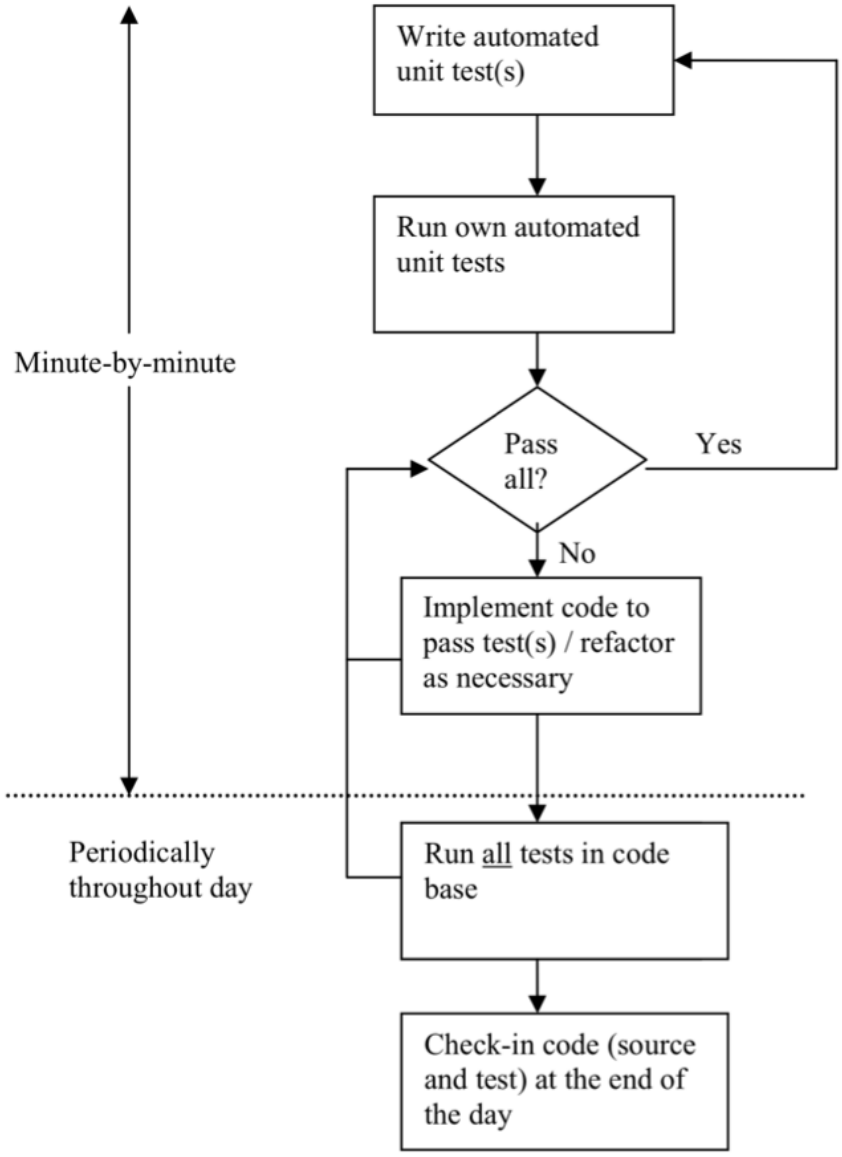
Exercise 5 Test Driven Design (TDD) with RSpec

SDM 409232 2015

Test Driven Design

TDD is an approach to iteratively designing and coding software that involves designing tests for features (at the method and class level) and then subsequently writing the test and then the feature code to pass the test, and then refactoring to meet quality code standards.

The following flowchart shows the steps and timescale of designing, writing and running tests, as well as writing the production code to be tested.



(Bhat & Nagappan, 2006)

There are many frameworks and tools to support TDD including JUnit for java and RSpec for Ruby and Rails. We will be using RSpec to write tests BEFORE writing production code. First it needs to be set up in the Gemfile.

