RSpec on Rails Tutorial

https://ihower.tw

2016/8

Agenda

- 在 Rails 上安裝 RSpec
- 介紹 Model Spec, Routing Spec, Controller Spec, View Spec, Helper Spec
- 介紹 Request Spec 和 Feature Spec
- 各式疑難雜症
- 測試相關工具和 CI (Continuous Integration)
- 如何寫好 Web 測試？

Install rspec-rails

- gem “rspec-rails”
- bundle
- rails g rspec:install
- git rm -r test

rake -T spec

- rake spec
- bundle exec rspec rspec/xxx/xxx

Generators

- rails g model A
- rails g controller B
- rails g scaffold C

spec/rails_helper.rb
spec/spec_helper.rb

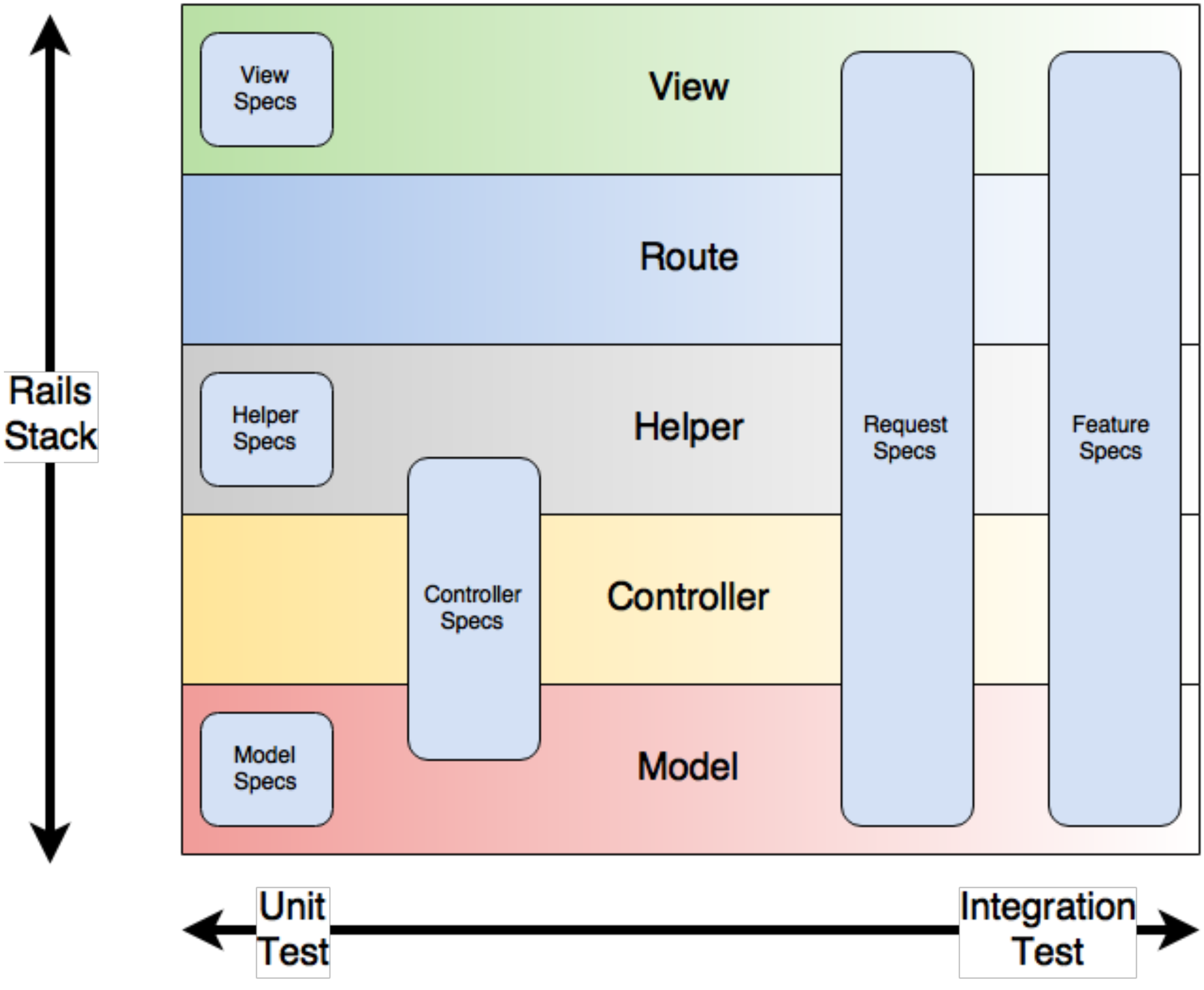
```
config.fail_fast = true

config.profile_examples = 3

config.order = :random
```

More Matchers

- expect(target).to eq(XXX)
- expect{ Post.find(9999) }.to raise_error(ActiveRecord::RecordNotFound)
- expect(target).to be_xxx # target.xxx?
- expect(target).to be_a_xxx
- expect(target).to be_an_xxx
- expect(collection).to be_empty
- expect([1,2,3]).to be_include(3)
- expect({ foo: "foo" }).to have_key(:foo)
- expect(3).to be_a_kind_of(Fixnum)
- 你也可以客製一個 Custom matcher

rspec-rails 測試分類

- 單元測試
  - model
  - controller (使用stub/mock)
  - view
  - helper
  - routing
- 整合測試
  - controller (不使用 stub/mock，直接使用 model 層)
- 驗收測試(跨controllers，開模擬瀏覽器)
  - request
  - feature (搭配 capybara)



https://robots.thoughtbot.com/rails-test-types-and-the-testing-pyramid

Model spec syntax

```
let(:valid_attributes){  { :name => "Train#123"}  }
  expect(Event.new).to_not be_valid
  expect(Event.new(valid_attributes)).to_not be_valid
```

Exercise 0

- 產生Rails 專案 (ticket_office)
- 安裝 rspec-rails gem
- 產生一個 scaffold 並觀察看看

Exercise 1:
Train Model Spec

- 產生 Train model
- 撰寫基本的 valid 測試

Kata

- Ticket Office
  - GET /trains/{train_id} 查座位資料
  - POST /trains/{train_id}/reservations 定位

Routing spec syntax

```
expect(:get => "/events").to route_to("events#index")
```

```
expect(:get => "/widgets/1/edit").not_to be_routable
```

But...

