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

```ruby
# File:  simple_number.rb

class SimpleNumber

  def initialize(num)
    raise unless num.is_a?(Numeric)
    @x = num
  end

  def add(y)
    @x + y
  end

  def multiply(y)
    @x * y
  end

end
```

Unit Testing, Example, cont.

```ruby
# File:  tc_simple_number.rb

require_relative "simple_number"
require "test/unit"

class TestSimpleNumber < Test::Unit::TestCase

  def test_simple
    assert_equal(4, SimpleNumber.new(2).add(2) )
    assert_equal(6, SimpleNumber.new(2).multiply(3) )
  end

end
```

Unit Testing, Example, cont.

```
$ ruby tc_simple_number.rb
Loaded suite tc_simple_number
Started
.
Finished in 0.002695 seconds.

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

Unit Testing, Example, cont.

```ruby
# File:  tc_simple_number2.rb

require_relative "simple_number"
require "test/unit"

class TestSimpleNumber < Test::Unit::TestCase

  def test_simple
    assert_equal(4, SimpleNumber.new(2).add(2) )
    assert_equal(4, SimpleNumber.new(2).multiply(2) )
  end

  def test_typecheck
    assert_raise( RuntimeError ) { SimpleNumber.new('a') }
  end

  def test_failure
    assert_equal(3, SimpleNumber.new(2).add(2), "Adding doesn't work" )
  end

end
```

Unit Testing, Example, cont.

```
$ ruby tc_simple_number2.rb
Loaded suite tc_simple_number2
Started
...
Finished in 0.038617 seconds.

  1) Failure:
test_failure(TestSimpleNumber) [tc_simple_number2.rb:16]:
Adding doesn't work.
<3> expected but was
<4>.

3 tests, 4 assertions, 1 failures, 0 errors
```

Unit Testing, Example, cont.

```ruby
# File:   tc_simple_number3.rb

require "./simple_number"
require "test/unit"

class TestSimpleNumber < Test::Unit::TestCase

  def setup
    @num = SimpleNumber.new(2)
  end

  def teardown
  end

  def test_simple
    assert_equal(4, @num.add(2) )
  end

  def test_simple2
    assert_equal(4, @num.multiply(2) )
  end

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

```
$ gem install rspec
```

- rspec_example
  - spec
  - lib

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

RSpec

```
$ gem install rspec
```

- rspec_example
  - spec
  - lib

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

```
$ 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::↩
              Nested_1:0x0000000193fe30>
7       # ./spec/add_spec.rb:5:in `block (2 levels) in <top (↩
            required)>'
8
9  Finished in 0.00077 seconds
10 1 example, 1 failure
```

RSpec, cont.

```ruby
# add.rb
def add a, b
  a + b
end
```

RSpec, cont.

```ruby
# add.rb
def add a, b
  a + b
end
```

```ruby
# add_spec.rb
require_relative "../lib/add.rb"

describe "add function" do
  it "adds two numbers" do
    add(2,3).should == 5
  end
end
```

RSpec, cont.

```ruby
# add.rb
def add a, b
  a + b
end
```

```ruby
# add_spec.rb
require_relative "../lib/add.rb"

describe "add function" do
  it "adds two numbers" do
    add(2,3).should == 5
  end
end
```

```
$ rspec
.

Finished in 0.00076 seconds
1 example, 0 failures
```

Rack Testing

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

Rack Testing

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

```ruby
# my_app.rb

class MyApp < Sinatra::Application

  get "/" do
    "Hello World!"
  end

end
```

