**Red Team**

**Stargazer**

**Automatic Telescope Control System**

**Testing Plan**

**Team Members:**

Jason Dunscombe

Rob Grmek

Robert Smith

**Instructor:**

**Youry Khmelevsky**

**Course:**

**COSC 470**

**Date:**

**November 18th, 2009**

# ****Revision History****

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 10/21/2009 | 1.0 | Document created. | Robert, Rob |
| 11/08/2009 | 1.1 | Added unit testing information. | Rob |
| 11/18/2009 | 1.2 | Added unit tests. | Rob |

# ****Table of Contents****

**Page**

|  |  |
| --- | --- |
| Unit Tests | 1 |
| Acceptance Tests | 3 |

# Unit Tests

We will be using the extreme programming style for all development in this project. This means we will be creating unit tests, using Ruby’s built-in unit testing, before writing any actual code. These tests will build on each other to form our test suite.

For a tutorial on using Ruby’s unit testing with Ruby on Rails, see here:

* <http://www.tutorialspoint.com/ruby-on-rails-2.1/rails-unit-testing.htm>
* <http://guides.rubyonrails.org/testing.html>
* <http://andrzejonsoftware.blogspot.com/2007/05/15-tdd-steps-to-create-rails.html>

**Example #1: creating a test case for a model (from SPIKE project):**

In ~/Blog/test/unit/post\_test.rb:

|  |
| --- |
| require 'test\_helper'  class PostTest < ActiveSupport::TestCase  def test\_should\_not\_save\_post\_without\_title  post = Post.new  assert !post.save, "Saved the post without a title"  end  end |

To run that single test method:

|  |
| --- |
| # from project directory root  cd test  ruby unit/post\_test.rb -n test\_should\_not\_save\_post\_without\_title |

To run the single test case:

|  |
| --- |
| # from project directory root  cd test  ruby unit/post\_test.rb |

To run all unit tests and functional tests:

|  |
| --- |
| rake test |

To run only all of the unit tests:

|  |
| --- |
| rake test:units |

To run only all of the functional tests:

|  |
| --- |
| rake test:functionals |

To run only recent tests for models and controllers that have been modified in last 10 minutes:

|  |
| --- |
| rake test:recent |

**Example #2: creating a test case for a controller (from SPIKE project):**

In ~/Blog/test/functional/post\_controller\_test.rb:

|  |
| --- |
| require 'test\_helper'  class PostsControllerTest < ActionController::TestCase  test "should get index" do  get :index  assert\_response :success  assert\_not\_nil assigns(:posts)  end  test "should get new" do  get :new  assert\_response :success  end  test "should create post" do  assert\_difference('Post.count') do  post :create, :post => { :name => "Bob", :title => "Something"}  end  assert\_redirected\_to post\_path(assigns(:post))  assert\_equal "Your post is saved!", flash[:notice]  end  test "should show post" do  get :show, :id => posts(:one).to\_param  assert\_response :success  end  test "should get edit" do  get :edit, :id => posts(:one).to\_param  assert\_response :success  end  test "should update post" do  put :update, :id => posts(:one).to\_param, :post => { }  assert\_redirected\_to post\_path(assigns(:post))  end  test "should destroy post" do  assert\_difference('Post.count', -1) do  delete :destroy, :id => posts(:one).to\_param  end  assert\_redirected\_to posts\_path  end  end |

user\_test.rb:

|  |
| --- |
| require 'test\_helper'  class UserTest < ActiveSupport::TestCase    # create a user with necessary, correct input values for the following tests  def setup  @user = User.new  @user.username = "unit\_test"  @user.email = "unit@test.com"  @user.password = "test"  @user.password\_confirmation = "test"  end    def test\_username\_cannot\_be\_null  @user.username = ""  assert !@user.save  end    def test\_username\_must\_be\_unique  # create another user with same name of "unit\_test"  assert @user.save  user2 = User.new  user2.username = "unit\_test"  user2.email = "unit2@test.com"  user2.password = "test2"  user2.password\_confirmation = "test2"  assert !user2.save  end    def test\_email\_cannot\_be\_null  @user.email = ""  assert !@user.save  end  def test\_email\_must\_look\_like\_an\_email\_address  # give it a non-valid looking email  @user.email = "test.ca"  assert !@user.save  @user.email = "test@ca"  assert !@user.save  @user.email = "test@testing.testing"  assert !@user.save  @user.email = "@testing.testing"  assert !@user.save  @user.email = "test@.com"  assert !@user.save    # give it a valid looking email  @user.email = "test@test.ca"  assert @user.save  end    def test\_password\_cannot\_be\_null  @user.password = ""  @user.password\_confirmation = ""  assert !@user.save  end    def test\_password\_must\_be\_confirmed  # do not confirm the password, should not save  @user.password = "hithere"  @user.password\_confirmation = ""  assert !@user.save    # confirm the password, should save  @user.password\_confirmation = "hithere"  assert @user.save  end    end |

