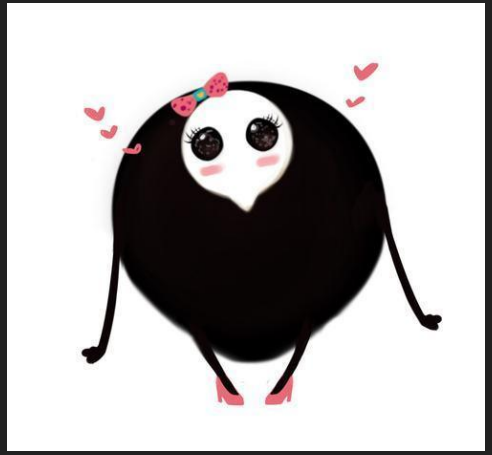
先別提測試了
你聽過 TDD 已死了嗎？

教授的一點軼事



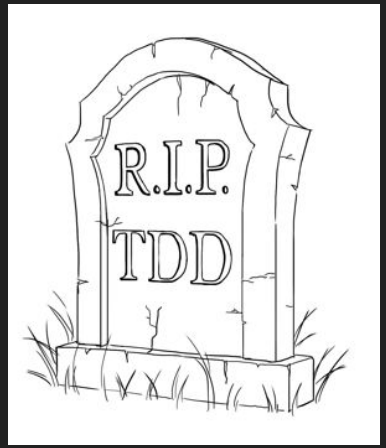
照片僅供參考，與實際人事物並無關連
如有雷同純屬巧合

- 什麼是測試驅動開發
- 測試驅動開發的好處
- 測試驅動開發的壞處
- 相關開發方式已死？
  - BDD
- 結論

TDD is Dead, Long Live Testing

David Heinemeier Hansson RailsConf 2014-04-23

又可以少學一樣東西啦！

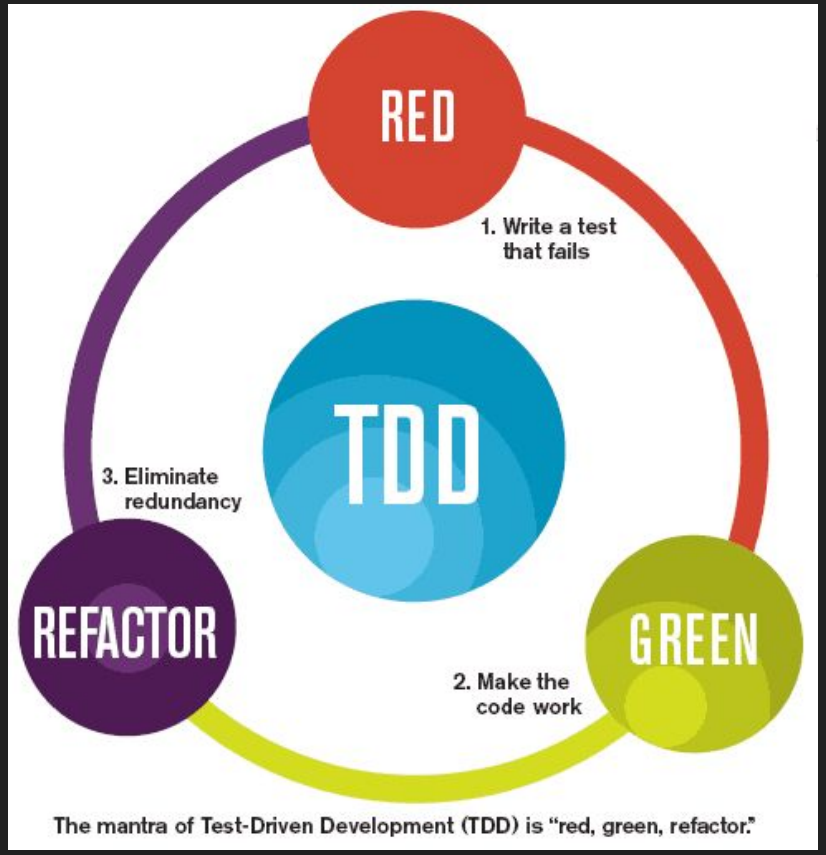

R.I.P.
TDD


我們懷念他

世界級筆戰（含 DHH，Uncle Bob……等人）

詳情可參考 91 懶人包

什麼是測試驅動開發



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

- 撰寫測試之前不寫功能
- 不撰寫過多測試
- 不撰寫過多功能

測試驅動開發的好處

Clean Code


Clean Code
無瑕的程式碼
Robert C. Martin
敏捷軟體開發技巧守則
A Handbook of
Agile Software Craftsmanship

