RSpec & TDD Tutorial

https://ihower.tw

2016/8

Who Am I?

- 張文鈿 a.k.a. ihower
  - https://ihower.tw
- Build web application since 2002

Ruby on Rails 實戰聖經

使用 Rails 4.2 及 Ruby 2.1

電子書和簡體版本準備中。如果您有任何意見、鼓勵或勘誤，歡迎來信給我，謝謝。

我是 ihower，本書介紹 Ruby on Rails 這套開放原始碼的網站開發框架。如果您對這本書有任何意見或勘誤指教，歡迎來信和我聯絡。

Part 1: 入門實作

Part 2: 深度剖析

ALPHAcamp
ABOUT US
BOOTCAMP
STUDENTS
SEMINARS
APPLY
COUNTRY
網站開發工程師實戰營
FULL TIME | 每週五天
十週 Ruby on Rails 課程實戰訓練，帶你學會 Rails、HTML、CSS 和 JavaScript，
打通前端、後端開發任督二脈，成為全球炙手可熱的網路全端工程師（Full Stack Developer），或親手實現創業的 idea！
瞭解更多
十週內打造全方位的 FULL STACK 開發能力
你將會學到
Ruby 程式語言
Rails 網站開發框架
前端網頁設計
版本控制開發流程
測試驅動開發
Web API 設計
網站效能調校
網站安全

Agenda

- 什麼是測試？寫測試有什麼好處？
- RSpec 語法介紹
- TDD Kata 練習

什麼是測試?

Twitter / @Wen-Tien Chang: 為什麼測試喜歡搞這麼多名目：單元測試、驗收測試、需求 ...

https://twitter.com/#!/ihower/status/25683492582

Google

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

twitter Search Home Profile Messages Who To Follow ihower

@ihower
Wen-Tien Chang

為什麼測試喜歡搞這麼多名目：單元測試、驗收測試、需求測試、煙霧測試、回歸測試、整合測試、系統測試、功能測試、確認(validation)測試、白箱測試、黑箱測試、灰箱測試、錯誤處理測試、壓力測試、效能測試、安全性測試、相容性測試、使用性測試、完整性測試、結構測試、安裝測試、

27 Sep via web ☆ Favorite Reply Delete

Retweeted by sandzhang and 15 others

Sorry, I’m not QA guy :/

測試 Overview

單元測試
Unit Test

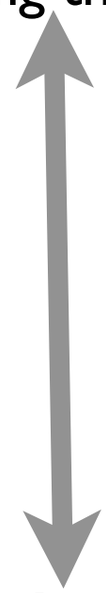
整合測試
Integration Test

驗收測試
Acceptance Test

Verification

確認程式的執行結果有沒有對?

Are we writing the code right?



Validation

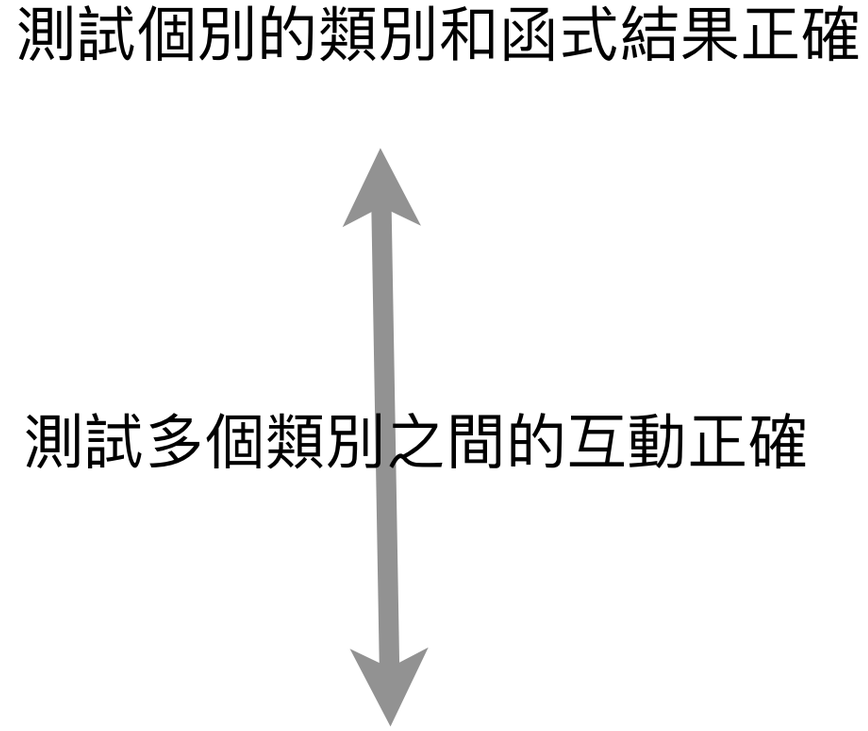
檢查這軟體有滿足用戶需求嗎?

