Day1. 動機－自我挑戰的開始。

這一次用的是什麼語言?

用哪個框架?

Day2. 什麼叫單元測試? 再給我說一次，什麼叫他\_\_的單元測試?

什麼叫做好的測試

3A原則



Day3. 來談點自動化測試吧

Day4. 套路，到處都是套路

[](http://www.zuilxy.com/wp-content/uploads/1214.jpg)

Day4. 從MS Test開始，創建一個測試專案

介紹一些Attribute，Run first 、initial等等

Day1.

動機

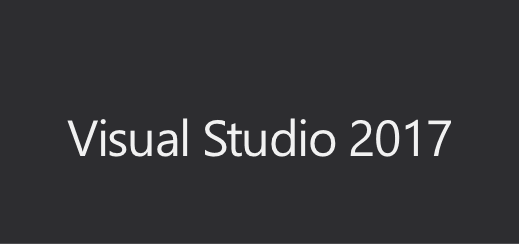
Hi，大家好，我是DZ，這一次是我第一次參加IT鐵人競賽，希望可以透過 ~~(自己挖的坑)~~ 30天的寫作關於測試、關於TDD，讓自己可以對於TDD的練習能夠更加的精進



練習TDD對我來說，不是練習寫Unit Test，而是讓我能夠了解如何拆解需求，把需求拆得更細，讓我可以針對每一個小需求進行測試，經過這樣不斷的實作，刻意練習，進而讓TDD的過程變得像是呼吸一樣正常，紅燈、綠燈、重構這個過程也會在每一天的文章中出現，而這一次的挑戰過程也會是紅燈、綠燈、重構。

這一次的寫作內容主要會先從何謂測試、測試的定義到開始用TDD練習，TDD練習的內容就會從拆解題目開始，紅燈，綠燈，重構最後將所有需求完成需求並驗證成功，帶大家一步一步的了解測試並利用TDD來進行實作。

這次要用的語言是C#，是小弟我較熟悉的語言，測試的框架就使用MS Test來進行TDD的練習



小弟我目前接觸的開發項目不多，都是遇到強者的提點而進化，其中一個就是91大大，在此感謝他點燃我對於開發的熱情，讓我開始用寫作來內化自己對於技術上的了解，透過寫文章的方式讓自己透過自己的方式來描述一件事，就可以知道自己對於某一種技術了解到了多少甚至可以知道有沒有缺漏。

我很喜歡之前與91私聊時的這一句話 :

*我們需要找到一種持續給自己回饋的方式，才能持續改善  
所以 scrum 裡面有 iteration (迭代)  
所以有 retrospective  
一切都是持續改善的基本要素*

目錄

此目錄為暫時的目錄，每日更新時會補充連結及主題

* Day1. 動機－自我挑戰的開始。
* Day2. 什麼叫單元測試? 再給我說一次，什麼叫他\_\_的單元測試?
* Day3. 來談點測試吧
* Day4. 套路，到處都是套路
* Day5. 套路的開始，第一個題目。

Day2.