可證偽的程式

可證偽的程式

每個元件功能都經過測試

可證偽的程式

每次修正錯誤都經過驗證

對自動測試的好處

對自動測試的好處

避免測試遺漏

對自動測試的好處

如果仔細來看，也許後寫測試還可以達到較高的覆蓋率。但是事後寫的測試只是一種防守，而先行編寫的測試則是進攻。

後寫的測試在深度和捕捉錯誤的靈敏度方面要遜色很多。

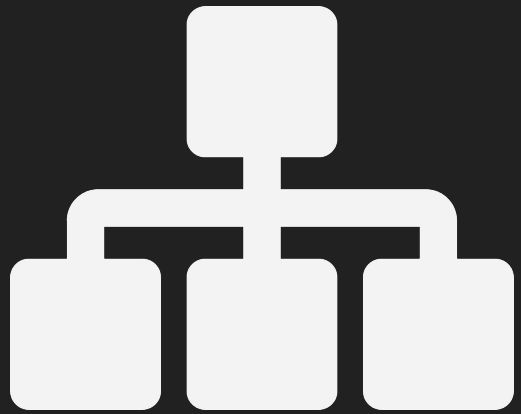
（The Clean Coder）

對架構的好處

對架構的好處

避免寫出難以測試的元件

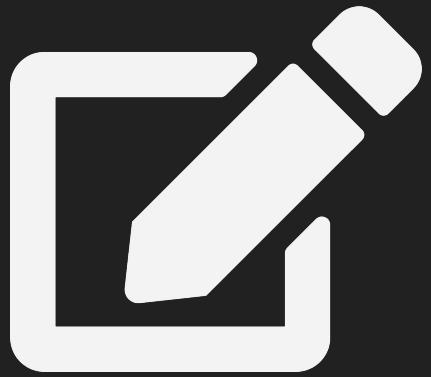
對架構的好處



協助降低物件大小 方便閱讀

對重構的好處

對重構的好處



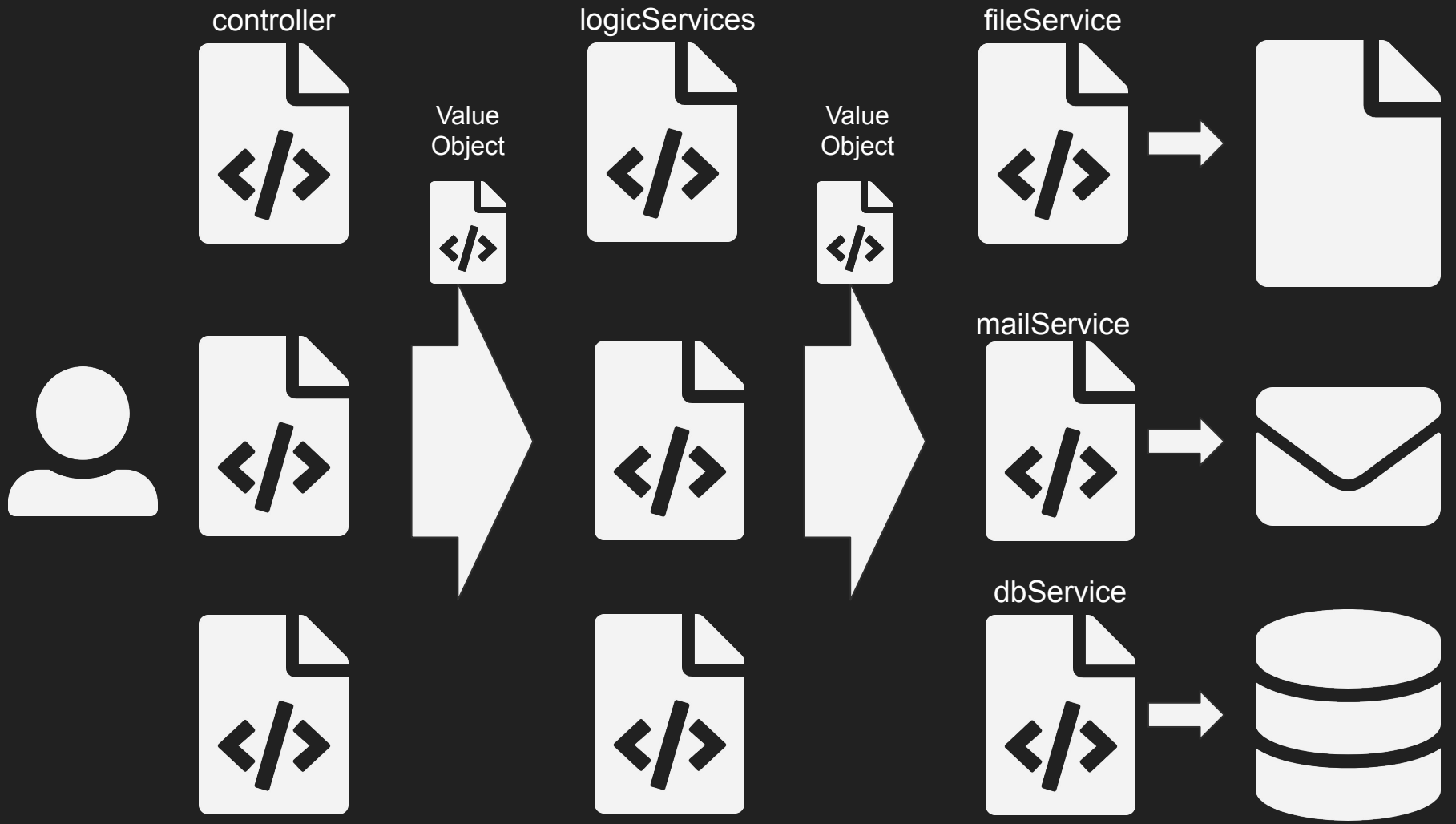