Are we writing the right software?

單元測試
Unit Test

整合測試
Integration Test

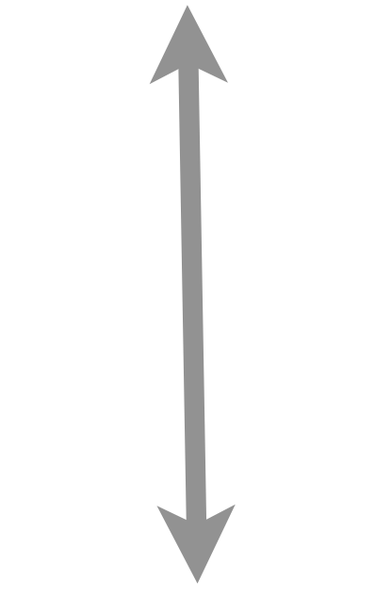
驗收測試
Acceptance Test



單元測試
Unit Test

整合測試
Integration Test

驗收測試
Acceptance Test

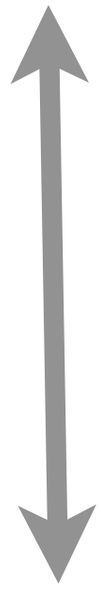


單元測試
Unit Test

整合測試
Integration Test

驗收測試
Acceptance Test

Developer 開發者



QA 測試人員

Verification

確認程式的執行結果有沒有對?

Are we writing the code right?

我們天天都在檢查程式的結果有沒有對

- 程式寫完後，手動執行看看結果對不對、打開瀏覽器看看結果對不對? (如果你是web developer 的話)
- 聽說有人只檢查 syntax 語法沒錯就 commit 程式的?