- 如果我們很確定正確性(例如那是 Rails 內建提供的方法)，那麼測試可以提供的價值就不高了
- 例如 resources 的 routing spec、model 的 validations 和 associations 常省略不寫
- 如果自訂 custom route 比較有機會用到

Controller spec syntax

```
get :show
post :create, :params => { :user => { :name => "a" } }
patch :update
delete :destroy

# more arguments
request.cookies[:foo] = "foo"
request.session[:bar] = “bar"

post :create, :params => { :name => "a" },
     :session => { :zoo => "zoo" },
     :flash => { :notice => "c"},
     :format => :html
```

注意: 用 params 是 Rails 5.0 之後才改的語法

Matcher syntax

```
expect(response).to render_template(:new)
expect(response).to redirect_to(events_url)
expect(response).to have_http_status(200)
expect(assigns(:event)).to be_a_new(Event)
```

Isolation Mode

- 預設 controller spec 是不會 render view 的，RSpec 希望你分層測試
- 可以用 render_views 打開

Exercise 2:
Train Controller show spec (stub version)

- 撰寫 trains/show 的測試和實作
- 分層測試：將 Train#find stub 掉，不碰 DB

View

isolated from controller too

```
assign(:widget, double("Widget", :name => "slicer"))
render
expect(rendered).to match /slicer/
```

Helper spec syntax

```
expect(helper.your_method).to eq("Q_Q")
```

Exercise 3:
Train show view

- 撰寫 train show 的 json view
- 使用 rails4 內建的 jbuilder
- 分層測試：Train 的方法尚未實作，stub 掉

Exercise 4: 實作

- 實作 Train, Seat, SeatReservation, Reservation models 和其關聯
- 實作 Train#available_seats 方法
- 可以拆掉 controller 和 view 的無謂 stub 了(完全拆掉or改用Partial Stub即可)

What have we learned?

- 把 stub&mock 當作臨時的設計工具
- 等真的物件跟方法實作出來，再拿掉多餘的 stub&mocks
  - 例如 ActiveRecord 內建的方法

Exercise 5: 實作

- 完成 ReservationsController 測試和實作
  - Train#reserve 尚未實作，可以先 mock 掉
- 完成 Train#reserve spec 測試和實作
  - 拆掉 ReservationsController 的 mock

Exercise 5`: 實作

- 先完成 Train#reserve spec 測試和實作
- 再完成 ReservationsController 測試和實作 (不用 mock)

Exercise 6: 錯誤處理

- GET /trains/{id} 找不到火車
- POST /trains/{id}/reservations 找不到座位
- POST /trains/{id}/reservations 座位已賣

Factory v.s. Fixtures

- rails 內建 fixtures，用 YAML 直接塞DB
  - 執行速度很快，但是比較脆弱不好維護，而且會跳過 model validation
- 比較多人用 factory 工廠方式，使用 ActiveRecord 去建立資料：
  - factory_girl gem 或 fabrication gem
  - 自己幹
- 工廠缺點：因為經過 ActiveReocrd 加上產生關聯很方便，所以執行速度較慢，建立出很多實際測試沒有用到的資料，造成測試速度變慢。
  - 可善用 factory_girl 的繼承、trait 功能將測試資料適當分類，每個 unit test 只建立必要的資料
  - 該次測試 model object 不需要存進 DB 的話，可用 build 就不要用 create，甚至用 build_stubbed

factory_girl 定義範例

```
FactoryGirl.define do

  factory :user do
    firstname "John"
    lastname "Doe"
    sequence(:email) { |n| "test#{n}@example.com"}
    association :profile, :factory => :profile
  end

  factory :profile do
    bio "ooxx"
  end

end
```

factory_girl 使用範例

```
before do
  @user = build(:user) # 不會存進 DB
  @event = create(:event) # 會存進 DB
end
```

```
it "should post user data"
  post :create, :params => { :user => attributes_for(:user) }
  # ...
end
```

参考資料

- https://github.com/thoughtbot/factory_girl/blob/master/GETTING_STARTED.md
- https://thoughtbot.com/upcase/videos/factory-girl

Tip: 使用 support 目錄

```ruby
Dir[Rails.root.join("spec/support/**/*.rb")].each { |f| require f }

# support/factory_helpers.rb
module FactoryHelpers
  # ...
end

Rspec.configure do |config|
  config.include FactoryHelpers
end
```

Exercise 7: Extract to factory method

- 將 Train 的建立測資 Extract 到 support/factory.rb

Tip: 如何 stub

```
before(:each) {
    allow(controller).to receive(:current_user) { ... }
}
```

Tip: 只測試 focus

- :focus => true 可以只測試目標 describe 或 it 區塊
- 用 rspec --tag focus
- 或是設定

```
config.filter_run :focus => true
config.run_all_when_everything_filtered = true
```

Request

- 目的是 full-stack 測試
  - 當然如果你要 stub 還是可以
- 通常拿來測試 Web APIs，例如 JSON, XML 等等
- 可以測試單一 Request，也可以測試跨 controllers
- 可以測試跨不同 sessions (不同使用者)
- 用法跟 Matchers 跟 controller spec 很像

Request spec syntax

```
describe "GET /events" do
  it "works! (now write some real specs)" do
    get “/events”
    expect(response).to have_http_status(200)
  end
end
```

Example: 跨 controller 的

```
it "creates a Widget and redirects to the Widget's page" do
  get "/widgets/new"
  expect(response).to render_template(:new)

  post "/widgets", :widget => {:name => "My Widget"}

  expect(response).to redirect_to(assigns(:widget))
  follow_redirect!

  expect(response).to render_template(:show)
  expect(response.body).to include("Widget was successfully created.")
end
```

Exercise 8:

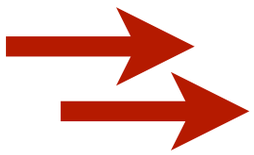
- 撰寫 1.查詢 2. 訂票 3. 查詢的完整流程測試

Feature spec

- 搭配 capybara gem 使用，跟 request spec 用途一樣
- http://rubydoc.info/github/jnicklas/capybara/master
- Capybara 的語法更適合測試 HTML 網頁

Capybara example

模擬瀏覽器行為



```
feature "signing up" do
  background do
    User.create(:email => 'user@example.com', :password => 'caplin')
  end

  scenario "signing in with correct credentials" do
    visit "/" # or root_path
    click_link 'Log In'
    within("#session") do
      fill_in 'Login', :with => 'user@example.com'
      fill_in 'Password', :with => 'caplin'
      choose('some_select_option_yes')
      check('some_checkbox')
    end
    click_button 'Sign in'

    expect(User.count).to eq(1) # you can test model
    expect(page).to have_content 'Login successfuuly' # and/or test page
  end
end
```