Rack Testing

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

```ruby
# my_app.rb

class MyApp < Sinatra::Application

  get "/" do
    "Hello World!"
  end

end
```

```ruby
require "sinatra"
require "./my_app.rb"

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.

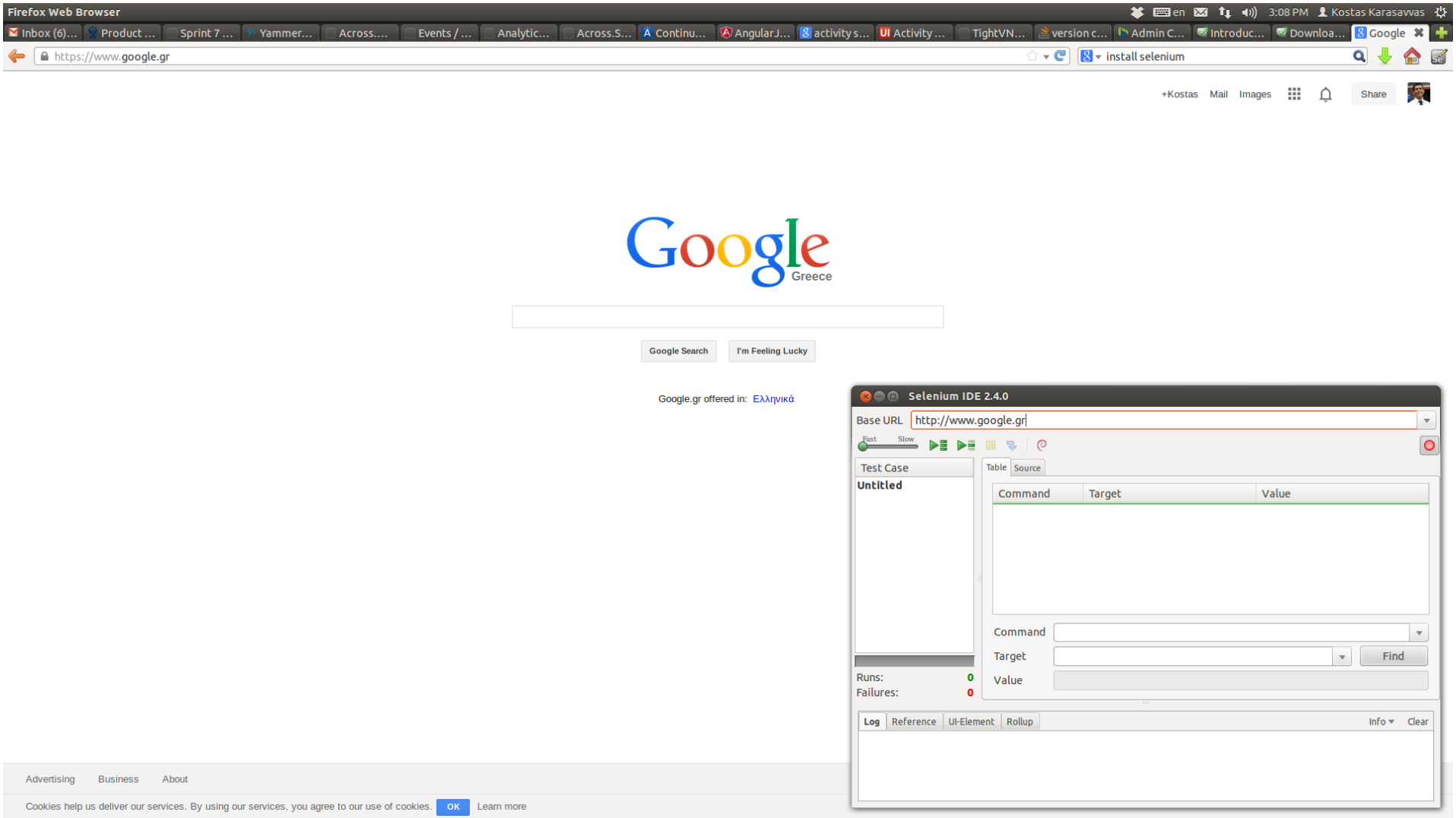
```
$ rspec
.