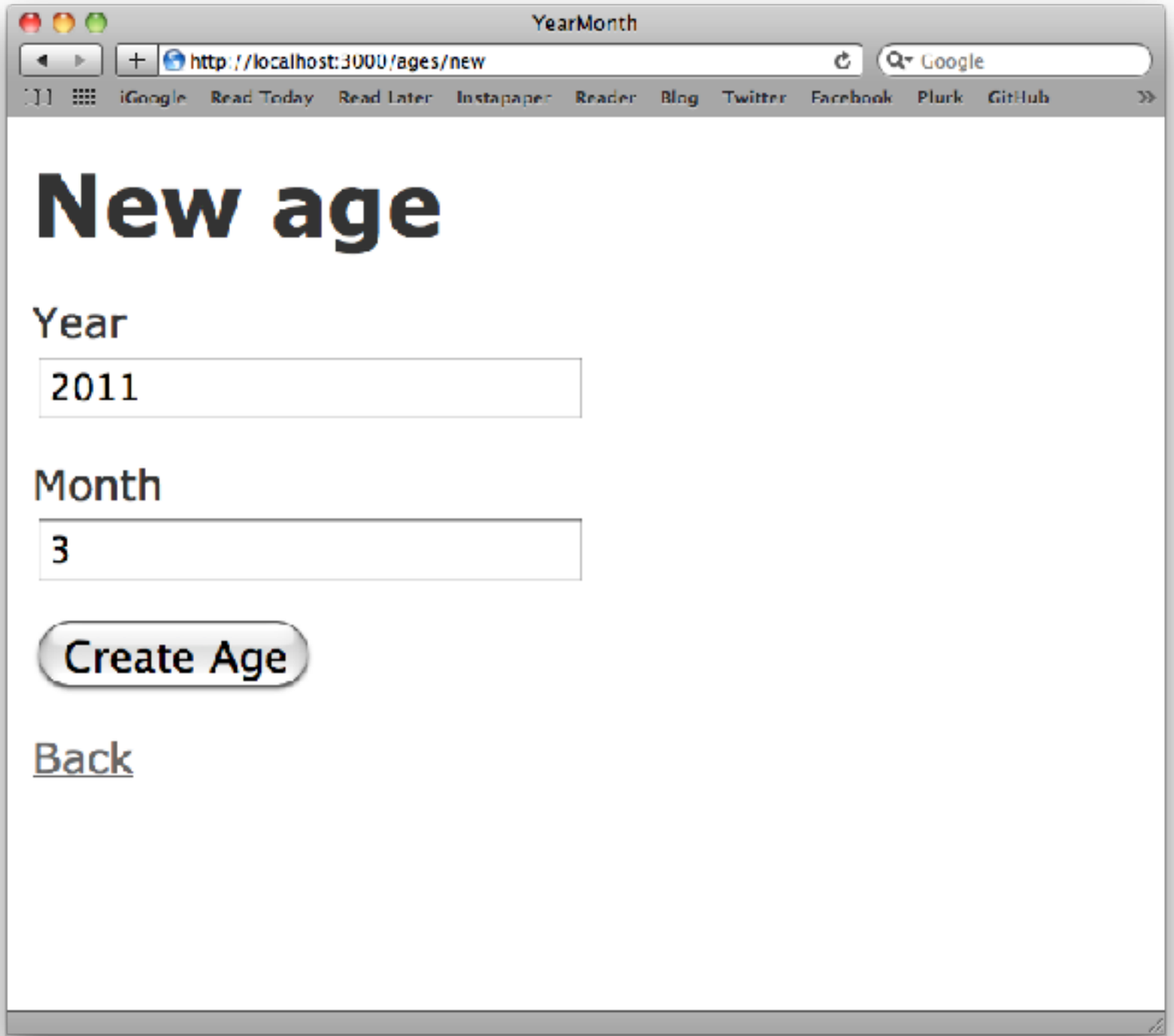
手動測試很沒效率

特別是複雜或 dependency 較多的程式，你可能會偷懶沒測試所有的執行路徑

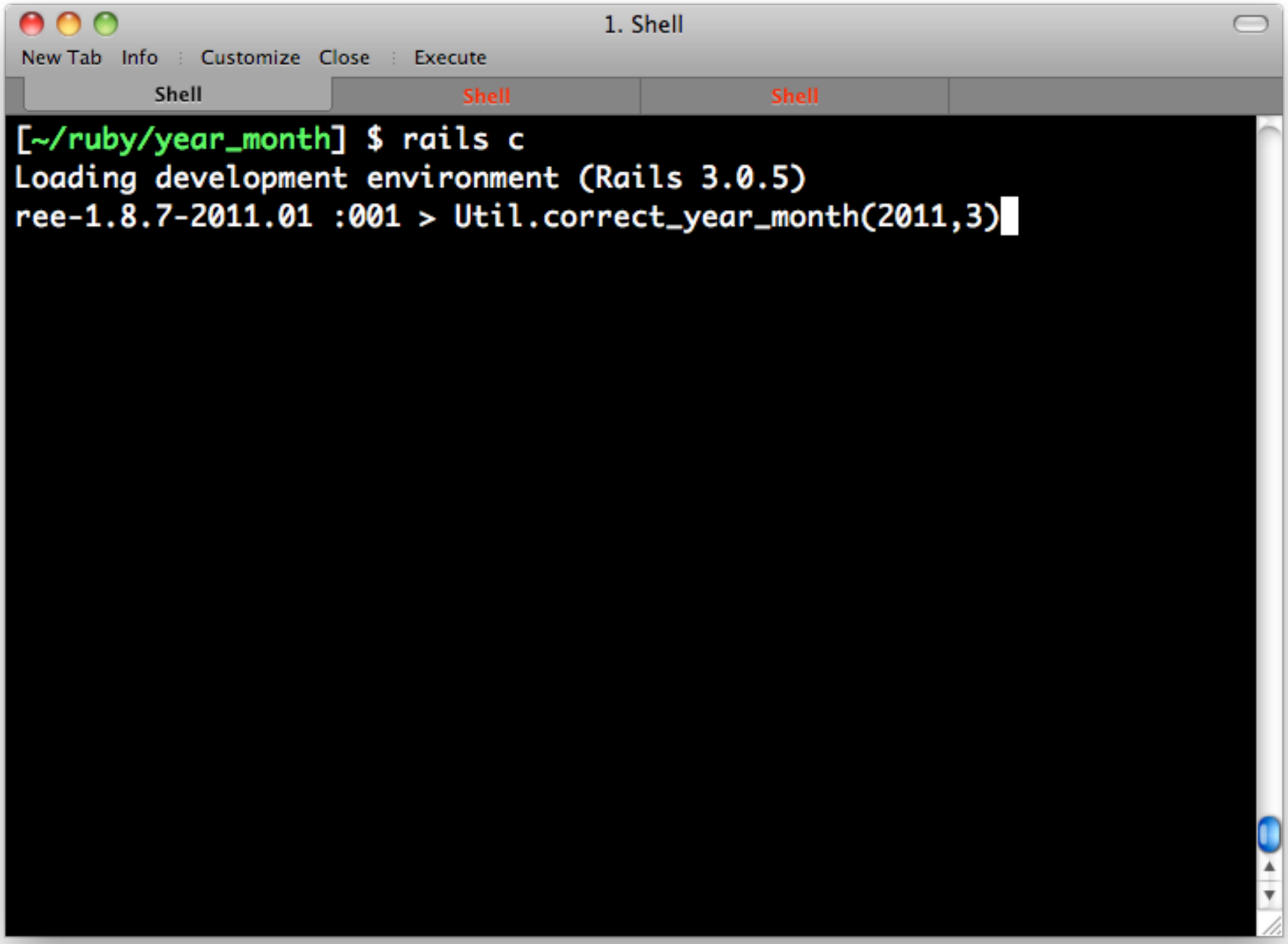
有三種路徑，你得執行(手動測試)三次才能完整檢查

```
def correct_year_month(year, month)

  if month > 12
    if month % 12 == 0
      year += (month - 12) / 12
      month = 12
    else
      year += month / 12
      month = month % 12
    end
  end

  return [year, month]

end
```