如果用文字比對太模糊，也可以用 find 搭配 css selector 或 xpath 去找元素

Debugging

- save_and_open_page
- 或是裝 capybara-screenshot gem
  - 測試出錯時自動儲存當時網頁

JavaScript Driver

- Capybara 預設是不會執行頁面上的 javascript 的
- 需要安裝 javascript_driver，注意都需要額外再裝非 Ruby 的套件，詳見 README
    - https://github.com/teampoltergeist/poltergeist
      用 PhantomJS
    - https://github.com/thoughtbot/capybara-webkit
      用 QtWebKit
    - https://rubygems.org/gems/selenium-webdriver
      用 Firefox
- 在需要的 test 用參數 js: true 打開

JavaScript Driver 的問題

- 這些 Browser tools 獨立於 Rails 之外，跟 Ruby 是不同 thread，造成一些非同步的問題
- 因為不在同一個 DB transaction 了，需要搭配 database cleaner
  - https://github.com/DatabaseCleaner/database_cleaner
  - 或 https://github.com/amatsuda/database_rewinder
  - 畫面上某些元素，需要等 javascript 執行完成才會出現(例如 Ajax)，所以第一次看到頁面時，可能還抓不到該元素進行測試
    - Capybara 碰到抓不到的元素會等 5 秒
    - Capybara.default_wait_time 這是預設的時間
    - 可用 using_wait_time(2) { …. } 局部變更時間

https://robots.thoughtbot.com/write-reliable-asynchronous-integration-tests-with-capybara

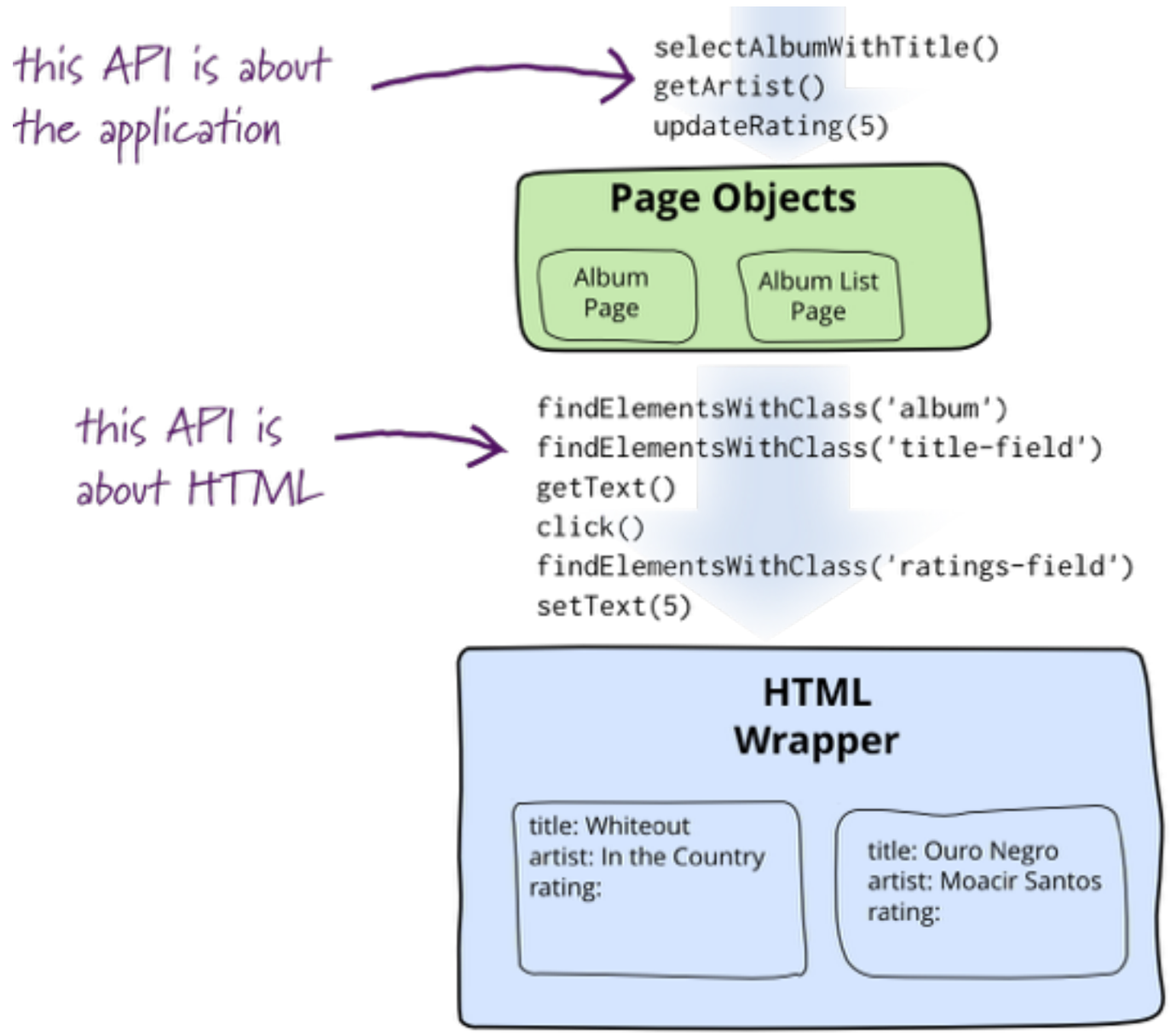
進一步組織重構

https://robots.thoughtbot.com/acceptance-tests-at-a-single-level-of-abstraction

- Extract behavior to helper methods
- Page Object

Page Object

http://www.infoq.com/cn/articles/martin-fowler-basic-rule-of-thumbon-for-Web-testing



Page Object example

https://teamgaslight.com/blog/6-ways-to-remove-pain-from-feature-testing-in-ruby-on-rails
https://thoughtbot.com/upcase/videos/page-objects
https://robots.thoughtbot.com/better-acceptance-tests-with-page-objects
https://medium.com/neo-innovation-ideas/clean-up-after-your-capybara-1a08b47a499b#.oyl7zi44d
https://www.sitepoint.com/testing-page-objects-siteprism/

疑難雜症 (1)

- 跑測試怎麼 Debugging ?
  - 用 puts 輸出在畫面上
    - https://tenderlovemaking.com/2016/02/05/i-am-a-puts-debuggerer.html
  - 也可用 byebug 下中斷點
  - 搭配 --only-failures option 可以只跑上次失敗的測試
    - https://relishapp.com/rspec/rspec-core/docs/command-line/only-failures