不會擔心重構影響其他功能

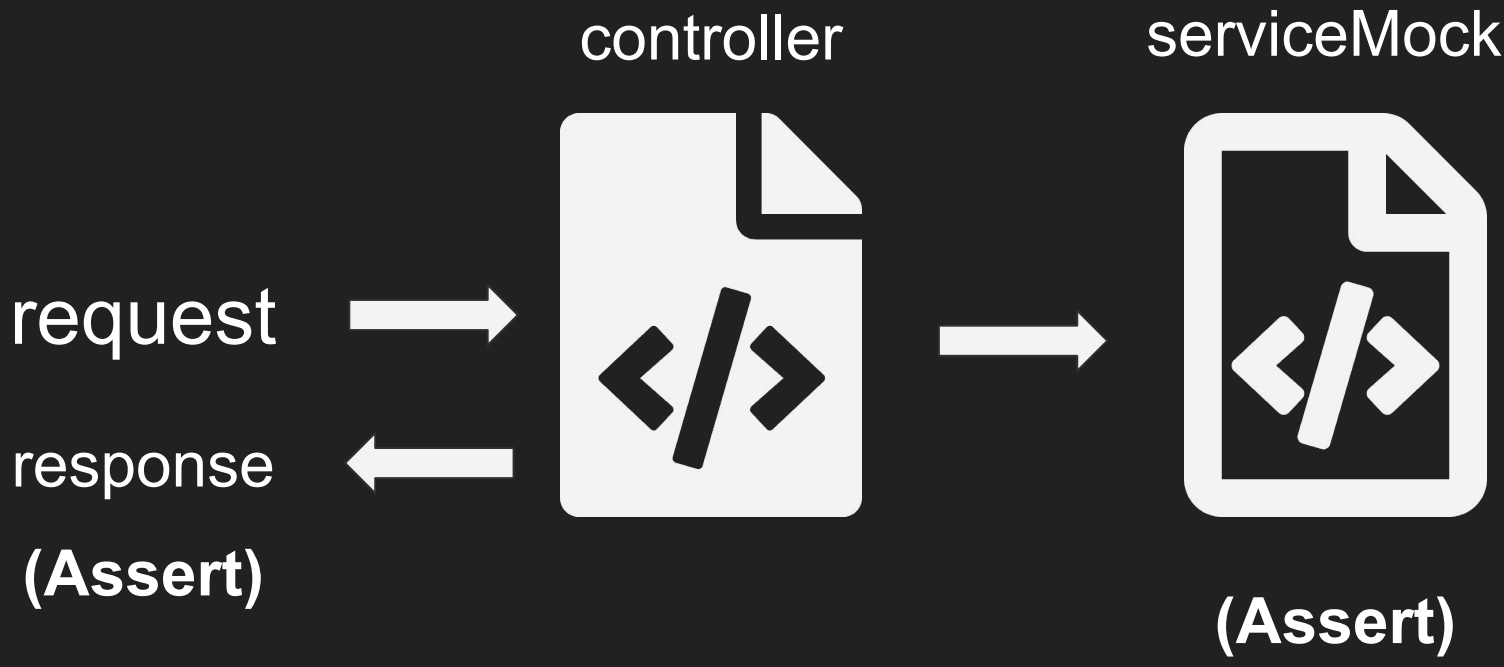
測試驅動開發的壞處

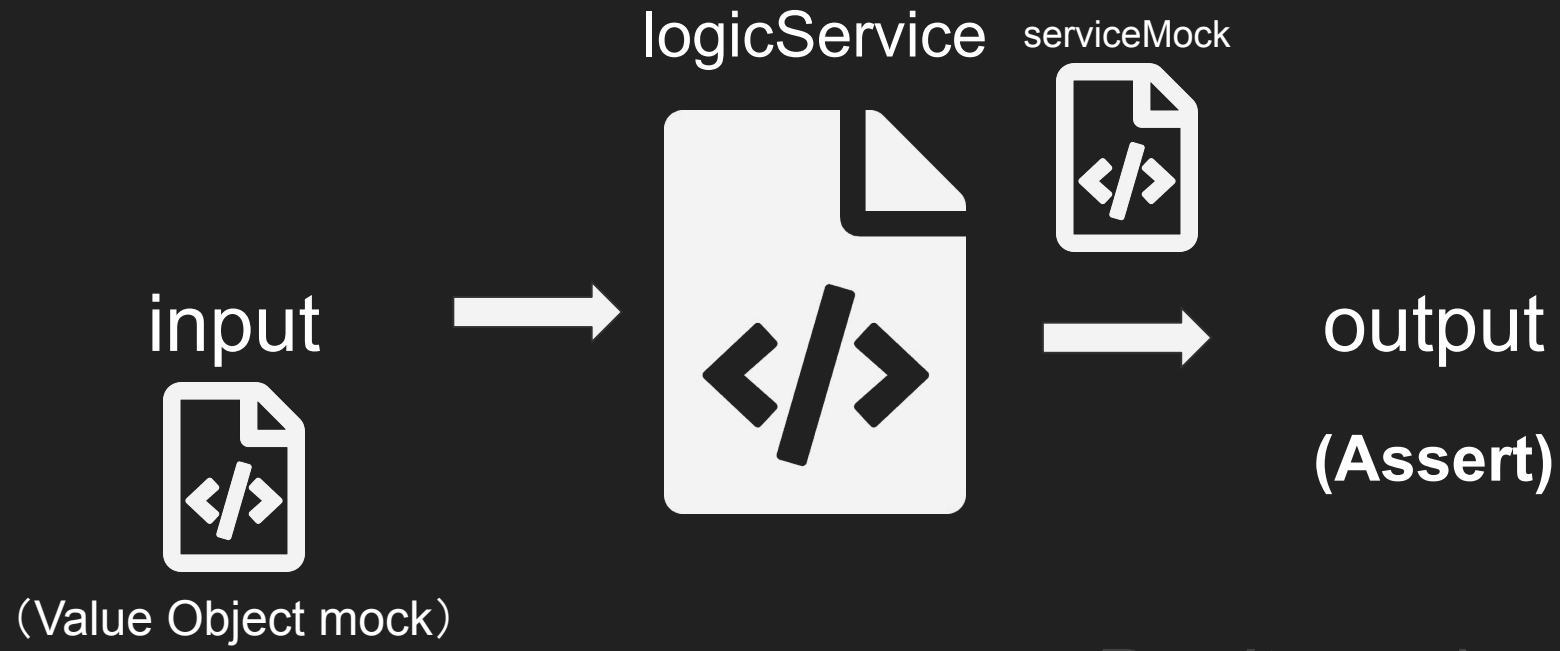
有人告訴你 TDD 有壞處

TDD is Dead, Long Live Testing

Test-first units leads to an overly complex web of intermediary objects and indirection in order to avoid doing anything that's "slow". Like hitting the database. Or file IO. Or going through the browser to test the whole system.