用介面手動測試?


YearMonth
http://localhost:3000/ages/new
Google
iGoogle
Read Today
Read Later
Instapaper
Reader
Blog
Twitter
Facebook
Plurk
GitHub
New age
Year
2011
Month
3
Create Age
Back

用 Console 介面手動測?



自動化測試

讓程式去測試程式

1. Instant Feedback

寫好測試後，很快就可以知道程式有沒有寫對

```
it "should return correct year and month" do

  expect(correct_year_month(2011,  3)).to eq [2011, 3]
  expect(correct_year_month(2011, 14)).to eq [2012,2]
  expect(correct_year_month(2011, 24)).to eq [2012,12]

end
```

寫測試的時間

小於

debug 除錯的時間

2. 回歸測試及重構

- 隨著程式越寫越多，新加的程式或修改，會不會搞爛現有的功能?
- 重構的時候，會不會造成破壞?
- 如果有之前寫好的測試程式，那們就可以幫助我們檢查。

3. 幫助你設計 API

- 寫測試的最佳時機點：先寫測試再實作
- TDD (Test-driven development)
  - Red-Green-Refactor 流程
- 從呼叫者的角度去看待程式，關注介面，協助設計出好用的 API。

4. 一種程式文件

- 不知道程式的 API 怎麼呼叫使用?
- 查測試程式怎麼寫的，就知道了。

小結

- 你的程式不 Trivial -> 寫測試節省開發時間
- 你的程式不是用過即丟 -> 回歸測試和重構
- TDD -> 設計出可測試，更好的 API 介面
- API 怎麼用 -> 測試也是一種文件

怎麼寫測試?

xUnit Framework

xUnit Framework

每個程式語言都有這樣的框架

- test case 依序執行各個 test :
  - Setup 建立初始狀態
  - Exercise 執行要測的程式
  - Verify (assertion)驗證狀態如預期
  - Teardown 結束清理資料

Ruby 的 Test::Unit

```ruby
class OrderTest < Test::Unit::TestCase

  def setup
    @order = Order.new
  end

  def test_order_status_when_initialized
    assert_equal @order.status, "New"
  end

  def test_order_amount_when_initialized
    assert_equal @order.amount, 0
  end

end
```

Method 的命名
是個麻煩

RSpec 寫法

```
describe Order do
  before do
    @order = Order.new
  end

  context "when initialized" do
    it "should have default status is New" do
      expect(@order.status).to eq(“New”)
    end

    it "should have default amount is 0" do
      expect(@order.amount).to eq(0)
    end
  end
end
```

RSpec 第一印象

- 語法 Syntax 不同
- 程式更好讀
- 更像一種 spec 文件

RSpec

- RSpec 是一種 Ruby 的測試 DSL
  (Domain-specific language)
  - Semantic Code：比 Test::Unit 更好讀，寫的人更容易描述測試目的
  - Self-documenting：可執行的規格文件
- 非常多的 Ruby on Rails 專案採用 RSpec 作為測試框架

RSpec (cont.)