疑難雜症 (2)

- 有 Time.now 時間依賴要怎麼測試?
  - 參考 http://api.rubyonrails.org/classes/ActiveSupport/Testing/TimeHelpers.html 用 travel_to 方法
- config.include ActiveSupport::Testing::TimeHelpers
- 或 Timecop gem

疑難雜症 (3)

- email 怎麼測試?
  - mail = ActionMailer::Base.deliveries.last
  - 建議加上 config.before(:each) { ActionMailer::Base.deliveries.clear }
  - 或用 https://github.com/email-spec/email-spec/

疑難雜症 (4)

- 檔案上傳怎麼測試?
  - 可把測試檔案放 spec/fixtures/ 目錄下
  - 建立測資用 File.new(Rails.root + ‘spec/fixtures/foobar.png') 就可以讀到檔案了，搭配 paperclip 例如Photo.create(:description => "Test", :attachment => File.new(Rails.root + ‘spec/fixtures/ac_logo.png'))
  - feature spec 可用 capybara 的 attach_file 方法 http://www.rubydoc.info/github/jnicklas/capybara/master/Capybara%2FNode%2FActions%3Aattach_file

疑難雜症 (5)

- devise 請見 https://github.com/plataformatec/devise Test helpers 一節

```
config.include Devise::Test::ControllerHelpers, type: :controller
config.include Devise::Test::ControllerHelpers, type: :view
config.include Devise::Test::IntegrationHelpers, type: :feature
```

疑難雜症 (6)

- sidekiq 非同步怎麼測試?
  - 參考 http://api.rubyonrails.org/classes/ActiveJob/TestHelper.html#method-i-perform_enqueued_jobs
- config.include ActiveJob::TestHelper
- 在測試中 enqueue job 預設是不會執行的，要執行的話可用 perform_enqueued_jobs { … } 包起來

疑難雜症 (7)

- after_commit 怎麼測?
  - 因為每個 unit test 包在 transaction 中，造成 after_commit 不會正確執行
  - 可用 https://github.com/grosser/test_after_commit
  - 或用 database_cleanner 的 truncation 方式，不用 transaction
  - 或手動 trigger
    - http://mytrile.github.io/blog/2013/03/28/testing-after-commit-in-rspec/
  - Rails 5.0 之後不需要 workaround 了

疑難雜症 (8)

- rake 怎麼測試?
  - 把程式搬到 model 去，例如成為 class method
  - 寫測試寫在 model spec

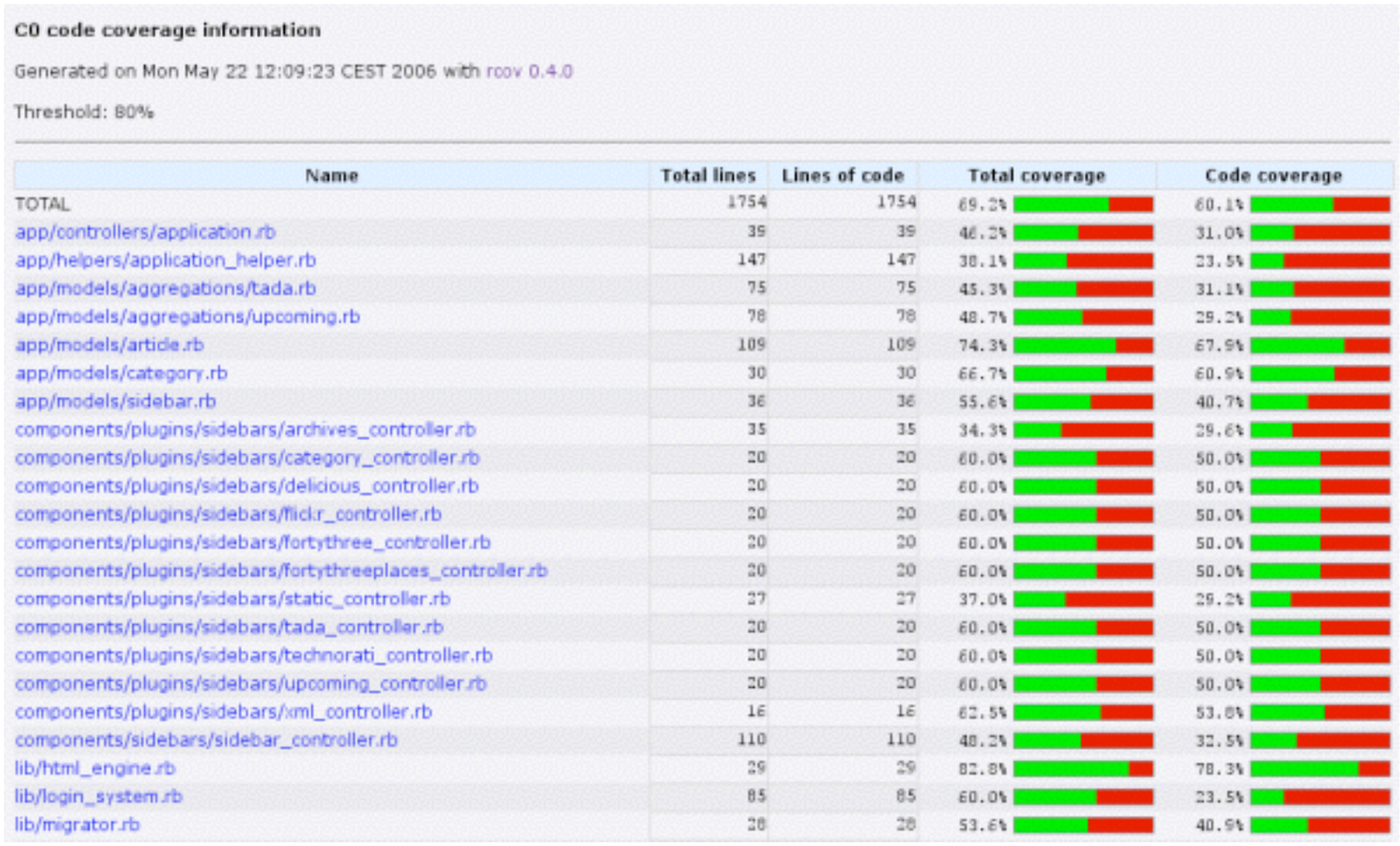
疑難雜症 (9)

- 測試跑太久怎麼辦?
  - 寫更多 Test double (?)
  - 還是善用多 CPU 比較簡單不會出錯
    - https://github.com/grosser/parallel_tests
    - 升級 CI 服務支援 concurrent build