controller
Value Object
logicServices
Value Object
fileService
mailService
dbService

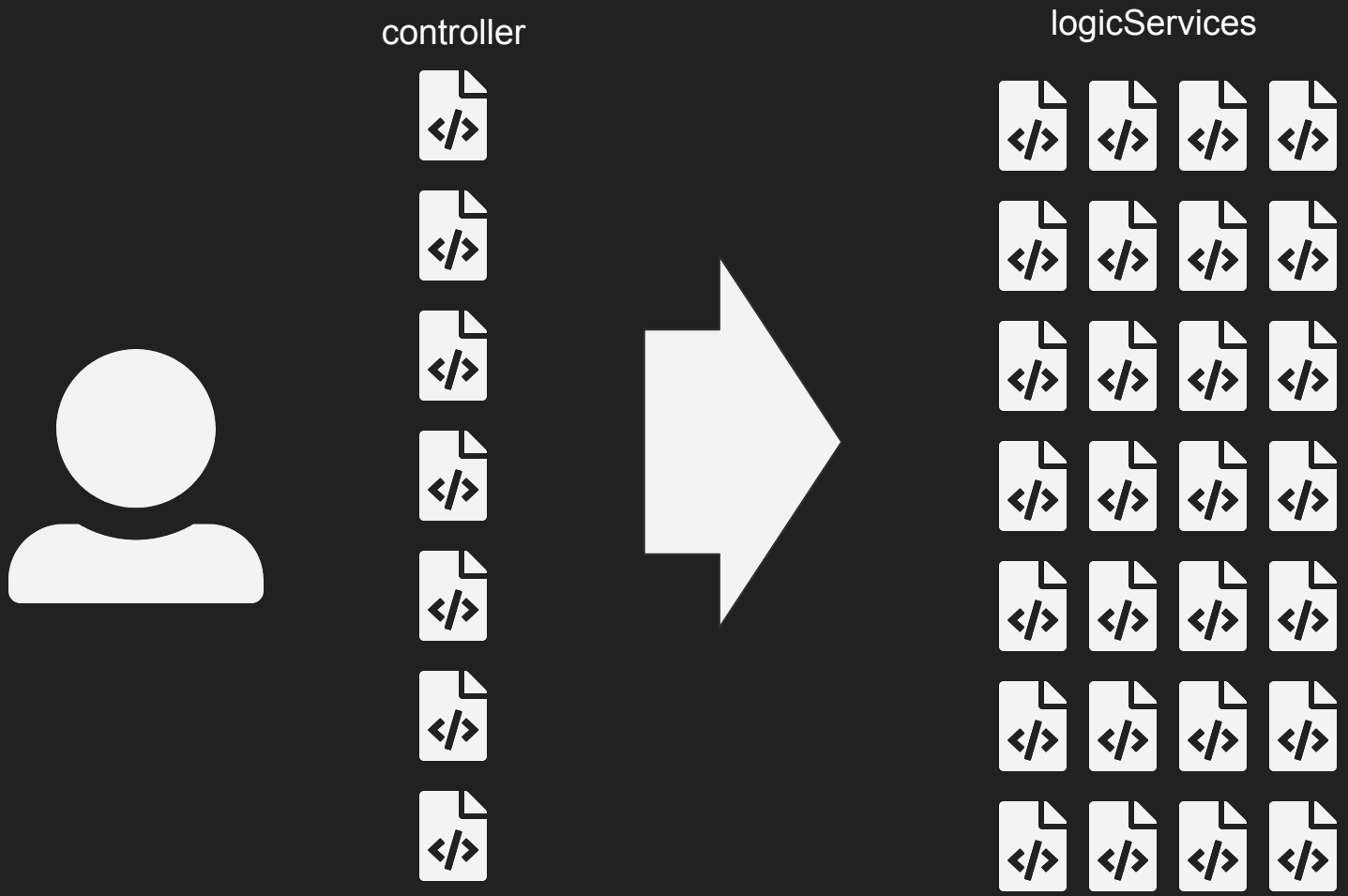

controller
serviceMock
request
response
(Assert)
(Assert)



Don't mock
value objects

TDD is Dead, Long Live Testing

(Test-first units) It's given birth to some truly horrendous monstrosities of architecture. A dense jungle of service objects, command patterns, and worse.