- 不是全新的測試方法論
- 是一種改良版的 xUnit Framework
- 演進自 TDD，改稱 BDD (Behavior-driven development)
  - 先寫測試，後寫實作

Learn RSpec

- syntax
- syntax
- syntax
- more syntax

describe 和 context

幫助你組織分類

要測的東西是什麼?

```ruby
describe Order do
  # ...
end

# 或是

describe "A Order" do
  # ...
end
```

通常是一個類別

可以 Nested 加入想要測試的方法是哪個

```
describe Order do

  describe "#amount" do
    # ...
  end

end
```

可以再 nested 加入不同情境

```
describe Order do

  describe "#amount" do
    context "when user is vip" do
      # ...
    end

    context "when user is not vip" do
      # ...
    end
  end

end
```

每個 it 就是一小段測試

Assertions 又叫作 Expectations

加入 it

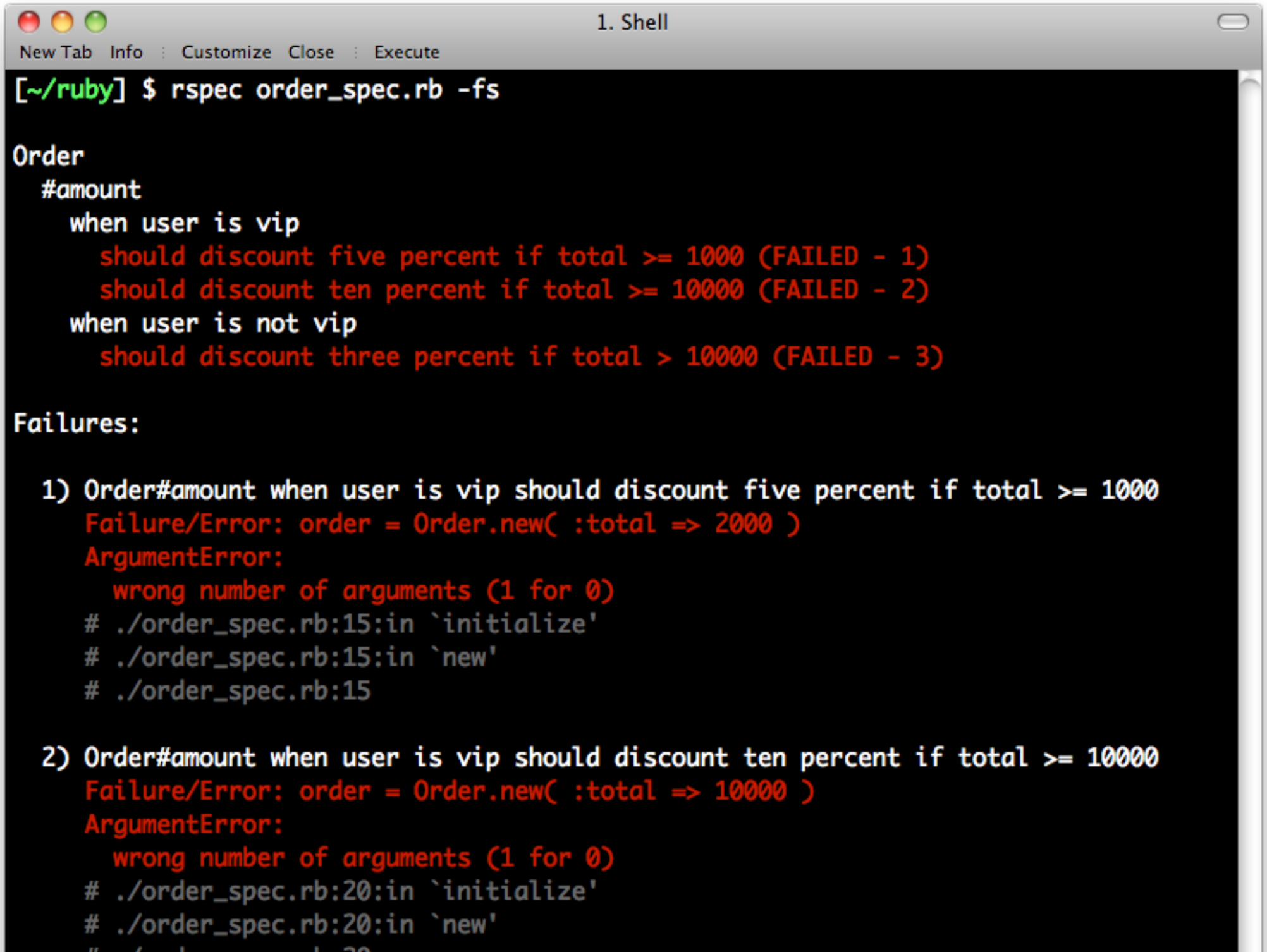
```
describe Order do

  describe "#amount" do
    context "when user is vip" do
      it "should discount five percent if total > 1000" do
        # ...
      end

      it "should discount ten percent if total > 10000" do
        # ...
      end
    end

    context "when user is not vip" do
      it "should discount three percent if total > 10000" do
        # ...
      end
    end
  end

end
```

expect(…).to 或 to_not

所有物件都有這個方法來定義你的期望

```ruby
describe Order do

  describe "#amount" do
    context "when user is vip" do

      it "should discount five percent if total >= 1000" do
        user = User.new( :is_vip => true )
        order = Order.new( :user => user, :total => 2000 )
        expect(order.amount).to eq(1900)
      end

      it "should discount ten percent if total >= 10000" { ... }
    end
    context "when user is vip" { ... }
  end

end
```