schedule\_test.rb:

|  |
| --- |
| require 'test\_helper'  class ScheduleTest < ActiveSupport::TestCase  def setup  # create a schedule with necessary, correct input values for the following tests  @schedule = Schedule.new  @schedule.start\_time = "2014-05-06 04:36:00"  @schedule.latitude = 65.29  @schedule.longitude = 34.04  @schedule.exposure = 12  @schedule.number\_of\_pictures = 2  end    # Tests for schedule values:  def test\_schedule\_time\_cannot\_be\_before\_current\_time  @schedule.start\_time = "2004-05-06 04:36:00"  assert !@schedule.save, "Worked? Saved a schedule to position itself in the past."  end    # Tests for latitude coordinate values:  def test\_latitude\_coordinate\_must\_be\_a\_number  @schedule.latitude = "abc"  assert !@schedule.save, "Worked? Saved a non-number latitude value."  end    def test\_no\_negative\_latitude\_coordinate\_can\_be\_entered  # enter a negative number  @schedule.latitude = -15.04  assert !@schedule.save, "Worked? Saved a negative latitude value."    # enter a positive number  @schedule.latitude = 15.04  assert @schedule.save  end    def test\_user\_must\_enter\_value\_for\_latitude  @schedule.latitude = nil  assert !@schedule.save, "Worked? Saved a nil value for latitude."  end    # Tests for longitude coordinate values:  def test\_lonitude\_coordinate\_must\_be\_a\_number  @schedule.longitude = "abc"  assert !@schedule.save, "Worked? Saved a non-number longitude value."  end    def test\_no\_negative\_longitude\_coordinate\_can\_be\_entered  # enter a negative number  @schedule.longitude = -12.02  assert !@schedule.save, "Worked? Saved a negative longitude value."    # enter a positive number  @schedule.longitude = 12.02  assert @schedule.save  end    def test\_user\_must\_enter\_value\_for\_longitude  @schedule.longitude = nil  assert !@schedule.save, "Worked? Saved a nil value for longitude."  end    # Tests for exposure\_rating values:  def test\_exposure\_rating\_cannot\_be\_past\_set\_limits  @schedule.exposure = 14  assert @schedule.save, "Saved!"  end    def test\_exposure\_should\_be\_an\_integer  # try to enter a non-integer value  @schedule.exposure = "f"  assert !@schedule.save, "Worked? Saved a string."  @schedule.exposure = 2.01  assert !@schedule.save, "Worked? Saved a float."    # should be able to save when an integer is entered  @schedule.exposure = 2  assert @schedule.save  end    def test\_user\_must\_enter\_value\_for\_exposure  @schedule.exposure = nil  assert !@schedule.save, "Worked? Saved a nil value for exposure."  end    # Tests for number\_of\_pictures values:  def test\_number\_of\_pictures\_should\_be\_an\_integer  # try to enter a non-integer  @schedule.number\_of\_pictures = "f"  assert !@schedule.save, "Worked? Saved a string."  @schedule.number\_of\_pictures = 2.01  assert !@schedule.save, "Worked? Saved a float."    # should be able to save when an integer is entered  @schedule.number\_of\_pictures = 2  assert @schedule.save  end    def test\_user\_must\_enter\_value\_for\_number\_of\_pictures  @schedule.number\_of\_pictures = nil  assert !@schedule.save, "Worked? Saved a nil value for number\_of\_pictures."  end    end |

# Acceptance Tests

Needs to be implemented.