其他 Tools 簡介

- shoulda 提供更多 rails 專屬 matcher
- database_cleaner 資料清理，支援不同DB
- vcr 錄 HTTP response 重播，配合 3-party service 測試
- simplecov 涵蓋度報表
- cucumber 將文件可執行化再推行一個層次
- CI (continuous integration (CI)

simplecov



C0 code coverage information

Generated on Mon May 22 12:09:23 CEST 2006 with rcov 0.4.0

Threshold: 80%

Name	Total lines	Lines of code	Total coverage	Code coverage
TOTAL	1754	1754	69.2%	60.1%
app/controllers/application.rb	39	39	46.2%	31.0%
app/helpers/application_helper.rb	147	147	38.1%	23.5%
app/models/aggregations/tada.rb	75	75	45.3%	31.1%
app/models/aggregations/upcoming.rb	78	78	48.7%	29.2%
app/models/article.rb	109	109	74.3%	67.9%
app/models/category.rb	30	30	66.7%	60.9%
app/models/sidebar.rb	36	36	55.6%	40.7%
components/plugins/sidebars/archives_controller.rb	35	35	34.3%	29.6%
components/plugins/sidebars/category_controller.rb	20	20	60.0%	50.0%
components/plugins/sidebars/delicious_controller.rb	20	20	60.0%	50.0%
components/plugins/sidebars/flickr_controller.rb	20	20	60.0%	50.0%
components/plugins/sidebars/fortythree_controller.rb	20	20	60.0%	50.0%
components/plugins/sidebars/fortythreeplaces_controller.rb	20	20	60.0%	50.0%
components/plugins/sidebars/static_controller.rb	27	27	37.0%	29.2%
components/plugins/sidebars/tada_controller.rb	20	20	60.0%	50.0%
components/plugins/sidebars/technorati_controller.rb	20	20	60.0%	50.0%
components/plugins/sidebars/upcoming_controller.rb	20	20	60.0%	50.0%
components/plugins/sidebars/xml_controller.rb	16	16	62.5%	53.8%
components/sidebars/sidebar_controller.rb	110	110	48.2%	32.5%
lib/html_engine.rb	29	29	82.8%	78.3%
lib/login_system.rb	85	85	60.0%	23.5%
lib/migrator.rb	28	28	53.6%	40.9%

哪些程式沒有測試到?

```
77  def full_html
78    "#{body_html}\n\n#{extended_html}"
79  end
80
81  protected
82
83  before_save :set_defaults, :transform_body
84
85  def set_defaults
86    self.published ||= 1
87    self.text_filter = config['text_filter'] if self.text_filter.blank?
88    self.permalink = self.stripped_title if self.attributes.include?("permalink") and self.permalink.blank?
89    self.guid = Digest::MD5.new(self.body.to_s+self.extended.to_s+self.title.to_s+self.permalink.to_s+self.author.to_s+Time.now.to_f.to_s
90  end
91
92  def transform_body
93    self.body_html = HtmlEngine.transform(body, self.text_filter)
94    self.extended_html = HtmlEngine.transform(extended, self.text_filter)
95  end
96
97  def self.time_delta(year, month = nil, day = nil)
98    from = Time.mktime(year, month || 1, day || 1)
99
100   to   = from + 1.year
101   to   = from + 1.month unless month.blank?
102   to   = from + 1.day   unless day.blank?
103   to   = to.tomorrow    unless month.blank?
104   return [from, to]
105 end
106
107 validates_uniqueness_of :guid
108 validates_presence_of :title
109 end
```

Generated using the rcov code coverage analysis tool for Ruby version 0.4.0.

Coverage 只是手段，不是目的!

CI 伺服器

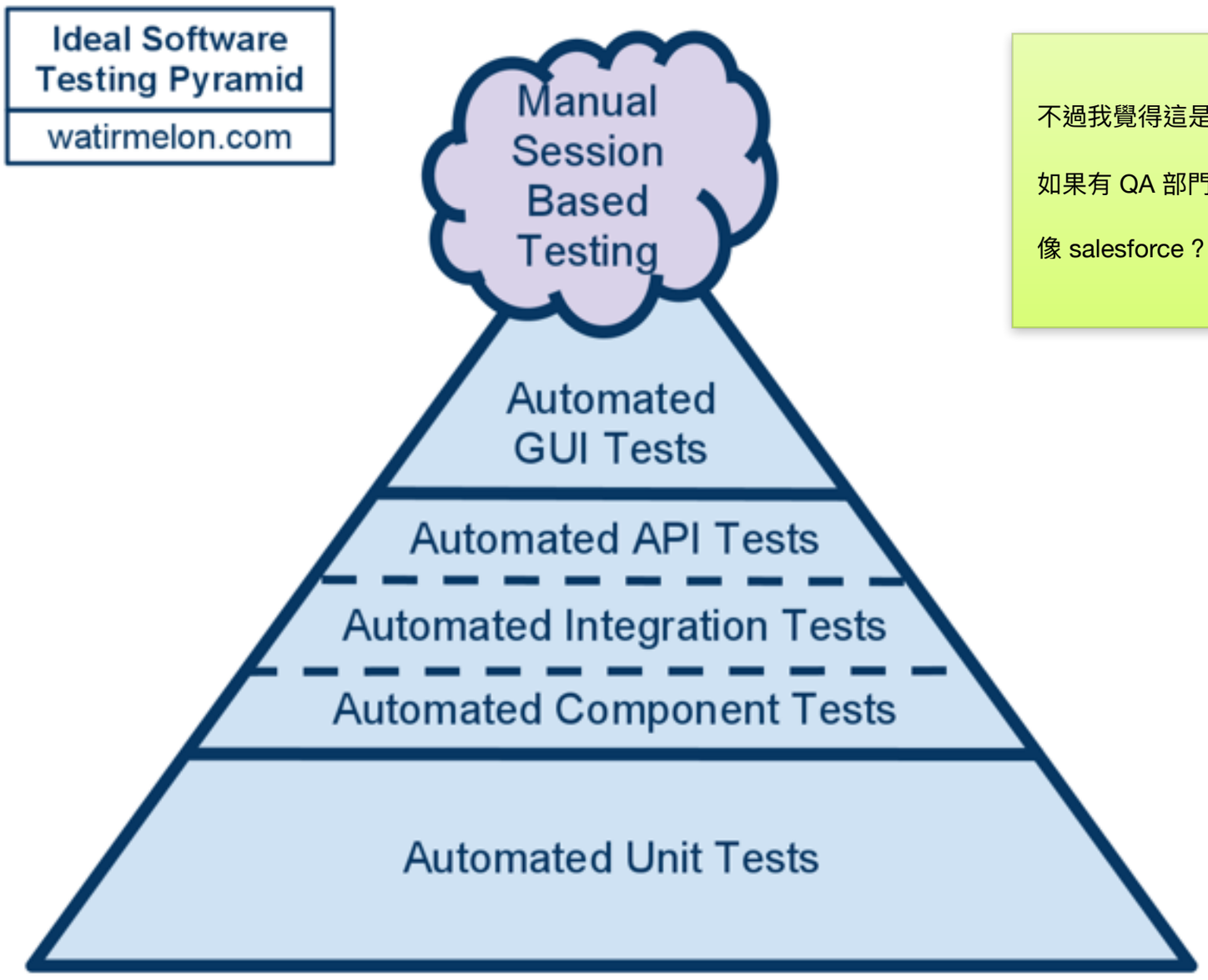
- 3-party service
  - https://www.codeship.io
  - https://circleci.com/
  - https://travis-ci.org/
- build your own
  - Jenkins