輸出結果(red)


1. Shell
New Tab Info Customize Close Execute
[~/ruby] $ rspec order_spec.rb -fs
Order
#amount
when user is vip
should discount five percent if total >= 1000 (FAILED - 1)
should discount ten percent if total >= 10000 (FAILED - 2)
when user is not vip
should discount three percent if total > 10000 (FAILED - 3)
Failures:
1) Order#amount when user is vip should discount five percent if total >= 1000
Failure/Error: order = Order.new(:total => 2000)
ArgumentError:
wrong number of arguments (1 for 0)
./order_spec.rb:15:in `initialize'
./order_spec.rb:15:in `new'
./order_spec.rb:15
2) Order#amount when user is vip should discount ten percent if total >= 10000
Failure/Error: order = Order.new(:total => 10000)
ArgumentError:
wrong number of arguments (1 for 0)
./order_spec.rb:20:in `initialize'
./order_spec.rb:20:in `new'

(狀態顯示為在寫 Order 主程式)

輸出結果(green)


1. Shell
New Tab Info Customize Close Execute
[~/ruby] rspec order_spec.rb -fs
Order
#amount
when user is vip
should discount five percent if total >= 1000
should discount ten percent if total >= 10000
when user is not vip
should discount three percent if total > 10000
Finished in 0.00102 seconds
3 examples, 0 failures
[~/ruby] $
漂亮的程式文件

before 和 after

- 如同 xUnit 框架的 setup 和 teardown
- before(:each) 每段 it 之前執行
- before(:all) 整段 describe 執行一次
- after(:each)
- afte(:all)
- 搭配 describe 和 context 放在適當的位置

```ruby
describe Order do

  describe "#amount" do
    context "when user is vip" do

      before(:each) do
        @user = User.new( :is_vip => true )
        @order = Order.new( :user => @user )
      end

      it "should discount five percent if total >= 1000" do
        @order.total = 2000
        @order.amount.should == 1900
      end

      it "should discount ten percent if total >= 10000" do
        @order.total = 10000
        @order.amount.should == 9000
      end

    end
    context "when user is vip" { ... }
  end

end
```

skip

略過不跑，可以先列出來打算要寫的測試

```
RSpec.describe "an example" do

  it "is skipped" # 沒有block
  skip "is skipped" do
  end

  xit "is skipped using xit" do
  end

end
```

```
[~/ruby] $ rspec order_spec.rb -fs

Order
  #amount
    when user is vip
      should discount five percent if total >= 1000
      should discount ten percent if total >= 10000
    when user is not vip
      should discount three percent if total > 10000
  #paid?
    should be false if status is new (PENDING: Not Yet Implemented)
    should be true if status is paid or shipping (PENDING: No reason given)

Pending:
  Order#paid? should be false if status is new
    # Not Yet Implemented
    # ./order_spec.rb:39
  Order#paid? should be true if status is paid or shipping
    # No reason given
    # ./order_spec.rb:41

Finished in 0.00137 seconds
5 examples, 0 failures, 2 pending
[~/ruby] $
```

pending

把失敗的測試暫時改成 pending

```
RSpec.describe "an example" do
  pending("something else getting finished") do
    expect(1).to eq(2)
  end
end
```

```
[~/ruby] $ rspec order_spec.rb -fs

Order
  #amount
    when user is vip
      should discount five percent if total >= 1000
      should discount ten percent if total >= 10000
    when user is not vip
      should discount three percent if total > 10000
  #paid?
    should be false if status is new (PENDING: Not Yet Implemented)
    should be true if status is paid or shipping (PENDING: No reason given)

Pending:
  Order#paid? should be false if status is new
    # Not Yet Implemented
    # ./order_spec.rb:39
  Order#paid? should be true if status is paid or shipping
    # No reason given
    # ./order_spec.rb:41

Finished in 0.00137 seconds
5 examples, 0 failures, 2 pending
[~/ruby] $
```

let(:name) { exp }

- 有使用到才會運算(lazy)，並且在同一個 example 測試中多次呼叫會 Memoized 快取起來。
- 相較於 before(:each) 可增加執行速度
- 因此就不用 instance variable 放 before 裡了，增加程式可讀性

```ruby
describe Order do

  describe "#amount" do
    context "when user is vip" do

      let(:user) { User.new( :is_vip => true ) }
      let(:order) { Order.new( :user => user ) }

      it "should discount five percent if total >= 1000" do
        order.total = 2000
        order.amount.should == 1900
      end

      it "should discount ten percent if total >= 10000" do
        order.total = 10000
        order.amount.should == 9000
      end

    end
    context "when user is vip" { ... }
  end

end
```

