測試驅動開法與持續重構

Chris Chang

鴻揚科技有限公司

測試驅動開發

大綱

- 測試驅動開發(TDD)介紹
- Test-Driven
- 開始 TDD 循環
- Tennis Kata Demo
- 困難點

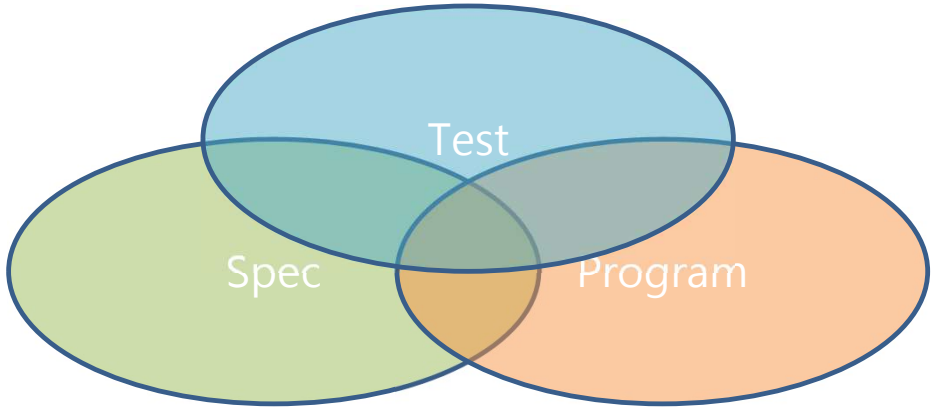
TDD 介紹

- 是一種軟體開發的方法，並不是軟體測試的方法，其倡導先寫測試程式，再實作產品代碼

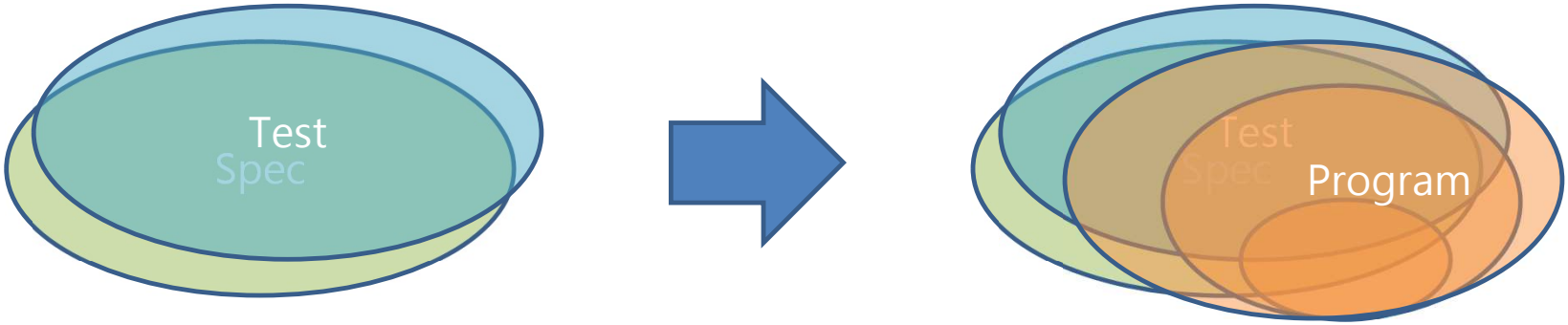
Test-Driven

- 描述需求使用情境
- 輔助開發、表達期望、當作目標
- 溝通協作、驗證疑問
- 設計高易用性 API

需求、功能、測試 (現況)