Setup Files Ready for Rspec

To ensure we are also testing with the test database we have to modify the Gemfile

Under the group :test add the line for rspec gem

gem 'rspec-rails'

After modifying the gem run the bundle install without production environment

bundle install –without production

This will install all required rspec gems.

To setup rspec directories rails generator of rspec needs to be run from the root directory

rails generate rspec:install

This will create a spec directory with all its required files in it

We need to create the main directory that we will be working with

spec/controllers

In the folder you will need to create a spec controller file

spec/controllers/actors_controller_spec.rb

Open the file and enter require helpers files

require 'rails_helper'

require 'spec_helper'

To setup the test database for testing so that rspec points to a test database rather than your development database.

rake spec

This will now create a test.sqlite3 file in your db folder which will be used for insertions during your tests

Since we are going to be writing tests for the 'actors controller', navigate to your "**actors_controller.rb"** and remove any code that you currently have in the file

app/models/actors_controller.rb

The controller file will be used when we write tests in **actors_controller_spec.rb** file.

At this point we are ready to write tests

To run the test from the root directory of you application run

rspec

Running rspec command should give you

```
No examples found.


Finished in 0.00398 seconds (files took 2.57 seconds to load)
0 examples, 0 failures
```

Rspec Testing Schema

When testing with Rspec the basic outline is

describe "*something you want to test*" **do**

it "*an example of what you want to test*" **do**

ruby/rails code to test

end

end

To test for values there are range of methods available through rspec.

Some most commonly used ones are

expect(*expr***).to eq(***value***)**

expect(*expr***).to_not eq(***value***)**

1.0 Note:

There are many other rspec expectations that can be used

https://github.com/rspec/rspec-expectations

https://relishapp.com/rspec/rspec-expectations/docs

Provides guides on how to use some of the available expectations

Test Controller

actors_controller_spec.rb

```
describe ActorsController do

end
```

When you run rspec in terminal, it should give an error for uninitialized constant

```
/home/saasbook/Documents/wpm4240new/spec/controllers/actors_controller_spec.rb:5
:in `<top (required)>': uninitialized constant ActorsController (NameError)
```

Enter in the class initialization for ActorsController

actors_controller.rb

```
class ActorsController < ApplicationController

end
```

Run the test again and it should now pass but without any examples because we have not defined any examples yet to test

For this exercise we are going to test CRUD starting with index.

To perform task with the database we need to create a row to be entered into the database

```
describe  ActorsController do

    let!(:actor_setup)(Actor.create!(:name = > "Jim", :lname => "Buchan", :gender => "Male"))

end
```

We can now start by testing for the index template. We are going to copy the same template as the movies controller. We are going to return all the actors as well render an index template page

Working file: actors_controller_spec.rb

```
describe  ActorsController do

    let!(:actor_setup)(Actor.create!(:name = > "Jim", :lname => "Buchan", :gender => "Male"))

    describe "GET #index" do
        before {get :index}
        it "should assign @actors" do
            expect(assigns(:actors)).to eq([actor_setup])
        end
        it "should render a index template" do
            expect(response).to render_template("index")

        end
    end

end
```

Run the test on the command and watch the test fail

```
1) ActorsController GET #index should assign @actors
   Failure/Error: before {get :index}
   AbstractController::ActionNotFound:
     The action 'index' could not be found for ActorsController
```

```
2) ActorsController GET #index should render a index template
   Failure/Error: before {get :index}
   AbstractController::ActionNotFound:
     The action 'index' could not be found for ActorsController
```

1.1 Note:

If you have done exercise 4 these tests should pass, because the actors_controller.rb has the method index defined as well as the actors layout folder also has an index.html.haml file in it.

Test 1 requires that an @actors variable be passed back from the index method, however there is no method defined in the actors_controller.rb, therefore in the actors_controller.rb file add the get index method

Working file: actors_controller.rb

```
class ActorsController < ApplicationController

    def index
        @actors = Actor.all
    end
end
```

Even though you have written the code to pass the first test, both tests will still fail. You still have to add the index template in the actors view folder.

app/views/actors/***index.html.haml***

Once you have created the file, run rspec from the command line.

```
Finished in 0.09613 seconds (files took 2.63 seconds to load)
2 examples, 0 failures
```

Writing minimum code we have now got the tests to pass.