一些別名方法

哪個念起來順就用哪個

```
describe Order do # describe 和 context 其實是 alias

  # it, specify, example 其實都一樣
  it { … }
  specify { … }
  example { … }

end
```

一些慣例

- 一個 rb 檔案配一個同名的 _spec.rb 檔案
  - 容易找
  - guard 等工具容易設定
  - editor 有支援快速鍵
- describe “#title” 是 instance method
- describe “.title” 是 class method

輸出格式

- rspec filename.rb 預設不產生文件
- rspec filename.rb -fd 輸出 specdoc 文件
- rspec filename.rb -fh 輸出 html 文件

HTML format

鼓勵寫出高 spec coverage 程式，因為可以當做一種文件

RSpec results

file:///Users/ihower/ruby/report.html

Google

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

RSpec Code Examples

Passed Failed Pending

11 examples, 1 failure, 1 pending
Finished in 0.023062 seconds

Order

should be valid

status

should == "New"

#amount

when user is vip

should discount five percent if total >= 1000

should discount ten percent if total >= 10000

when user is not vip

should discount three percent if total > 10000

#paid?

should be false if status is new

should be true if status is paid or shipping

#receiver_name

should be user name

#ship!

with paid

should call gateway API

```
(Mock "ezcat").deliver(#[RSpec::Mocks::Mock:0x809e3830 @name="user"])
    expected: 1 time
    received: 0 times

./order_spec.rb:84
```

```
82      context "with paid" do
83        it "should call gateway API" do
```

Cheat Sheet

- http://www.rubypigeon.com/posts/rspec-core-cheat-sheet/
- http://www.rubypigeon.com/posts/rspec-expectations-cheat-sheet/

Code Kata

- 一種練習方法，透過小型程式題目進行鍛鍊，就像學功夫套拳，重複練功一樣
- 除了自己練，也可以 pair-programming
- TDD 需要練習才能抓到訣竅，與其說它是一種測試方式，不如說它更像一種設計手法
- 抓到 TDD 測試精神和訣竅，比學再多漂亮的語法糖更重要

