatra"
2 require "rack/test"
3
4 require_relative "../my_app.rb"
5
6 set :environment, :test
7
8 describe MyApp do
9   include Rack::Test::Methods
10   def app
11     MyApp
12   end
13
14   describe "First simple test with Sinatra" do
15     it "says hello world" do
16       get "/"
17
18       last_response.status.should == 200
19       last_response.body.should =~ /Hello World/
20     end
21   end
22 end
```

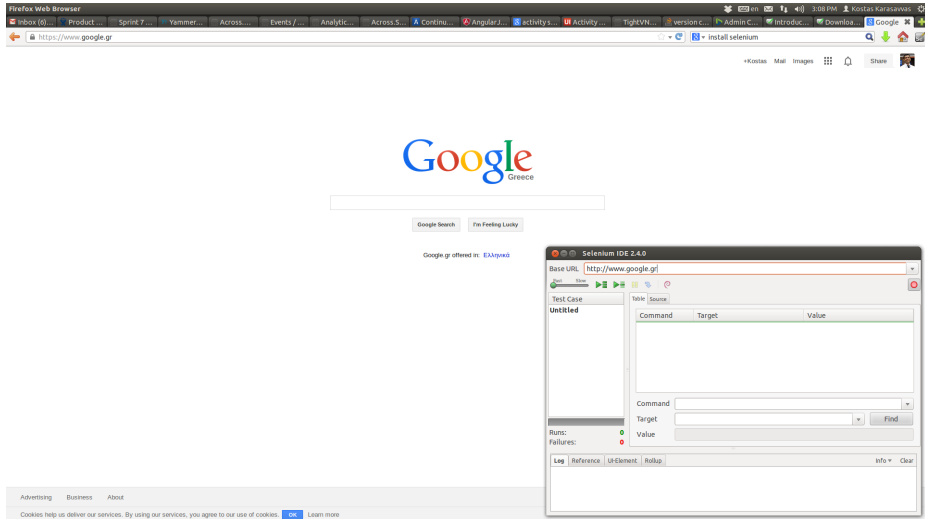
Rack Testing, cont.

```
1 $ rspec
2 .
3
4 Finished in 0.06143 seconds
5 1 example, 0 failures
```

Functional Testing with Selenium

- Selenium is a set of different tools to enable test automation
 - Selenium IDE
 - Selenium Server
 - ...
- Selenium IDE
 - Plugin for Firefox
 - Captures user actions and allows assertions on them
 - `assertXXX`; e.g. `assertText`
 - `verifyXXX`; e.g. `verifyText`
 - `verify` allows test to continue, `assert` does not
- Need to install Selenium IDE plugin to Firefox

Selenium IDE Example



Selenium IDE Example, cont.

functional testing - Google Search - Mozilla Firefox

https://www.google.gr/#q=functional-testing

functional testing

About 38,700,000 results (0.25 seconds)

Cookies help us deliver our services. By using our services, you agree to our use of cookies.

Functional testing - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Functional_testing
Functional testing is a type of software testing that bases its test cases on the specification of the intended functions of the software component.

System testing - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/System_testing
System testing is performed on the complete system to verify that it meets the requirements specified in the System Requirements Specification (SRS) and the Functional Requirements Specification (FRS).

Unit, Integration, and Functional Testing
docs.pythonsprint.org/projects/...
A functional test is a form of integration test that verifies that the system meets the functional requirements.

Functional Tests - Codeception - Documentation
codeception.com/docs/00-FunctionalTests
Now that we've written some acceptance tests, functional tests are almost the same, with just one major difference: Functional tests don't require a web server to ...

Functional Testing | Functional Testing - SoapUI
www.soapui.org/Functional-Testing/functional-testing.html
In this easy-to-follow tutorial you'll learn the basics of functionally testing your API using SoapUI, the most downloaded testware in the world.

testing - Unit tests vs Functional tests - Stack Overflow
stackoverflow.com/questions/2741832/
Apr 30, 2010 - What is the difference between unit tests and functional tests? ... Unit Test - testing an individual unit, such as a method (function) in a class, with ...

Functional Testing | HP's Official Site - Hewlett Packard
www.hp.com/us/en/software-solutions/functional-testing.html
Achieve modernization with Functional Testing from HP in order to rely less on manual testing and receive more successful automation and effectiveness of the ...

Functional Testing for Web, Mobile and Desktop Applications | uTest
www.utest.com/functional-testing
uTest provides a wide range of functional testing services.

open /#q=functional-testing
assertTitle functional testing - Google Search
assertValue
assertText css=span.st Functional testing is a quality assurance (QA) process and a ...
assertTable
assertElementPresent css=span.st
verifyTitle functional testing - Google Search
verifyValue
verifyText css=span.st Functional testing is a quality assurance (QA) process and a t...
verifyTable
verifyElementPresent css=span.st
waitForTitle functional testing - Google Search
waitForValue
waitForText css=span.st Functional testing is a quality assurance (QA) process and ...
waitForTable
waitForElementPresent css=span.st
storeTitle functional testing - Google Search
storeValue
storeText css=span.st Functional testing is a quality assurance (QA) process and a T...
storeTable
storeElementPresent css=span.st

Selenium IDE Example, cont.

The screenshot displays the Selenium IDE 2.4.0 interface. The top bar shows the Base URL as `https://www.google.gr/`. Below this is a toolbar with icons for Fast, Slow, Play, Stop, and other controls. The main area is divided into two tabs: 'Table' and 'Source'. The 'Table' tab is active, showing a table of test steps. The table has three columns: Command, Target, and Value. The steps are as follows:

Command	Target	Value
open	/	
click	id=gbqfq	
type	id=gbqfq	functional testing
click	id=gbqfb	
assertText	css=span.st	Functional testing is a quality assurance (QA) process and a type ...

Below the table, there are input fields for Command, Target, and Value, along with a 'Find' button. The bottom of the interface features a 'Log' tab and a 'Reference' tab, with a 'Rollup' button and an 'Info' dropdown menu.

Selenium IDE Example, cont.

The screenshot displays the Selenium IDE 2.4.0 interface overlaid on a Firefox Web Browser. The browser shows a Google search for "functional testing". The Selenium IDE interface includes a Base URL field set to "https://www.google.gr/", a Test Case dropdown showing "Google Test 1", and a "Play current test case" button. Below these, a table lists the test steps:

Command	Target	Value
open	/	
click	id=gbqfq	
type	id=gbqfq	functional testing
click	id=gbqfb	
assertText	css=span.st	Functional testing is a quality assurance (QA) process and a type ...

At the bottom of the IDE, the "Log" tab shows the execution progress:

```

[info] Executing: open [/]
[info] Executing: click [id=gbqfq]
[info] Executing: type [id=gbqfq] functional testing
[info] Executing: click [id=gbqfb]
[info] Executing: assertText [css=span.st] Functional testing is a quality assurance (QA) process and a type of black box testing that bases its test cases on the specifications of the software component...
  
```

The status bar indicates 1 Run and 0 Failures.

Selenium IDE Example, cont.

The screenshot shows the Selenium IDE 2.4.0 interface. The 'Export Test Case As...' menu is open, displaying various export options. The 'Ruby / RSpec / WebDriver' option is highlighted. The background shows a test case with steps like 'open', 'click', 'type', and 'click', and a table with command and target details.

Export Test Case As... Menu Options:

- Ruby / RSpec / WebDriver
- Ruby / Test::Unit / WebDriver
- Ruby / RSpec / Remote Control
- Ruby / Test::Unit / Remote Control
- Python 2 / unittest / WebDriver
- Python 2 / unittest / Remote Control
- Java / JUnit 4 / WebDriver
- Java / JUnit 4 / WebDriver Backed
- Java / JUnit 4 / Remote Control
- Java / JUnit 3 / Remote Control
- Java / TestNG / Remote Control
- C# / NUnit / WebDriver
- C# / NUnit / Remote Control

Test Case Steps:

Command	Target
open	/
click	id=gbqfq
type	id=gbqfq
click	id=gbqfb
assertText	css=span.st

Selenium and RSpecs

```

1 require "json"
2 require "selenium-webdriver"
3 require "rspec"
4 include RSpec::Expectations
5
6 describe "GoogleTest1" do
7
8   before(:each) do
9     @driver = Selenium::WebDriver.for :firefox
10    @base_url = "https://www.google.gr/"
11    @accept_next_alert = true
12    @driver.manage.timeouts.implicit_wait = 30
13    @verification_errors = []
14  end
15
16  after(:each) do
17    @driver.quit
18    @verification_errors.should == []
19  end
20
21  it "test_google_test1" do
22    @driver.get(@base_url + "/")
23    @driver.find_element(:id, "gbqfq").click
24    @driver.find_element(:id, "gbqfq").clear
25    @driver.find_element(:id, "gbqfq").send_keys "functional testing"
26    @driver.find_element(:id, "gbqfb").click
27    (@driver.find_element(:css, "span.st").text).should == "Functional testing is a ←
    quality assurance (QA) process and a type of black box testing that bases its ←
    test cases on the specifications of the software component ..."
28  end
29

```

Selenium and RSpecs

```
1 $ gem install rspec selenium-webdriver
```

```
1 $ rspec spec/google_selenium_rspec.rb
2 .
3
4 Finished in 16.47 seconds
5 1 example, 0 failures
```

- default browser will open and test *driven* visually