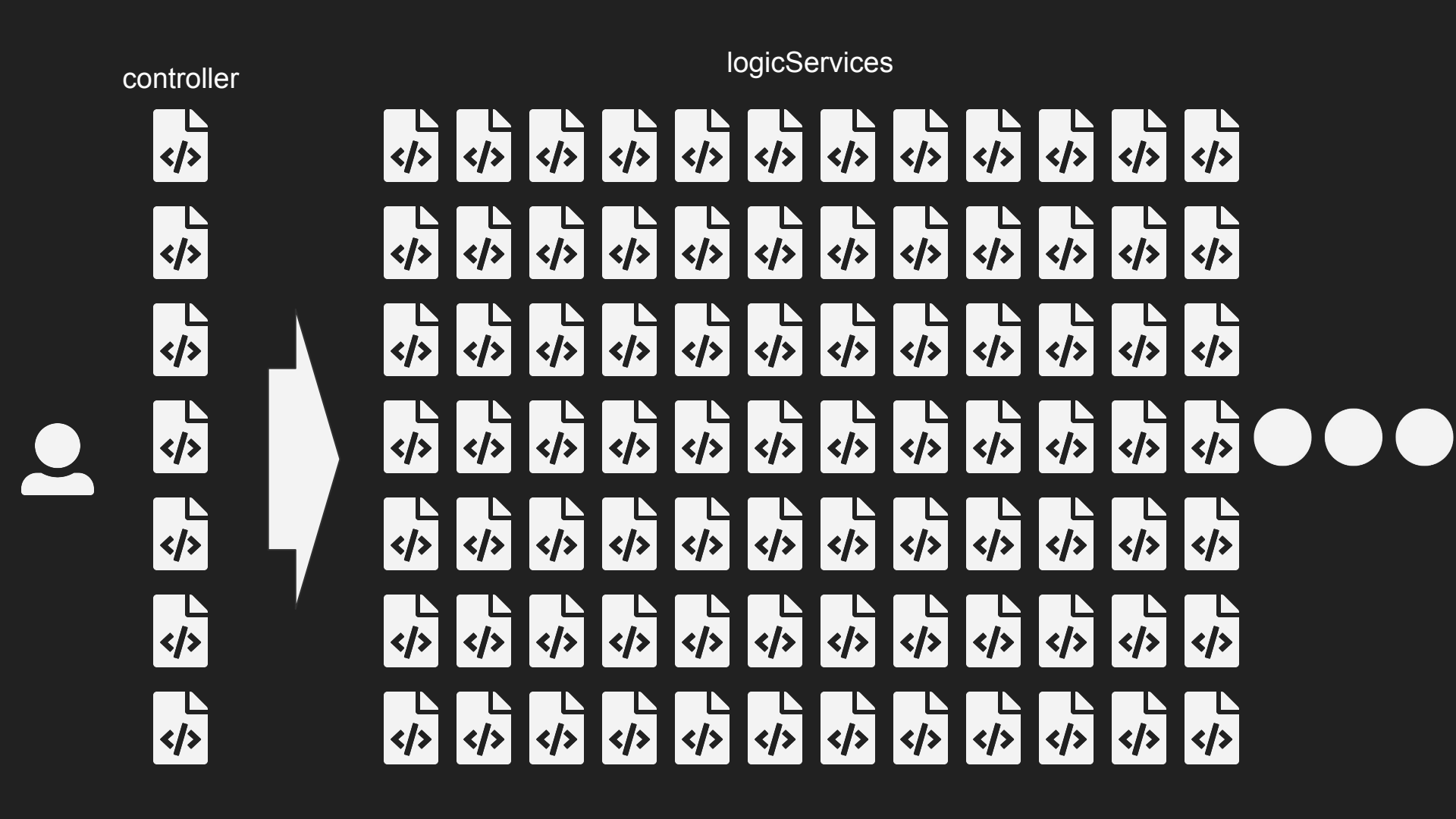

controller
logicServices

單一職責原則

＝每個類別只有一個職責

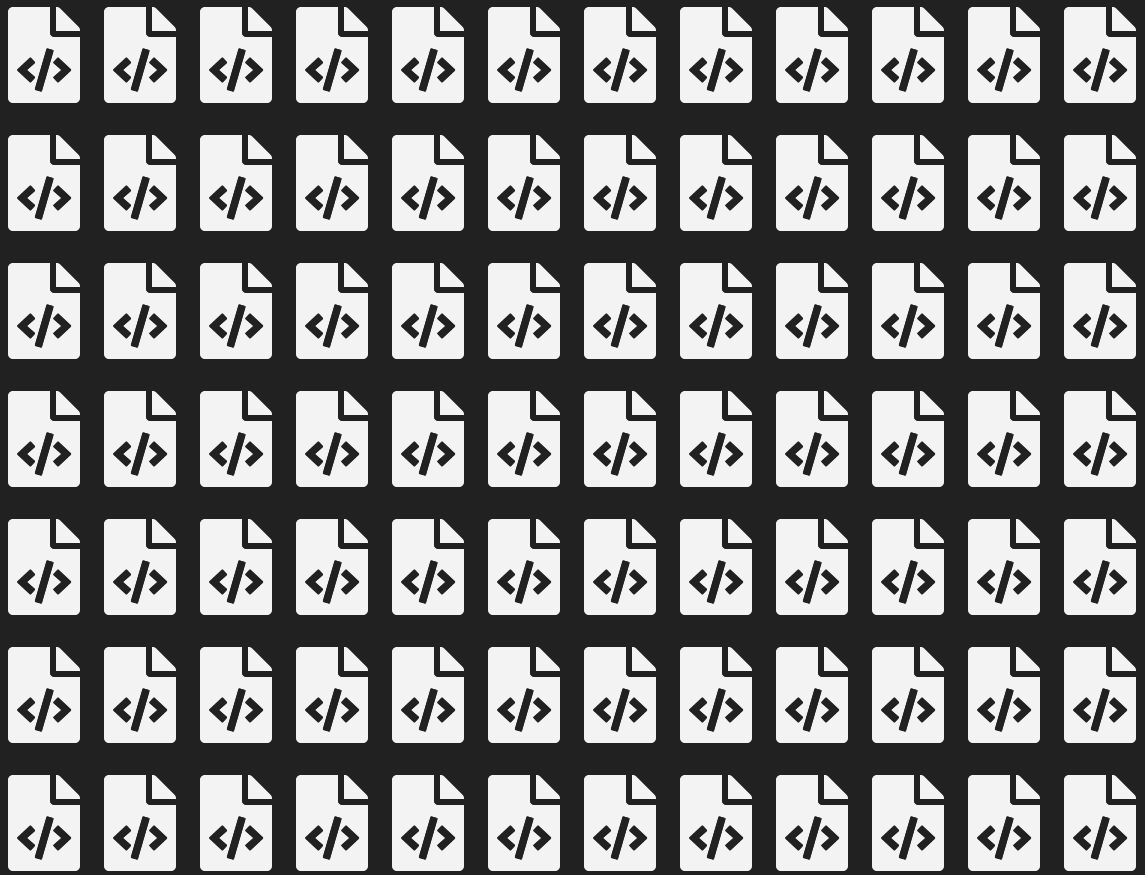
＝每個類別只有一個 **public function**

（Recca 不支持這個觀念）

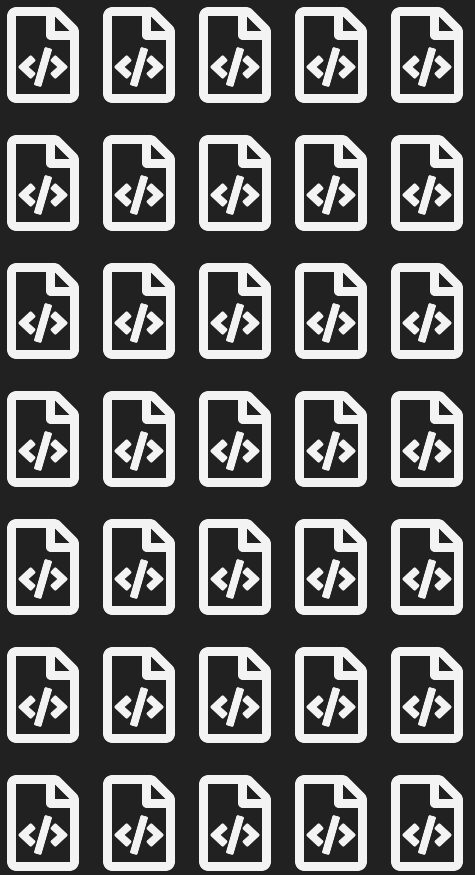

controller
logicServices

然後

logicServiceTests

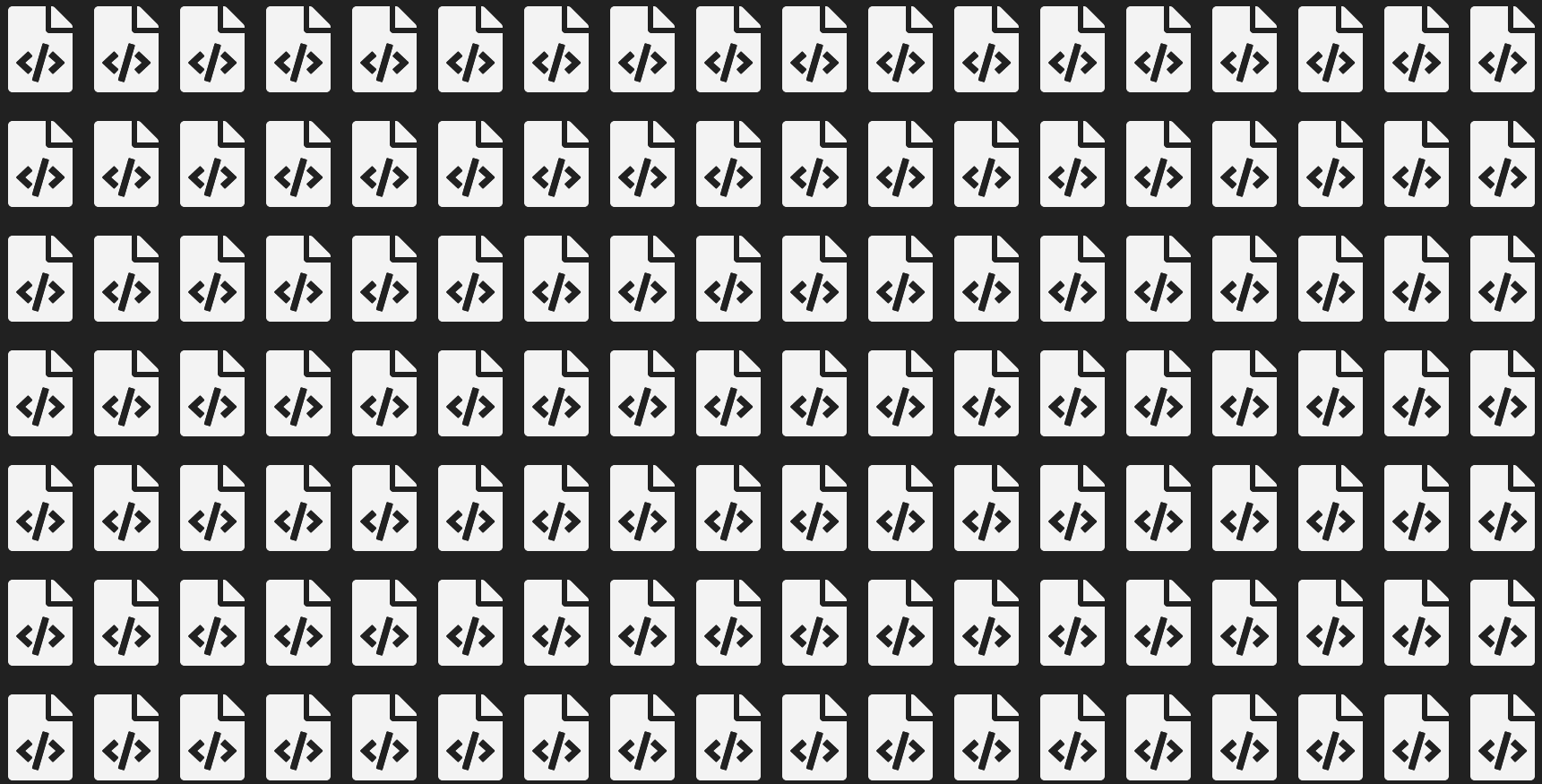


logicServiceMocks



如果你常用 command 協助測試

Commands



看起來 TDD 不好

When TDD doesn't work.(Uncle Bob)

複式記帳法（double-entry bookkeeping）

When TDD doesn't work.

- The physical boundary (bell)
- The layer just in front of that boundary（css）
- The test code itself


有人說某些場景不適用 TDD
imgflip.com

Anti Pattern 11: Treat TDD as a Religion

In summary, TDD is a good idea but you don’t have to follow it all the time.