怎樣寫好測試?

Spec 文件化

- Rspec 提供了很好的 spec 撰寫架構
  - 測試訊息輸出
- 可以自訂 Custom Matcher

Test Pyramid

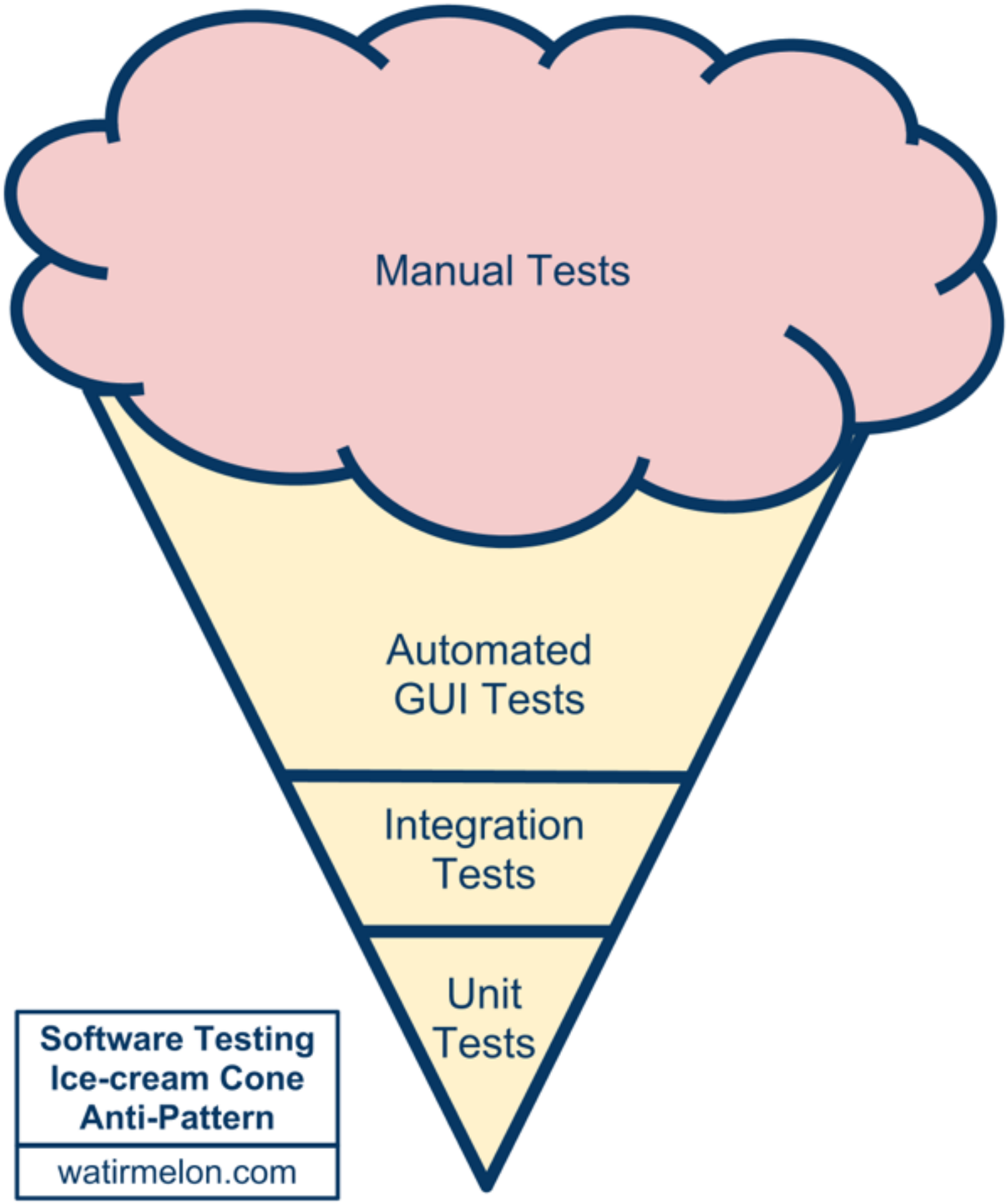


不過我覺得這是針對 developer 講的

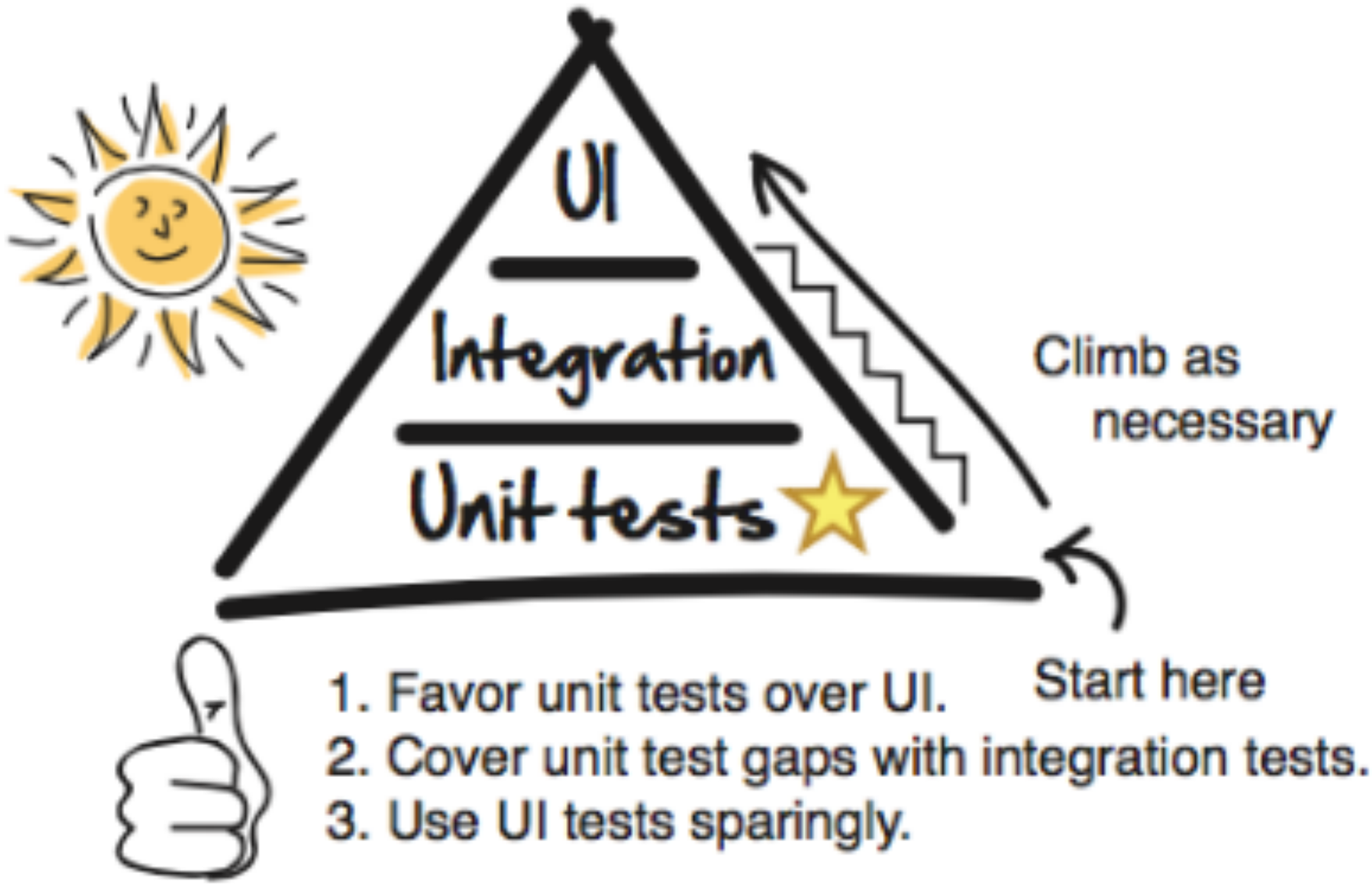
如果有 QA 部門，那還是可以做 XD

像 salesforce ?

http://watirmelon.com/2012/01/31/introducing-the-software-testing-ice-cream-cone/
http://martinfowler.com/bliki/TestPyramid.html


Manual Tests
Automated
GUI Tests
Integration
Tests
Unit
Tests
Software Testing
Ice-cream Cone
Anti-Pattern
watirmelon.com

What is the best way to test the web application?



https://www.quora.com/What-is-the-best-way-to-test-the-web-application

Why?

- 越 low-level 的測試，測試跑起來快，出錯時也容易 debug。
- 越 high-level 的測試，需要跑越久，而且出錯時也很難 debug。
  - 測試較脆弱，常 view 和流程一改，測試就要大改
  - 因此不建議 developer 投資太多，除非有專門的 QA 人員

單元測試壞味道

- 只是一個 bug，卻出來一堆 failures，不知如何 trace 起。
- 改了程式實作，結果一堆測試要跟著改
- 測試都有涵蓋到，但是還是沒捕捉到明顯錯誤 (剛才的 Mocks 例子)

https://thoughtbot.com/upcase/videos/testing-antipatterns

原則一：Isolation

- 一個 it 裡面只有一種測試目的，最好就只有一個 Expectation
- 盡量讓一個單元測試不被別的週邊因素影響
- one failure one problem
  這樣才容易 trace 問題所在

錯誤示範

```
describe "#amount" do
  it "should discount" do
    user.vip = true
    order.amount.should == 900
    user.vip = false