前言

這一篇，就先來簡單的談談什麼叫做「單元測試」~~你再給我翻譯一下 什麼他\_\_\_的叫單元測試~~

這一篇將會談談單元測試原則、FIRST和3A原則

~~~~

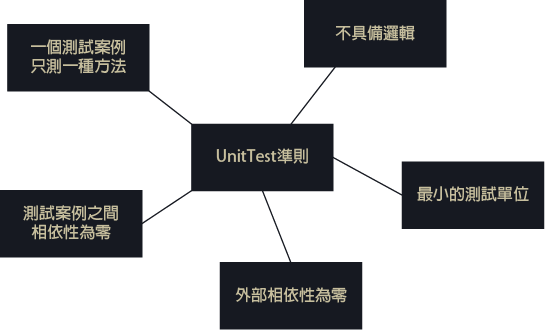
什麼叫單元測試?

單元測試，英文又稱為Unit Test，就是針對程式最小單元，進行的測試，稱之為單元測試。

單元測試的準則

現在我們終於知道了單元測試以及最小單元的定義，做單元測試之前，我們要知道要保持什麼樣的原則及特性才能成功的做出「好的」單元測試。

下圖參考[91大大的部落格的圖](https://dotblogs.com.tw/hatelove/archive/2012/11/05/learning-tdd-in-30-days-day2-unit-testing-introduction.aspx)翻做出來的Unit Test的定義及基本準則圖



1. 一個測試案例只測一種方法

若一個測試案例測了多種方法，那你這個測試案例就會很複雜，假設今天這個測試案例中有很多種方法，如果這個測試案例錯了，你會不知道錯在哪一個方法之中

1. 最小的測試單位
2. 不與外部（包括檔案、資料庫、網路、服務、物件、類別）直接相依

不與外部相依的原因是需要排除「不是在我們的方法上出錯」的可能性，假設相依在線上的資料庫上，很有可能會因為網路、資料庫被異動而導致測試案例失敗的可能，所以不與外部直接相依，而是以stubs、mock的方式來進行單元測試。

1. 不具備邏輯

單元測是本身不可以具備邏輯，如果單元測試具備了邏輯那就需要在寫一個測試來測試這個測試案例是不是正確的，這樣一看就知道是一個不符合邏輯以及奇怪的事情。

1. 測試案例之間相依性為零

測試案例之間如果有相依性，那就很有可能會造成測試案例之間的「干擾」，假設某一個B測試案例需要A測試案例先啟動才會通過，如果A測試案例不過，B測試案例就不會過，這樣就會干擾我們去判斷今天是因為錯了A才錯B還是B本身就是錯的。

什麼叫做最小單元?

我就想先讓你翻譯翻譯，什麼叫最小單元?



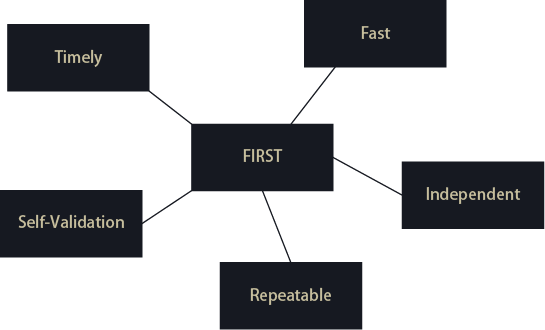
最小就是單個程式、函式、過程等，在物件導向中的解釋，就是在基礎類別、抽象類別或者子類別中的「方法」，就是最小單元。

好的單元測試－FIRST特性

大哥這個是FIRST原則啊



如何定義這個單元測試是不是好的單元測試?，一個字FIRST(參考於無暇程式碼)



1. Fast：快速。

好的單元測試要可以快速的執行完成。

1. Independent：獨立。

好的單元測試要是獨立性的，也就是無相依的。

1. Repeatable：可重複。

好的單元測試要是可以重複執行的。

1. Self-Validating：可反應驗證結果。

好的單元測試是要可以真的驗證一件事情的結果的，不需要透過人為的操作來進行驗證。

1. Timely：及時。

單元測試的程式碼與產品程式碼是兩個一起完成的。

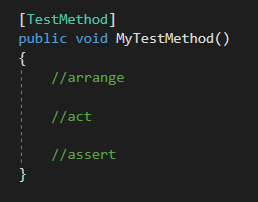
談了這麼多的準則、原則，這些雖然看起來很明瞭，但卻又很模糊，那我們在實作上到底要如何開始呢?

單元測試就是3A原則

呼呼，終於說到了最後一個原則啦!!!