If you work in a fortune 500 company, surrounded by business analysts and getting clear specs on what you need to implement, then TDD *might* be helpful.

需求極頻繁更動

測試剛寫好需求就變了

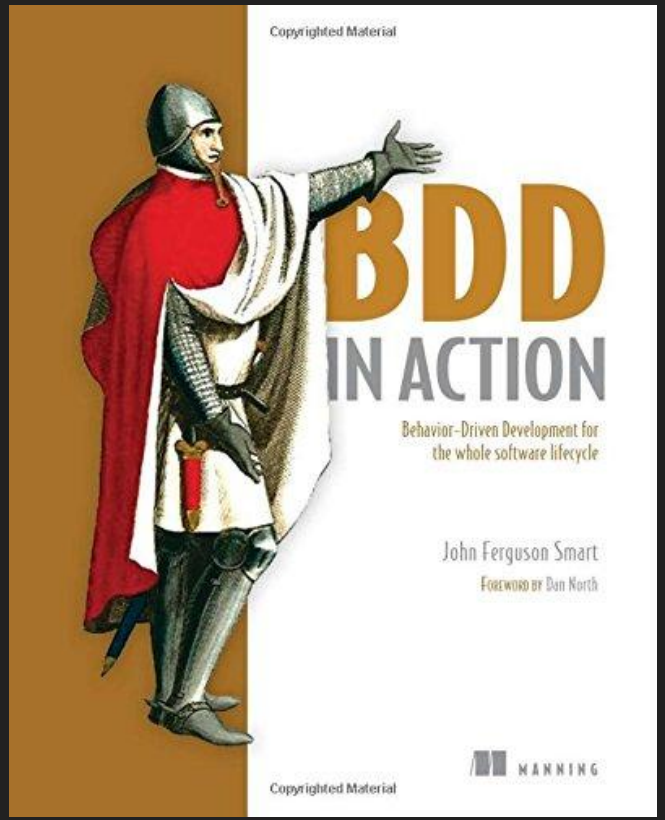
需求極頻繁更動

利用測試來保護需求更動

等等，那 BDD 呢？

什麼是 BDD

- 行為驅動開發（Behavior-driven development）
- Behavior-driven development combines the general techniques and principles of TDD with ideas from domain-driven design and object-oriented analysis and design to provide software development and management teams with shared tools and a shared process to collaborate on software development.


Copyrighted Material
BDD
IN ACTION
Behavior-Driven Development for the whole software lifecycle
John Ferguson Smart
Foreword by Dan North
MANNING
Copyrighted Material

composer require --dev behat/behat

Gherkin

- ~~醃黃瓜~~
- 描述功能，場景，階段的一種語言

Gherkin 範例：購物系統：功能

Feature: Product basket
  In order to buy products
  As a customer
  I need to be able to put interesting products into a basket

Gherkin 範例：購物系統：規則

Rules:

- VAT is 20%
- Delivery for basket under $10 is $3
- Delivery for basket over $10 is $2

Gherkin 範例：購物系統：場景 1

Scenario: Buying a single product under $10

Given there is a "Sith Lord Lightsaber", which costs $5

When I add the "Sith Lord Lightsaber" to the basket

Then I should have 1 product in the basket

And the overall basket price should be $9

（5 * 120% + 3 = 9）

商務行為說明／測試程式碼

將案例直接當作測試執行

```
$ vendor/bin/behat

3 scenarios (3 passed)
14 steps (14 passed)
```