    order.amount.should == 1000
  end
end
```

善用 context

```
describe "#amount" do

  context "when user is vip" do
    it "should discount ten percent" do
      user.vip = true
      order.amount.should == 900
    end
  end

  context "when user is not vip" do
    it "should discount five percent" do
      user.vip = false
      order.amount.should == 1000
    end
  end

end
```

原則二：
盡量用較穩定的介面來進行測試

- 不要測 Private methods
- 抽象化可能會改變的實作，使用較穩定的介面來進行測試。

例一：

Private methods

```ruby
class Order

  def amount
    if @user.vip?
      self.caculate_amount_for_vip
    else
      self.caculate_amount_for_non_vip
    end
  end

  private

  def caculate_amount_for_vip
    # ...
  end

  def caculate_amount_for_non_vip
    # ...
  end

end
```

錯誤示範

```
it "should discount for vip" do
  @order.send(:caculate_amount_for_vip).should == 900
end
```

```
it "should discount for non vip" do
  @order.send(:caculate_amount_for_vip).should == 1000
end
```

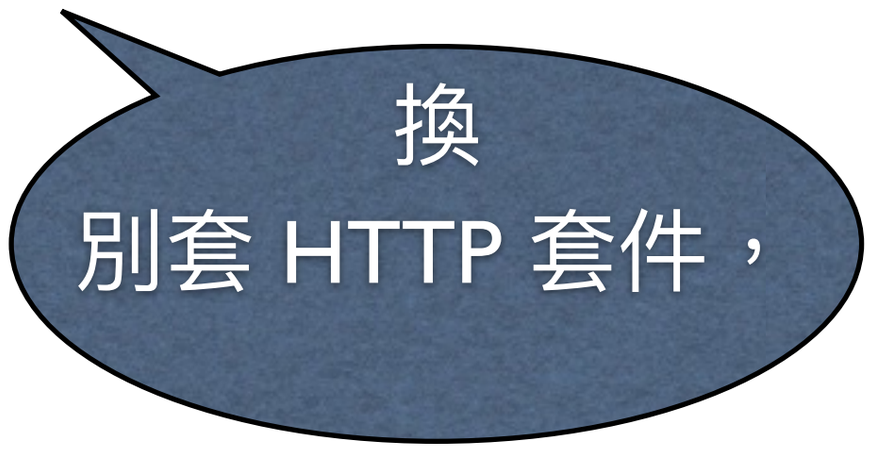
不要測 Private methods

- 測試 public 方法就可以同樣涵蓋到測試 private/protect methods 了
- 修改任何 private 方法，不需要重寫測試
  - 變更 public 方法的實作，只要不改介面，就不需要重寫測試
- 可以控制你的 Public methods 介面，如果只是類別內部使用，請放到 Private/Protected

