

Efficient Rails Test-Driven Development — Week 6

Wolfram Arnold
www.rubyfocus.biz

In collaboration with:
Sarah Allen, BlazingCloud.net
marakana.com

Integration Frameworks

Cucumber, Webrat, Capybara,...

lightweight browser without JS

fast

easy to set up

great for API testing

useful for exploratory application testing

Integration Frameworks

Selenium

drives a browser (Firefox, IE, Safari, Chrome...)

slow

many moving pieces

many technology choices

great if client/server testing is a must

useful for exploratory application testing

Selenium

Se

Selenium



Console Experiments

irb

> require 'support/boot'

> Page.start_new_browser_session

> ...

> Homepage.visit

> Page.selenium_driver.try_something_here

> ...

> Page.close_current_browser_session

Procedural Selenium Test

```
page.open
```

```
page.type "username", "joe"
```

```
page.type "password", "secret password"
```

```
page.click "submit", :wait_for => :page
```

```
page.text?("My HomePage").should be_true
```

```
page.click "link=New Messages"
```

```
...
```

Problems with this

- Very low level
 - hard to see what's being checked
 - hard to reuse
 - prone to copy & paste
- Next step
 - pull out common methods
 - but common file quickly gets cluttered
 - most tests only use a small fraction of common methods

Page Objects



Page Objects

Each Page represented by an Object

Methods are

- things you can *do*
- things you can *see*
- apply to *this page only*

Each method returns a page object

- itself
- another page object

Advantages

- Reuse is easy
- Documentation
- Maintainability
 - Changes to the UI or page require only one change of the test code, in only one place
- Method Chaining
 - Development of new tests is easy and fast
 - Easy reuse of existing “library of things to do” on a page

How to build Page Objects?

Model the UI in Page Objects

- entire pages

- parts of a page

Locators



Locators

Selenium Locators

“link=Sign in”

“username”

CSS Locators

“css=ul.global_nav_list li”

“css=input[name=username]”

XPath Locators

“xpath=//ul[@class='global_nav_list']/li”

“xpath=//input[@name='username']”

Selenium Locators—Default

“identifier” matches:

id attribute

```
element("identifier=navigation")
```

```
finds: <div id="navigation">
```

identifier keyword
is optional

This is the default
locator

name attribute

```
element("identifier=username")
```

```
finds: <input name="username">
```

*Note: matches id first, if not found, looks for
name attribute*

Other Selenium Locators

Explicit “id” locator:

matches id attribute

`element(“id=navigation”)`

finds: `<div id=“navigation”>`

Explicit “name” locator:

matches name attribute

`element(“name=username”)`

finds: `<input name=“username”>`

More on name locator

The “name” locator may be followed by filters:

```
element(“name=username value=Joe”)
```

Supported filters:

value=value pattern

index=element index, starting at 0

Link Locators

The “link” locator is used for links:

```
element("link=Sign in")
```

matches the text (pattern) of an a tag:

```
<a href="...">Sign in</a>
```

Select from Drop-Down

```
select(selectLocator, optionLocator)
```

optionLocator matches on label by default

Example:

```
select("name=language", "French")
```

Text Verification

glob:pattern

Similar to filename matching no command line:

* matches any sequence of characters

? matches any single character

regexp:pattern

match against a regular expression

regexpi:pattern

match against a regular expression, case-insensitive

exact:string

exact match

String matching examples

“Welcome John Smith”

glob:Welcome*

regexp:Welcome.*

regexpi:welcome.*

exact>Welcome John Smith

If nothing is specified, *glob is the default.*

CSS Locators

css=a

selects <a> tag

css=a.some_class

selects

css=a#some_id

selects

css=a.some_class#some_id

selects

CSS Locators: Attributes

css=input[name=username]

selects <input name="username">

css=img[src*="btn_img"]

selects

*= partial match

^= match beginning

\$= match end

CSS Locators Chained

tag1 tag2

decendents

css=ul.navlist li a

```
<ul class="navlist">
  <li>
    <a>Some link</a>
  </li>
</ul>
```

tag1+tag2

siblings

css=input+a

```
<input>
<a></a>
```



Waiting Rules

When to Wait?

An Event Trigger

Click

Mouse over/out

Form Submit

Waiting for:

Page load

AJAX load

JavaScript action (e.g. hide/show)

Waiting for Page

open “/”

will automatically wait for page

clicking on a link

wait_for :page

Wait for Element

Used for: Ajax, JavaScript

```
click "locator",  
  :wait_for => :element,  
  :element => "locator"
```

Wait for Visibility

Something appearing or disappearing dynamically

```
click "locator",  
  :wait_for => :visible,  
  :element => "locator"
```

Note: The element must be present (see wait for element) or visibility check will fail with a “no element found error”

RSpec Custom Matchers

```
[1,2,3] == [2,3,1]
```

```
([1,2,3] - [2,3,1]).should be_empty
```

```
[1,2,3].should be_commutative_with([2,3,1])
```

<http://wiki.github.com/dchelimsky/rspec/custom-matchers>

<http://railscasts.com/episodes/157-rspec-matchers-macros>

Testing for Access Control

Access Control

Controller access

Disallowed action should be blocked via `before_filters`

Most important!

View access

Disallowed pages/actions should not be linked to

Purely cosmetic

Devise

<http://github.com/plataformatec/devise>

Latest in Authorization Plugins

Comes with controllers and views

password recovery feature, etc.

Rack-based

Test Driven Development

Test-Driven Development

Benefit #1: Better Design

Testing as-you-go makes code better structured.

Testing emphasizes decoupling.

Fat model, skinny controller becomes easier.

Benefit #2: Focus & Project Management

If you don't know what to test for, how can you know what to code for?

Knowing when you're done.

Test-Driven Development

Benefit #3: Documentation & Collaboration

Tests are documentation that's always current.

Tests illustrate how code is meant to be used.

Benefit #4: Creation of Tests

Testing happens in-situ, not after the fact

When you're done coding, you've got a full test suite

No tedious chores and guilt afterwards

Inside-Out vs. Outside-In

Start with what you understand

can be model, view or controller

When you **discover** you need something else...

make failing tests pending

implement what you're missing

resume

Discover the application inside out.