和自動測試的關係

- 沒有自動測試保障的 BDD
  - 無法保證需求行為和程式行為相同！
  - 你寫你的商務行為，我寫我的程式
  - 井水不犯河水

和 TDD 的關係

- 你可以先寫測試，然後開發
- 你可以先寫開發，然後測試
- 嚴格來說，兩者都符合 BDD 的做法

再看一次這段話

如果仔細來看，也許後寫測試還可以達到較高的覆蓋率。但是事後寫的測試只是一種防守，而先行編寫的測試則是進攻。

後寫的測試在深度和捕捉錯誤的靈敏度方面要遜色很多。

（The Clean Coder）

有沒有延伸的問題

結論

可以避免 TDD 容易出現的問題嗎？

善用隔離框架

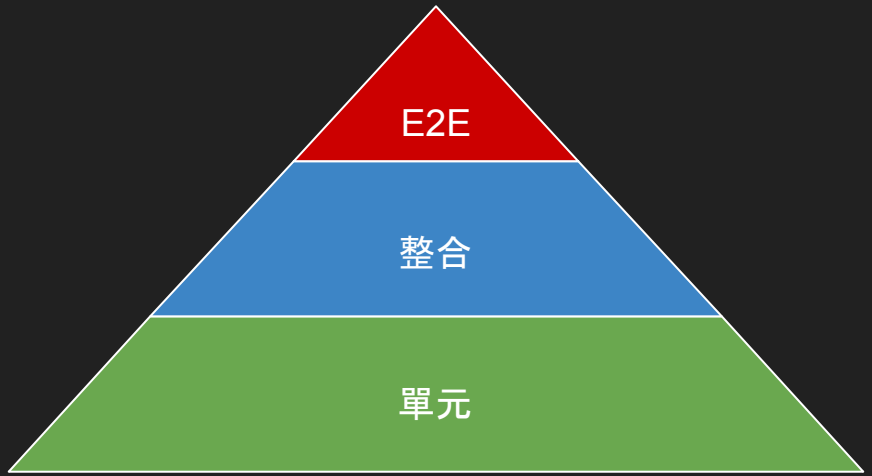
Mockery

- Mockery::*mock*()
- Mockery::*spy*()

Laravel

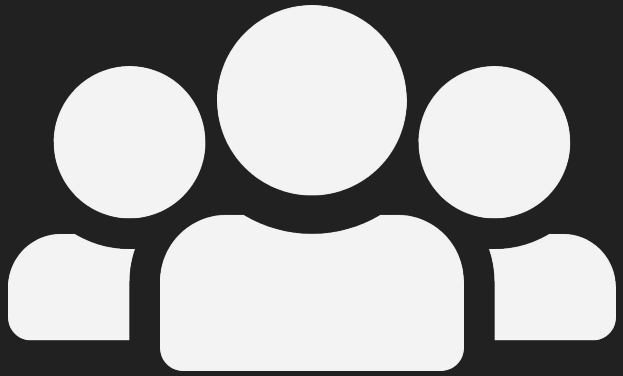
- Mail::*fake*();
- Queue::*fake*();
- Storage::*fake*();

測試金字塔



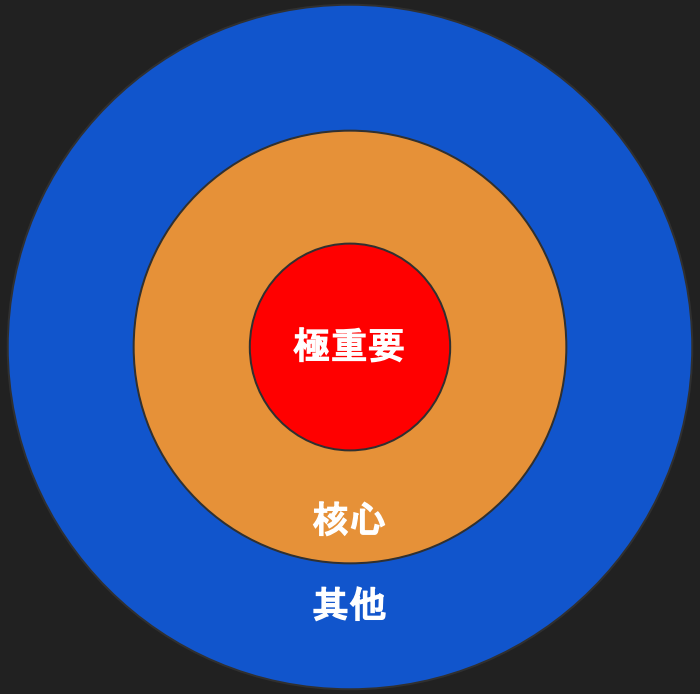
世界上有不需要雙方帳本的交易？

產品的生命週期



開發團隊的狀況

還不會走之前 不要學飛



Trying to get 100% coverage on ***total*** code is not recommended.

CTM（Codepipes Testing Metrics）

名稱	簡介	標準值	通常值
PDWT	撰寫測試的開發人員比例	100%	20% ~ 70%
PBCNT	能產生新測試的臭蟲比例	100%	0% ~ 5%
PTVB	驗證行為（而非實作）的測試比例	100%	10%
PTD	決定性（不會無故出錯）的測試比例	100%	50% ~ 80%

測試驅動開發 ≠ 自動測試

放棄測試驅動開發 ≠ 不寫自動測試

避免已知問題

- 程式是否粒度過細？
- E2E／整合／單元測試比例分配是否恰當？
- 是否有不必要的測試？

守破離

- ~~大安區懷石料理~~
- 學習的三個階段
  - 守
    - 遵守指導，熟練基礎教條的 內容
  - 破
    - 打破規則，知道原本教條的缺失
  - 離
    - 脫離規範，不再被教條 內容束縛


No Silver Bullets

選用適合當下場景的工具

正確理解現況 討論 決策


NO EXCUSES!

tl;dr

- 測試驅動開發的好處
  - 避免自動測試有遺漏
  - 協助撰寫功能時設計成容易自動測試
- 拒絕測試驅動開發的理由
  - 容易寫出過多層次的架構
  - 容易測試原本不需測試的部分
  - 更動需求時需多維護測試部分
- 沒有銀彈
  - 審慎判斷需求
  - 不要找藉口或偷懶

套件程式討論團

@ReccaChaoWebDev

投影片網址

找到投影片的問題了嗎？


谢谢大家~