3A原則是單元測試的開始，常常在寫測試的時候不知道該如何開始就可以直接在程式碼寫下這三行再開始



不過這3A到底是甚麼意思呢?

1. Arrange

初始化物件、相依物件、方法參數、預期結果或是預期的互動方式

1. Act

呼叫被測試的目標物件方法

1. Assert

驗證是否符合預期

Note:這一個3A原則最主要是用來輔助撰寫的測試程式碼，讓我們的測試程式碼更好理解。



談點常常看到的情形

當然，在這一個巨大的開發世界中，我們常常會遇到某些單元會覺得也太大一個了吧！！~~(跟我現在的肚子一樣)~~

這些一開始嬌小可愛的「單元」就跟剛開始交往的女/男朋友一樣，好看，好維護，好測試。

只要一不注意，這些嬌小可愛的「單元」就會像交往多年的女/男朋友一樣，經過每一次的需求或版本的更新 ~~(吃飯、出遊)~~，不知不覺就變大、變肥、變得不好維護和測試了，不過只要有愛~~(錢、薪水)~~就可以把這個單元~~(感情?)~~繼續更新和維護下去。

不是大隻就是不好，而是「不健康」。 (如果你跟成吉思汗館長一樣壯碩大隻那另當別論了)，程式碼就跟人一樣，太肥就會讓人覺得「不健康」。

通常看到這一些變肥的單元，我們必須要想辦法把它拆解，重構，讓這些單元「可測試化」，但我們這一次要說的主題是預防寫出難以「可測試」的程式，所以我們需要TDD來幫助我們寫出「可測試」的程式。

(廢話 不能測試的話 就沒辦法開發了啊XDDDD

我們，下一篇見!

Day3. 來談一點點的測試

這一篇將會談一點點地測試和為什麼需要自動化測試。

測試在開發過程中是不可或缺的一環，軟體上的測試方法分了兩種

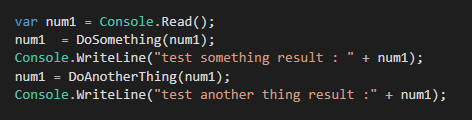
* 黑箱測試(black-box testing)

這種測試方法主要測試的是應用程式的功能，而不是其內部結構或過程，此種方法大部分應用於整合測試和系統測試。

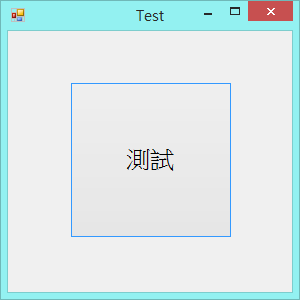
* 白箱測試(white-box testing)

和黑箱測試箱法的測試方法，他主要測試的就是內部結構或過程，通常會由開發者以程式語言的角度來設計測試案例。

或許你可能沒有做過「真正的測試」，但是你肯定寫過類似這種東西



甚至是…



你發現了嗎?我們其實都已經做過了測試，但這些方式都十分的沒效率，但又時常出現在我們的開發過程中，並且反覆的出現，甚至很常需要重複的build程式碼→啟動→輸入→驗證，無限的輪迴，並驗證我們自己所打的Code到底是否為我們預期，這樣子的過程，就可以稱之為測試。

害我想到…..



在沒有自動化的情況之下，我們每一次進行軟體的測試都必須要Build、啟動、輸入(執行)、驗證，每一個步驟都是重複的。

我們人類最不喜歡做的就是重複的動作，但也時常因為精神上的狀況，而產生「就算我們做的是重複的動作，但我們也很有可能會發生錯誤」的問題，不過電腦就跟我們人類不一樣，電腦擅長做重複且乏味的事情。

自動化測試的基礎就是建構於我們讓電腦幫我們做測試這件事情，我們只要告訴電腦，我要做哪些事情，我要讓他做10遍、100遍甚至是1000遍都可以做出同樣的結果，只要我們給電腦的指令是正確的，那電腦就會忠實的反饋給我們它執行的結果。

自動化測試的存在是為了有效率的我們認為重複性高且簡單的測試案例。