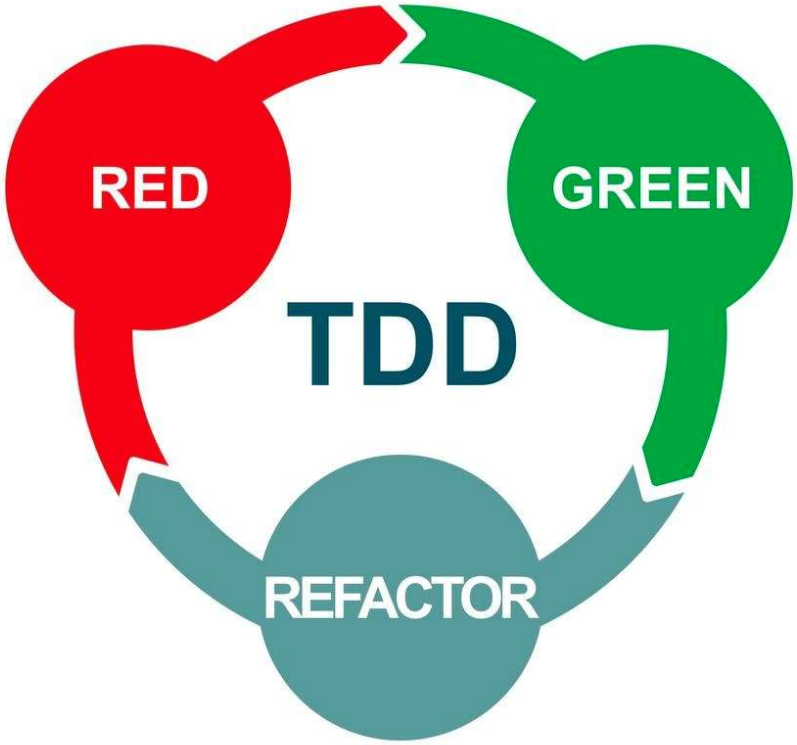


需求、功能、測試 (TDD)



開始 TDD 循環

- 紅綠燈
- Baby Step



TDD 循環 (Step 1)

- 新增會失敗的測試程式 (使用者情境)
  - 僅定義 API 規格及其預期結果，不實作內容
  - 思考怎麼使用目標程式

TDD 循環 (Step 2)

- 執行測試 (紅燈)
  - 未實作 API 內容
  - 確保測試程式是可被執行，並且預期結果是未通過的

TDD 循環 (Step 3)

- 實作「夠用」的產品代碼
  - 力求快速實作出功能邏輯，用「最低限度」通過測試案例即可
  - 無法快速實作出功能邏輯，代表使用情境過於龐大，應考慮拆細

TDD 循環 (Step 4)

- 再次執行測試 (綠燈)
  - 確保產品代碼是可被執行，並且符合預期結果
  - 此階段建立出一個功能邏輯正確的版本

TDD 循環 (Step 5)