https://blog.alphacamp.co/2015/03/02/tdd-kata/

準備 Kata 環境

https://github.com/ihower/ruby-kata

- Gemfile 設定 rspec, guard
- 安裝 growl notification (optional)
- bundle

Continuous Testing

- 使用 guard-rspec
  - 程式一修改完存檔，自動跑對應的測試
    - 節省時間，立即回饋

TDD 訣竅一：

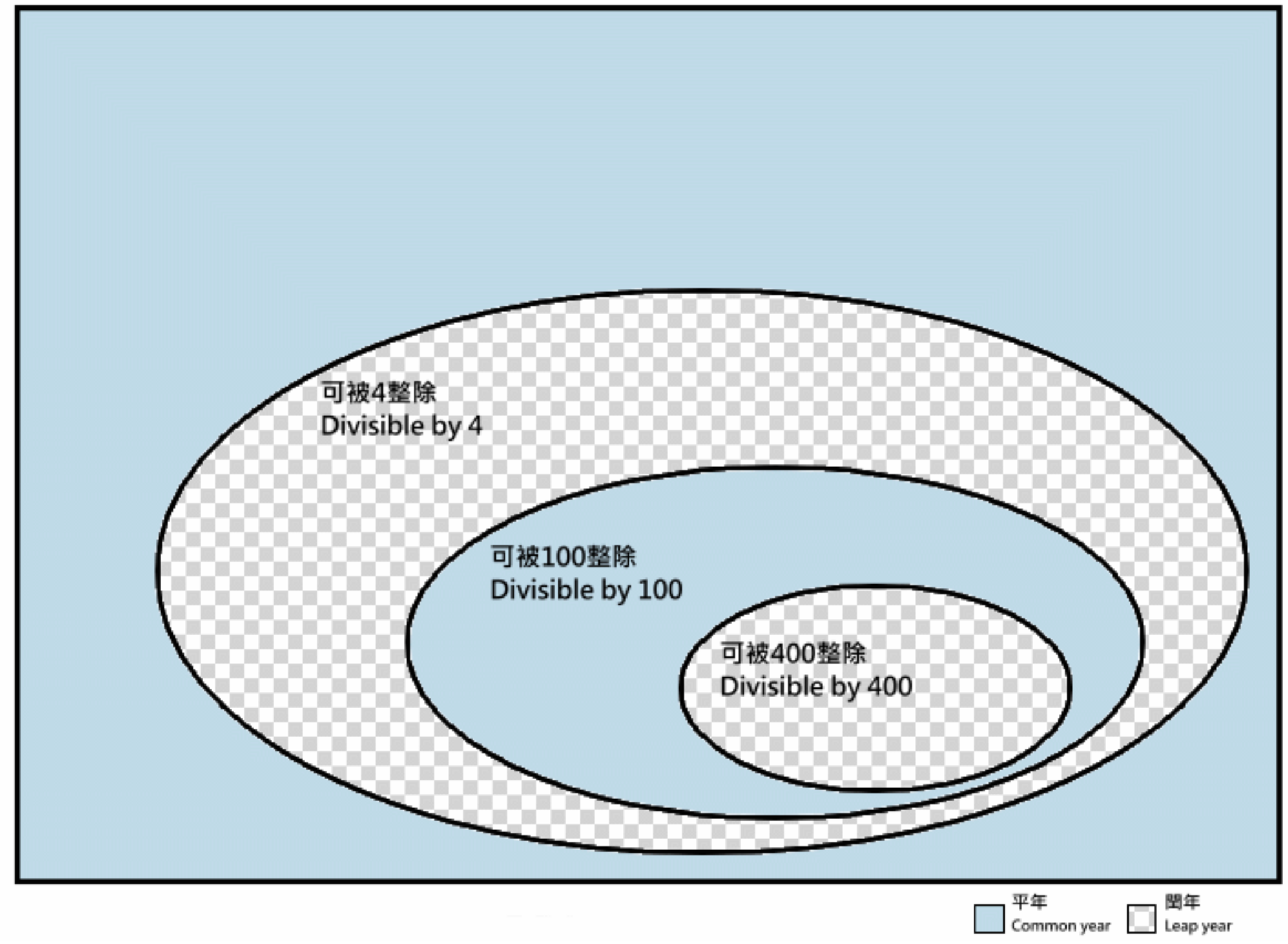
Red-Green-Refactor development cycle

FizzBuzz

- 逢三整除，輸出 Fizz
- 逢五整除，輸出 Buzz
- 逢三五整除，輸出 FizzBuzz

Leap Years

- 判斷閏年
  - 西元年份除以400可整除，為閏年。
  - 西元年份除以4可整除但除以100不可整除，為閏年。
  - 其他則是平年


可被4整除
Divisible by 4
可被100整除
Divisible by 100
可被400整除
Divisible by 400
平年
Common year
閏年
Leap year

Roman Numerals

- 轉換羅馬數字
  - I -> I
  - 4 -> IV
  - 5 -> V
  - 9 -> IX
  - 10 -> X
  - 20 -> XX

What have we learned?

- 一個 it 裡面只有一種測試目的，最好就只有一個 expectation
- 要先從測試 failed 失敗案例開始
- 確保每個測試都有效益，不會發生砍掉實作卻沒有造成任何測試失敗
- 一開始的實作不一定要先直攻一般解，可以一步一步在 cycle 中進行思考
- 測試程式碼的可讀性比 DRY 更重

As the tests get more specific, the code gets more generic.

Programmers make specific cases work by writing code that makes the general case work.

Three Laws of TDD

- 一定是先寫一個不通過的單元測試，才開始實作功能
- 每次只新加一個單元測試，只需要剛剛好不通過即可，不要一次加多個測試情境
- 每次實作功能時，只需要剛剛好通過測試即可，不多也不少

http://programmer.97things.oreilly.com/wiki/index.php/The_Three_Laws_of_Test-Driven_Development

TDD 訣竅二：

讓測試引導 API 設計

Bowling Game

http://www.sportcalculators.com/bowling-score-calculator

- 計算保齡球分數
- X (strike) 累加後兩球分數
- / (spare) 累加後一球分數

BowlingGame

roll
roll_many(1,2,3,4)
or roll_many(“1234512345”)
finished?
score

What have we learned?

- 透過寫測試，定義出哪些是類別公開的介面(API)
  - 測試碼方便呼叫的API，就是好API
  - 不需要公開的，定義成 private 方法，讓實作有更好的物件導向封裝。
  - 不需要針對 private 方法直接寫單元測試，而是透過 public 方法間接測試

Reference:

- The RSpec Book
- The Rails 3 Way
- Foundation Rails 2
- xUnit Test Patterns
- everyday Rails Testing with RSpec
- 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.