例二：

測試特定的實作

```
describe User do

  describe '.search' do
    it 'searches Google for the given query' do
      HTTParty.should_receive(:get).with('http://www.google.com',
                                         :query => { :q => 'foo' } ).and_return([])
      User.search query
    end
  end

end
```

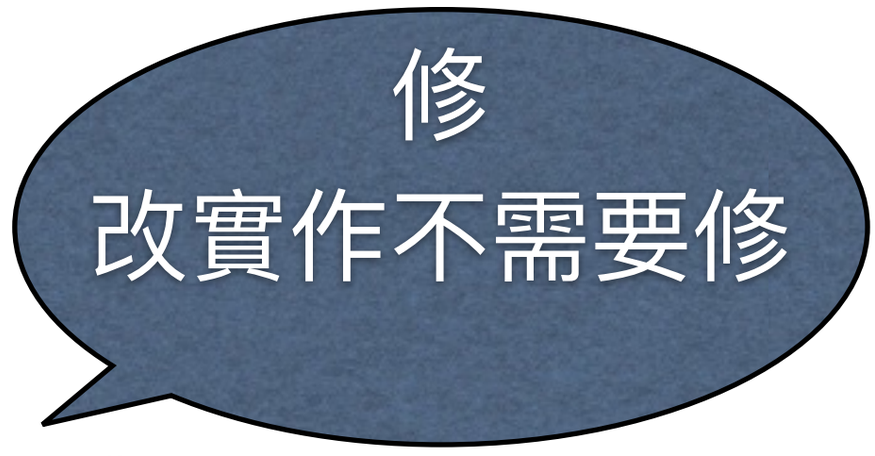


透過抽象化介面

```
describe User do

  describe '.search' do
    it 'searches for the given query' do
      User.searcher = Searcher
      Searcher.should_receive(:search).with('foo').and_return([])

      User.search query
    end
  end

end
```

對應的實作

```
class User < ActiveRecord::Base

  class_attribute :searcher

  def self.search(query)
    searcher.search query
  end

end

class Searcher

  def self.search(keyword, options={})
    HTTParty.get(keyword, options)
  end

end
```



哪些要測試?


I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO I DO IT IN PRODUCTION
memegenerator.net

TATFT

test all the f**king time

測試你最擔心出錯的部分，

不要等待完美的測試。

不要因為測試無法捕捉所有bug，
就不撰寫測試。
因為測試的確可以抓到多數 bug。

每當接獲 bug 回報，

請先撰寫一個單元測試來揭發。

DHH Way

- 100% test coverage 不是目標
- Code-to-test 比例超過 1:2 要注意，1:3 表示你寫太多測試了
- 不超過 1/3 的時間寫測試，超過一半時間你一定搞錯了
- 不要測試標準的 Active Record associations, validations, or scopes.
- 使用整合測試 (但不要測試 Unit Test 已經測到的東西)
- 不要用 Cucumber，別幻想了
  - 但其開發理論基礎「Specification by Example 中文版」仍推薦一讀
- 不要強迫自己用 TDD (DHH 大概只有 20% 用 TDD)
- Model 測試不需要隔離 DB、建議用 Fixtures 執行速度比較快
- Controller 測試用整合測試的方式
- Views 測試用 system/browser testing

https://signalvnoise.com/posts/3159-testing-like-the-tsa
http://david.heinemeierhansson.com/2014/test-induced-design-damage.html

如何鼓勵寫測試? 提高 coverage?

code review

Commit c75e4aeca398c1c48bf40ef24bd89067a87124fc to rails/rails – GitHub

https://github.com/rails/rails/commit/c75e4aeca398c1c48bf40ef24bd89067a87124fc#comments

Google

iGoogle Read Today Read Later Instapaper Reader Blog Twitter Facebook Plurk GitHub Tumblr Yahoo! WebTV Redmine(O) Redmine(T) 翻訳(G) 字典(Y)

```
273     -                   undef :#{method_name}
    272 +                 if method_defined?(:'#{method_name}')
    273 +                   undef :'#{method_name}'
274 274                   end
275 275                   def #{method_name}(*args)
276 276                     send(:#{matcher.method_missing_target}, '#{attr_name}', *args)
```

11 notes on commit c75e4ae

dkubb added a note to c75e4ae

Is the #{} interpolated inside a single quoted string? I didn't think it would be.

tenderlove added a note to c75e4ae repo collab

o_O

Common **@spastorino**.

pair programming

最後

寫測試的確不是萬能，不能保證程式的品質，
也不能保證你的專案不會 EPIC FAIL。

Twitter / @Wen-Tien Chang: 軟體開發沒有銀彈，只能靠多重手段避免失敗，pair ...

https://twitter.com/#!/ihower/statuses/48226324866793472

iGoogle Read Today Read Later Instapaper Reader Blog Twitter Facebook Plurk GitHub Tumblr Yahoo! WebTV Redmine(O) Redmine(T)

twitter Search Home Profile Messages Who To Follow ihower

@ihower
Wen-Tien Chang

軟體開發沒有銀彈，只能靠多重手段避免失敗，pair programming、code review、TDD、continuou integration、continuous deployment 這一路走來真是越用越多。讀 teddy-chen-tw.blogspot.com/2011/03/redund... 有感。

17 Mar via Twitter for Mac ☆ Favorite Reply Delete

Retweeted by deduce and others

別期待有時間補寫測試，現在就開始

邊開發邊寫 Unit Test，從做中學。

Reference:

- 官方文件 http://guides.rubyonrails.org/testing.html
- 更多範例 https://github.com/eliotsykes/rspec-rails-examples#api-request-specs-docs--helpers
- The RSpec Book
- The Rails 3 Way
- Foundation Rails 2
- xUnit Test Patterns
- everyday Rails Testing with RSpec
- http://betterspecs.org/
- http://pure-rspec-rubynation.heroku.com/
- http://jamesmead.org/talks/2007-07-09-introduction-to-mock-objects-in-ruby-at-lrug/
- http://martinfowler.com/articles/mocksArentStubs.html
- http://blog.rubybestpractices.com/posts/gregory/034-issue-5-testing-antipatterns.html
- http://blog.carbonfive.com/2011/02/11/better-mocking-in-ruby/

Thanks.