- 重構程式
  - 消除重複設計，優化程式碼
  - 提升程式的可讀性、可維護性、擴充性

Tennis Kata Demo

需求探索

需求探索與設計

- 顯示有五種

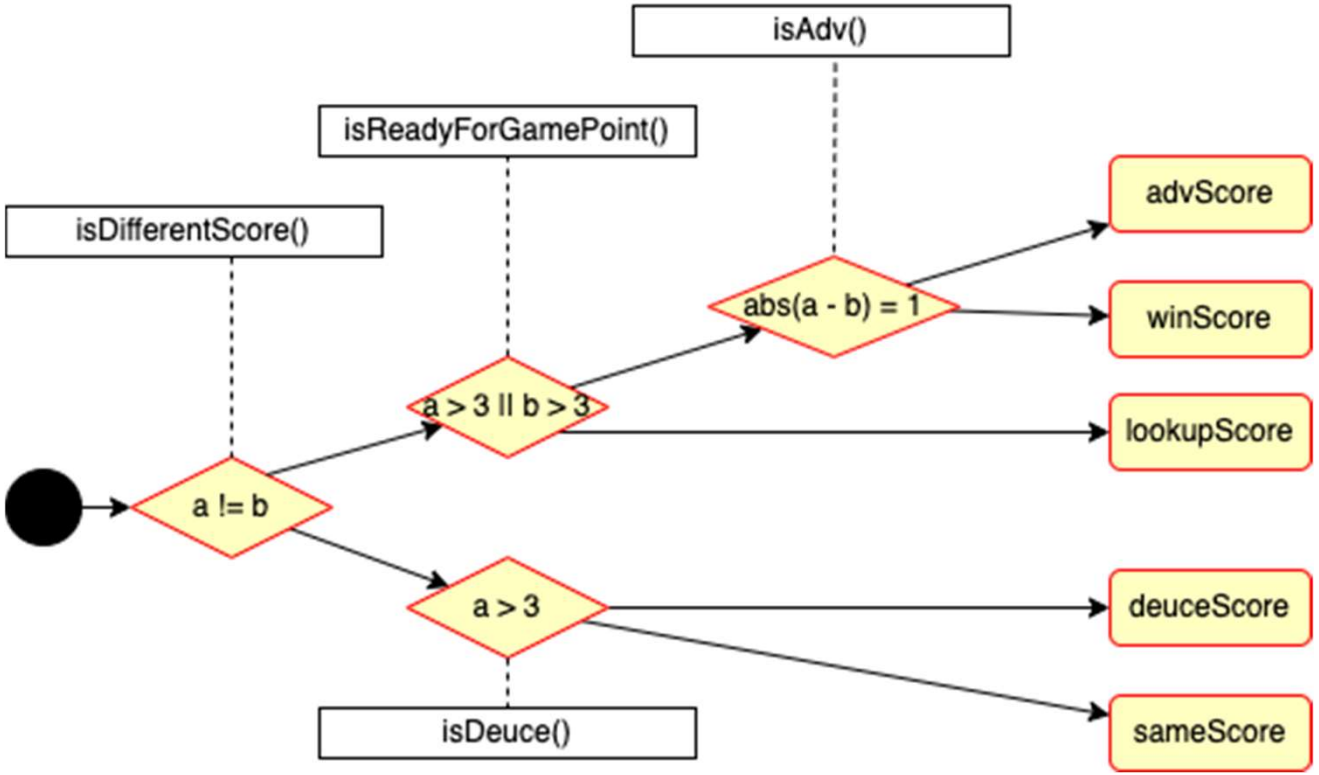
	顯示	程式設計
一般比分	thirty fifteen	lookupScore()
相同比分	love all	sameScore()
平手比分	deuce	deuceScore()
領先比分	tom adv	advScore()
獲勝比分	jerry win	winScore()

需求探索與設計

- 決策點

	程式設計
判斷比分相同	isSameScore()
判斷比分不同	isDifferentScore()
判斷平手	isDeuce()
判斷決勝點 (進入獲勝情境)	isReadyForGamePoint()
判斷領先	isAdv()
判斷獲勝	isWin()
取得領先者名字	advPlayer()

決策樹



測試案例

測試情境	預期結果	Production Code
0 : 0	love all	1. Tennis Class 2. add score()
1 : 0	fifteen love	1. add firstPlayerScore()
2 : 0	thirty love	1. use HashMap 2. lookuScore()
3 : 0	forty love	
0 : 1	love fifteen	1. add secondPlayerScore() 2. isDifferentScore() 3. sameScore()
0 : 2	love thirty	
~~0 : 3~~	~~love forty~~	

測試情境	預期結果	Production Code
1 : 1	fifteen all	
2 : 2	thirty all	
3 : 3	deuce	1. isDeuce() 2. deuceScore()
4 : 3	tom adv	1. Tennis("tom") 2. isReadyForGamePoint() 3. advScore()
3 : 4	jerry adv	1. Tennis("tom", "jerry") 2. isAdv() 3. advPlayer()
3 : 5	jerry win	1. winScore()
~~4 : 0~~	~~tom win~~	

困難點

- 需求分析與拆解
- 列出有效的測案與安排測試的順序
- 設計與建模，維持抽象思考
- 辨識壞味道與重構
- 打字速度與工具的使用

重構

大綱

- 重構定義
- 重構目的
- 重構時機
- 重構準則
- 壞味道
- Budget Demo

重構定義

- 名詞
  - 對軟體內部結構進行變動，目的是在「不改變軟體的可見行為」這個前提之下，提高它的可理解性，並降低修改它的成本
- 動詞
  - 在「不改變軟體可見行為」這個前提之下，用一系列的重構手法重新架構軟體