Finished in 0.06143 seconds
1 example, 0 failures
```

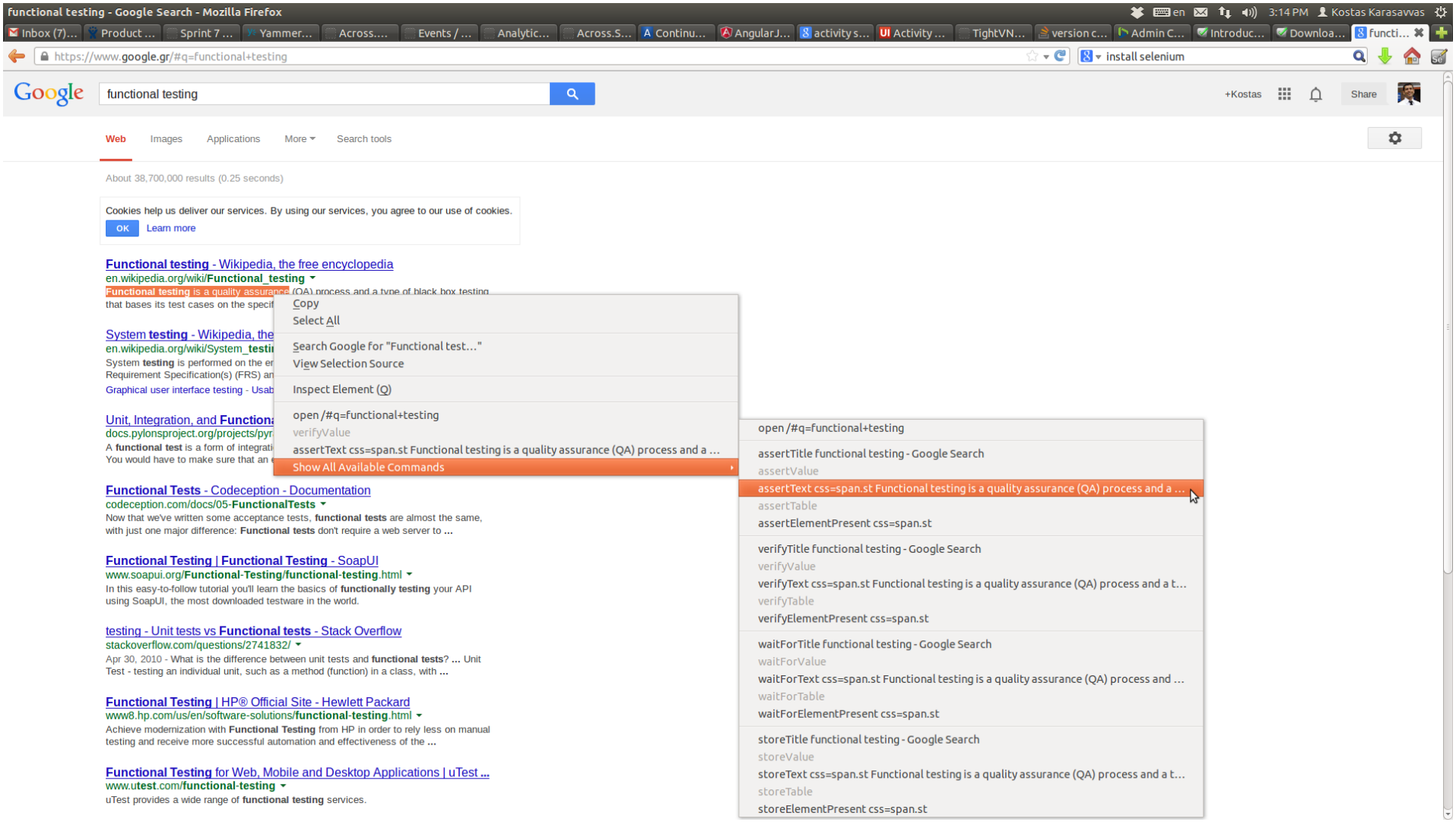
Functional Testing with Selenium

- 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

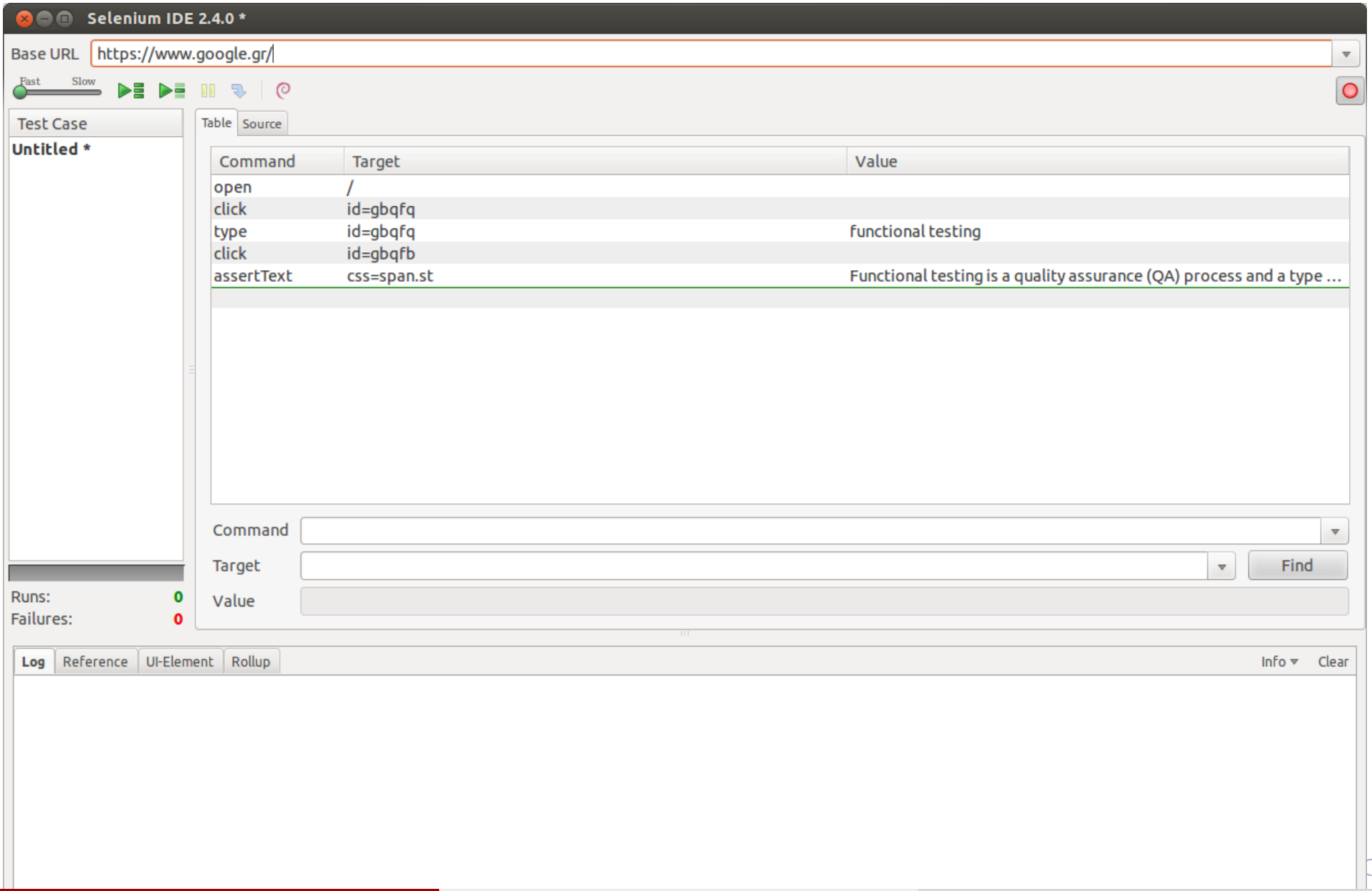
Selenium IDE Example



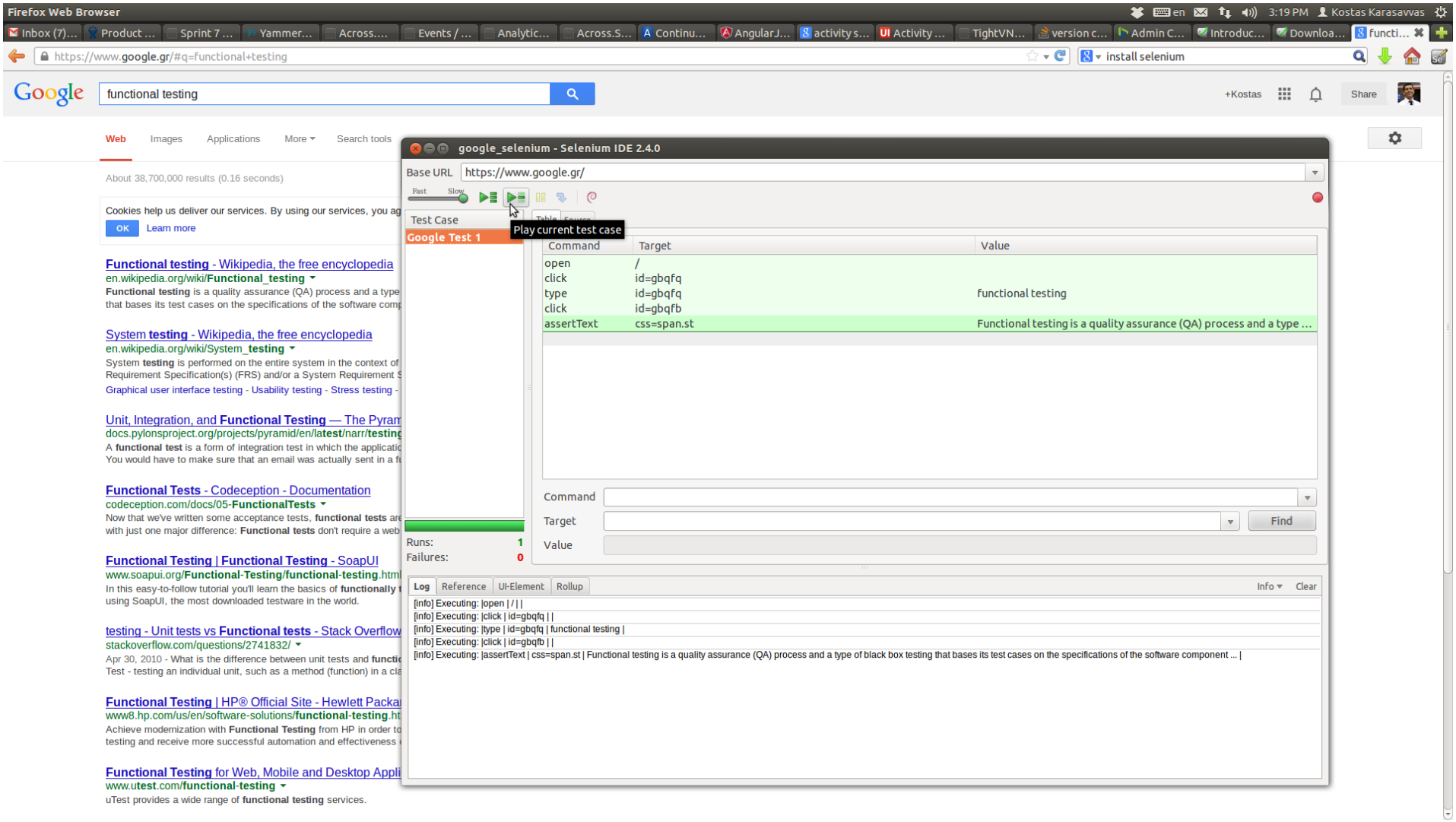
Selenium IDE Example, cont.



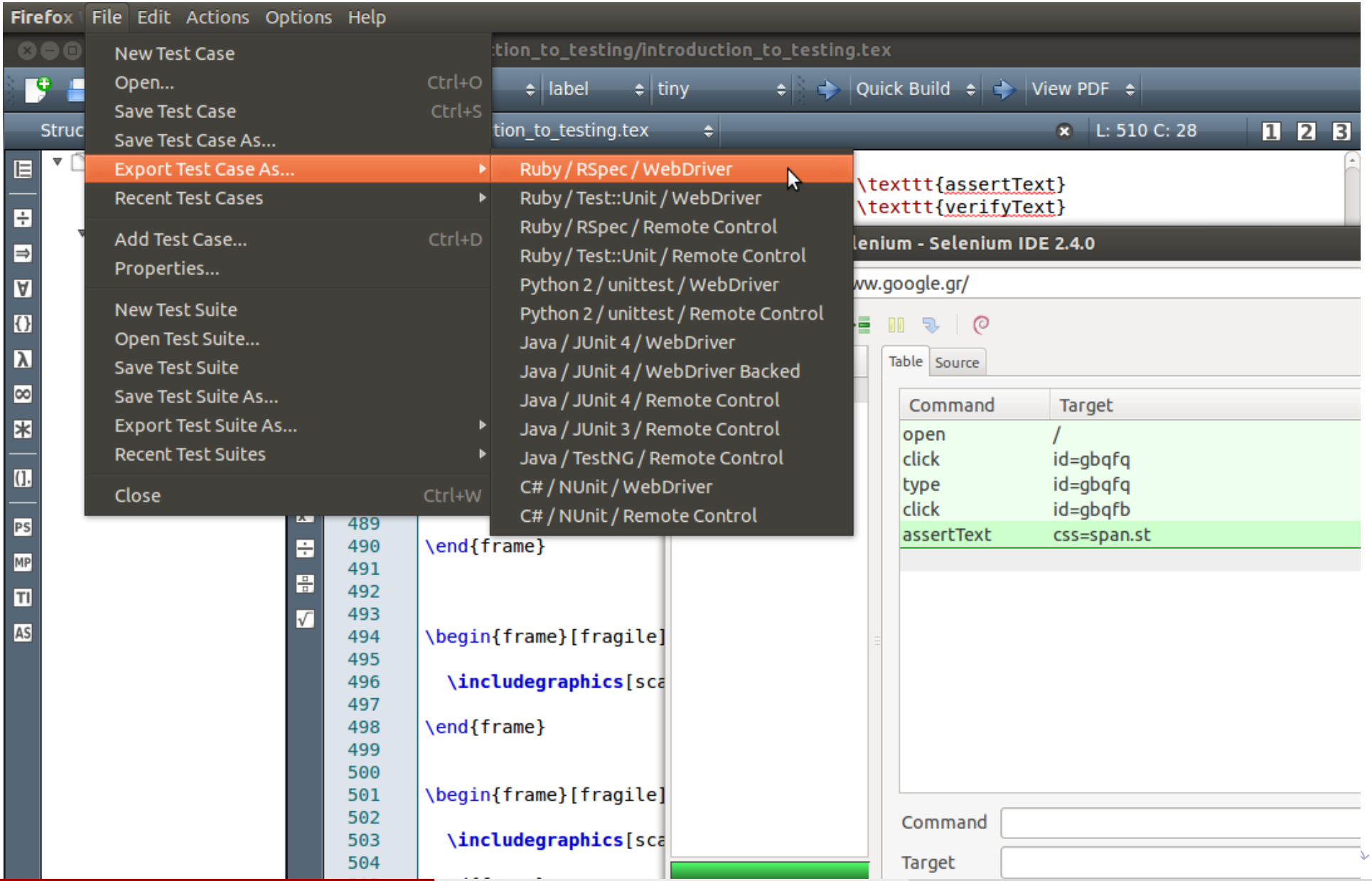
Selenium IDE Example, cont.



Selenium IDE Example, cont.



Selenium IDE Example, cont.



Selenium and RSpecs

```
require "json"
require "selenium-webdriver"
require "rspec"
include RSpec::Expectations

describe "GoogleTest1" do

  before(:each) do
    @driver = Selenium::WebDriver.for :firefox
    @base_url = "https://www.google.gr/"
    @accept_next_alert = true
    @driver.manage.timeouts.implicit_wait = 30
    @verification_errors = []
  end

  after(:each) do
    @driver.quit
    @verification_errors.should == []
  end

  it "test_google_test1" do
    @driver.get(@base_url + "/")
    @driver.find_element(:id, "gbqfq").click
    @driver.find_element(:id, "gbqfq").clear
    @driver.find_element(:id, "gbqfq").send_keys "functional testing"
    @driver.find_element(:id, "gbqfb").click
    (@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 ..."
  end
...
```

Selenium and RSpecs

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

```
$ rspec spec/google_selenium_rspec.rb
.

Finished in 16.47 seconds
1 example, 0 failures
```

- default browser will open and test *driven* visually