Similarly we want to write test for the main components of CRUD

Create, Read, Update and Destroy

Using the techniques used for testing the index we can continue on to write the tests for all the other need methods needed for the main functions of Actors.

The ***Show*** method, type ***get***

The ***New*** method, type ***get***

The ***Create*** method, type ***post***

The ***Edit*** method, type ***get***

The ***Update*** method, type ***put***

The ***Destroy*** method, type ***delete***

```
    movies GET    /movies(.:format)           movies#index
           POST   /movies(.:format)           movies#create
 new_movie GET    /movies/new(.:format)       movies#new
edit_movie GET    /movies/:id/edit(.:format)  movies#edit
     movie GET    /movies/:id(.:format)       movies#show
           PUT    /movies/:id(.:format)       movies#update
           DELETE /movies/:id(.:format)       movies#destroy
    actors GET    /actors(.:format)           actors#index
           POST   /actors(.:format)           actors#create
 new_actor GET    /actors/new(.:format)       actors#new
edit_actor GET    /actors/:id/edit(.:format)  actors#edit
     actor GET    /actors/:id(.:format)       actors#show
           PUT    /actors/:id(.:format)       actors#update
           DELETE /actors/:id(.:format)       actors#destroy
     casts GET    /casts(.:format)            casts#index
           POST   /casts(.:format)            casts#create
  new_cast GET    /casts/new(.:format)        casts#new
 edit_cast GET    /casts/:id/edit(.:format)   casts#edit
      cast GET    /casts/:id(.:format)        casts#show
           PUT    /casts/:id(.:format)        casts#update
           DELETE /casts/:id(.:format)        casts#destroy
      root        /                           :controller#:action
```

Get methods require templates, whereas the others need to be redirected to another path

Using this we can now setup up tests for the appropriate handlers

Every test block needs to be run and checked. Attempt to write code for each block before attempting next test block

Working file: actors_controller_spec.rb

```
describe  ActorsController do

    let!(:actor_setup)(Actor.create!(:name = > "Jim", :lname => "Buchan", :gender => "Male"))

    describe "GET #index" do
            before {get :index}
            it "should assign @actors" do
                    expect(assigns(:actors)).to eq([actor_setup])
            end
            it "should render a index template" do
                    expect(response).to render_template("index")

            end
    end

    describe "GET #show" do
            before {get :show, :id => actor._setup.id}
            it "should assign @actor" do
                    expect(assigns(:actor)).to eq(actor_setup)
            end
            it "should render the show template" do
                    expect(response).to render_template("show")
            end
    end

    describe "GET #new" do
            before{get :new}
            it "should assign @actor" do
                    expect(assigns(:actor)).to be_a_new(Actor)
            end
            it "should render a new template" do
                    expect(response).to render_template("new")
            end
    end

    describe "POST #create" do
            before {post :create, :actor => {:fname => "Jack", :lname => "Nicholson", :gender => "Male"}
            it "should redirect to the root path of the application" do
                    expect(response).to redirect_to(movies_path)
            end
    end

    # The additional tests for edit, update and destroy needs to be implemented here

    # use of context is recommended when you have different possible inputs from a post/put/update

end
```

Working file: actors_controller.rb

```
class ActorsController < ApplicationController

    def index
        @actors = Actor.all
    end

    def show
        @actor = Actor.find(params[:id])
    end

    def new
        @actor = Actor.new
    end

    def create
        actor = Actor.new(params[:actor])
        redirect_to movies_path

    end

end
```

The structure follows movies_controller.rb.

As you can see minimum code has been written to make the tests pass

Recommendation

Add further validation for inputs coming in from post.

Add further functionality for adding the new actor straight to the move as well as the actor database.