重構目的

- 改善軟體的設計，讓程式更容易理解
- 協助找出Bug
- 提升開發效率

重構時機

- 三次法則
- 需求異動
- 除錯 (Debug)
- Code Review

重構準則

- 自我測試
- 程式碼是穩定的
- 兩頂帽子
- 分辨壞味道

重構三部曲

- 整理代碼
  - 刪除 comment
  - 調整爛命名
  - 多餘空白行
  - Temp Variables
- 整理流程
  - 消除重複
  - 凸顯意圖
- 分職責
  - 物件導向 (高內聚)
  - Design Pattern (低耦合)

壞味道

Duplicated Code (重複的程式)

- 同樣的程式出現在不同的地方，代表有相同邏輯散落各處

Temp Variable（臨時變數)

- 有時候會定義一臨時變數來計算某些值，這個值或許只有在一個地方使用

Data Clumps (資料泥團)

- 總是一起出現的資料，經常玩在一起才具有意義，就像個泥團，這種情況會建議將它封裝一起

```
private String getFullAddress(String city, String area, String address) {
    return String.format("%s %s %s", city, area, address);
}

private String getFullAddress(Address address) {
    return String.format("%s %s %s", address.getCity(), address.getArea(), address.getAddress());
}
```

Feature Envy (依戀情結)

- 若函式對某個類別的興趣高過自己所處的 host 類別，這種情況會將其函式搬移

```
private String getFullAddress(Address address) {
    return String.format("%s %s %s", address.getCity(), address.getArea(), address.getAddress());
}

public class Address {
    // 略
    public String getFullAddress() {
        return String.format("%s %s %s", getCity(), getArea(), getAddress());
    }
```

Primitive Obsession (基本型別偏執)

- 他與 Data Clumps 類似，通常是會因為懶，不使用物件表達事實，放棄了物件導向「將行為與操作封裝再一起」
- String 型態也是是這味道的培養皿

Abstraction Distraction (抽象干擾)

- 在一個方法帶入一個抽象/介面作為參數，這個參數可能與這個方法的意義是不同的，那麼就容易被傳入的參數干擾。

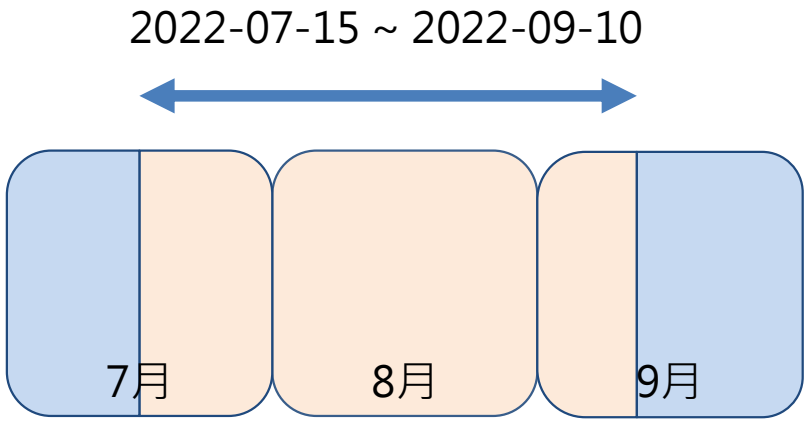
```
public void doEat(Car car) {
    car.driver.eat();
}

public void doEat(Driver driver) {
    driver.eat();
}
```

Budget Query Demo

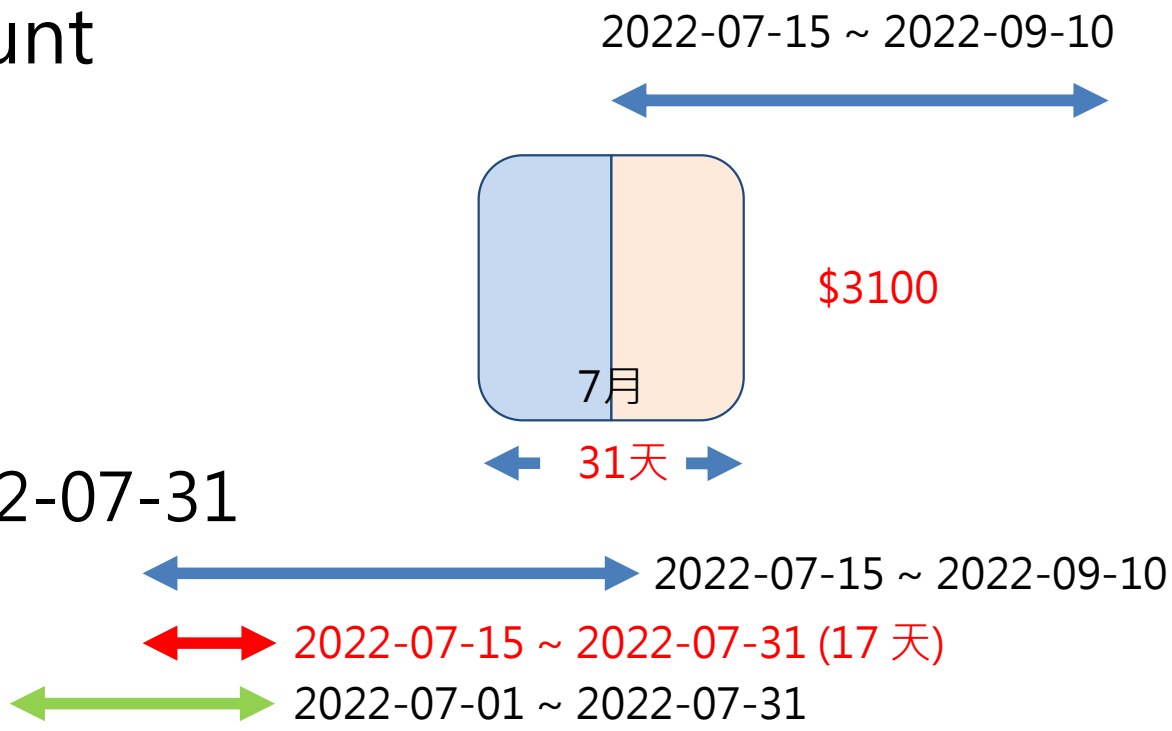
目標

月份	金額
2022-07	3100
2022-08	3100
2022-09	3000



目標

- Overlapping Amount
  - 100 * 17
- Daily Amount
  - 3100 / 31
- Overlapping Days
  - 2022-07-15 ~ 2022-07-31



目標

- Overlapping Amount
  - Daily Amount * Overlapping Days
- Daily Amount
  - Budget Amount / Budget YearMonth days
- Overlapping Days
  - Query 的期間與 Budget YearMonth 的重疊天數

測試案例

情境	起訖日	資料庫	預期結果
no_budgets	2022-08-27 ~ 2022-08-27	-	0
invalid_period	2022-08-27 ~ 2022-08-26	-	0
period_inside_budget	2022-08-27 ~ 2022-08-27	2022-08, 310	10
period_no_overlap_before_budget_first_day	2022-07-27 ~ 2022-07-27	2022-08, 310	0
period_no_overlap_after_budget_last_day	2022-09-01 ~ 2022-09-01	2022-08, 310	0
period_overlap_before_budget_first_day	2022-07-27 ~ 2022-08-02	2022-08, 310	20
period_overlap_after_budget_last_day	2022-08-31 ~ 2022-09-01	2022-08, 310	10
multiple_budgets	2022-07-30 ~ 2022-09-09	2022-07, 31 2022-08, 310 2022-09, 3000	1212

重構與設計

- 彼此為互補關係

Thanks for your listening.