SOLID Principles

Hubert (@yhchan)

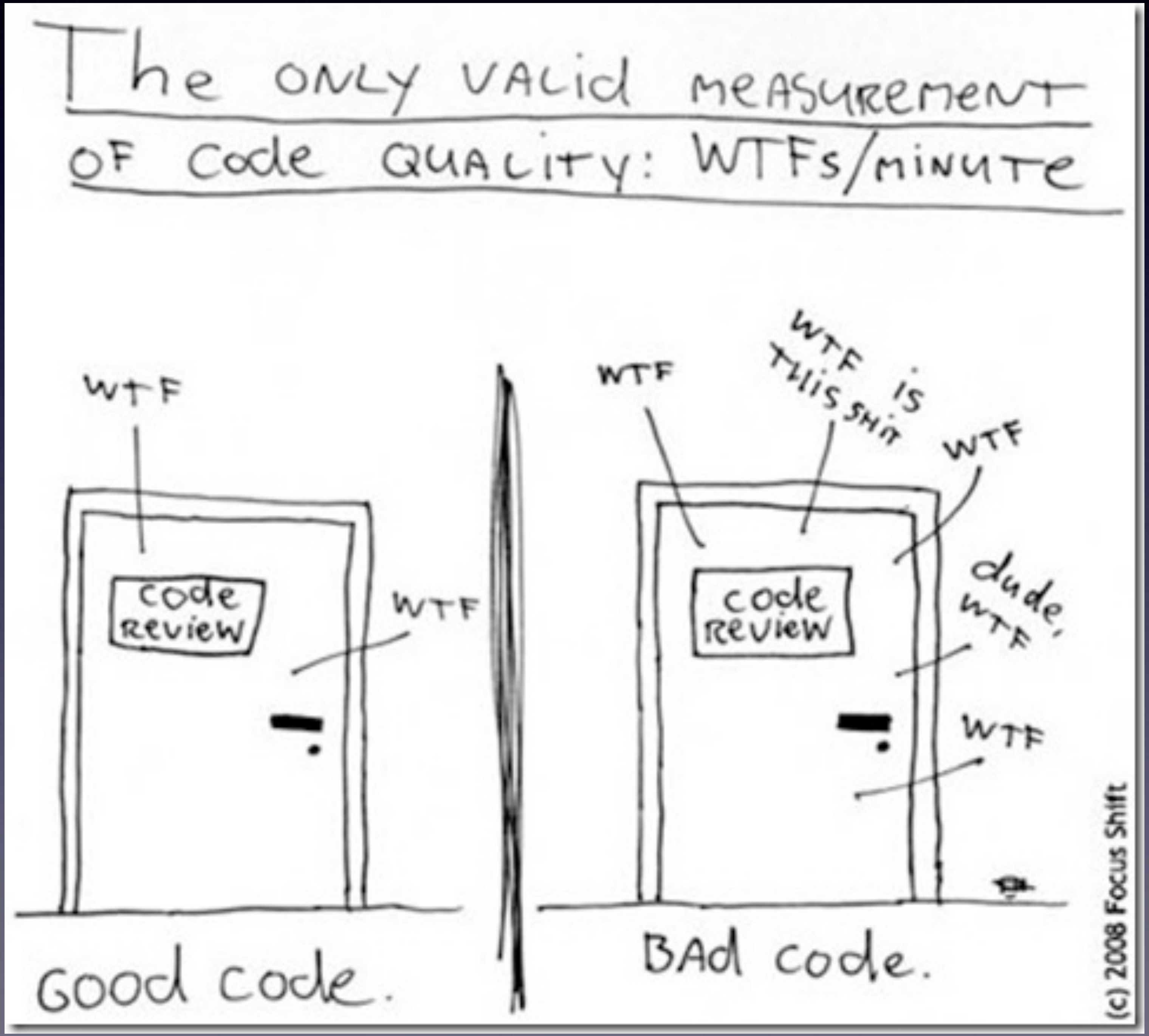
關於我

- Hubert (@yhchan)
- 趨勢科技
- Python Programmer
- http://blog.hubert.tw

好的程式碼？

- Readability
- Maintainability
- Reusability

Code Quality



完美的程式碼
Big Design Up Front

完美 v.s. 夠好

一次做好 v.s 改得動

原本今天要說
Design Pattern

比 Design Pattern 更重要的事情

- Maintainability
- Principles
- Refactoring
- Unit Testing
- Testability

據說只能講 30 分鐘
講完天都黑了…

SOLID Principles

- Single Responsibility Principle (SRP)
- Open-Closed Principle (OCP)
- Liskov substitution principle (LSV)
- Interface segregation principle (ISP)
- Dependency inversion principle (DIP)

Single Responsibility Principle

- 每個 Class 只有「一個」職責
- 如果需要修改，只能因為這個職責而修改

三個權責

```
class Employee
{
    public function calculatePay() {}   // Business Logic?
    public function save() {}           // Schema change?
    public function reportHours() {}    // Format string?
}
```

https://www.flickr.com/photos/pennuja/5363518281

SRP 的好處

- Unit Test 單純化
- 減少 Code Bloat 的機會
    - 一個 class 1000 行？
- Code Reuse
- 維護性

一輩子做好一件事
就功德圓滿

Keep it simple stupid (KISS)

Open Close Principle

- Open for extension, closed for modification
- 不用修改原本的 code，就可以增加新功能！
- 增加功能的手段
  - subclass and override
  - strategy injection
  - template pattern

OCP Violation

```
class Rectangle
{
    public $width;
    public $height;
}

class AreaCalculator
{
    public function area(array $shapes) {
        $area = 0;
        foreach ($shapes as &$shape) {
            if ($shape instanceof Rectangle) {
                $area += $shape->width * $shape->height;
            }
        }
        return $area;
    }
}
```

OPEN CLOSED PRINCIPLE

Open Chest Surgery Is Not Needed When Putting On A Coat

Shotgun Surgery

- Bad Smells
  - 增加一個 feature，要改非常多零碎的地方
  - 過份使用 if / else 作為增加 feature 的手段

Interface and override

```
interface Area
{
    public function area();
}

class Rectangle implements Area
{
    public $width;
    public $height;

    public function area() {
        return $this->width * $this->height;
    }
}

class AreaCalculator
{
    public function area(array $shapes) {
        $area = 0;
        foreach ($shapes as &$shape) {
                $area += $shape->area();
        }
        return $area;
    }
}
```

Adapter Connections

```php
class FacebookAdapter
{
    private $pool;

    public function doAPI($params) {
        // Make $request from $params
        $this->sendRequest($request);
    }

    private function sendRequest($request) {
        $pool = $this->makeConnectionPool();
        $pool->doRequest($request);
    }

    private function makeConnectionPool() {
        if ($this->pool) {
            return $this->pool;
        }
        // connection pool implementation
    }
}
```

Connection Pool
不用嗎？

Google / Twitter Adapter?

使用 Mixin 或 Abstract class

```php
trait AdapterMixin
{
    private $pool;

    private function sendRequest($request) {
        $pool = $this->makeConnectionPool();
        $pool->doRequest($request);
    }

    private function makeConnectionPool() {
        if ($this->pool) {
            return $this->pool;
        }
        // connection pool implementation
    }
}

class FacebookAdapter
{
    use AdapterMixin;
    public function doAPI($params) {
        // Make $request from $params
        $this->sendRequest($request);
    }
}
```

Once and Only Once

Bug 修一個地方就好

Template Method

```php
abstract class RESTAdapter {
     abstract protected function createConnection();

     protected function parseResponse() {
         // Default implementation
     }
     public function apiRequest($uri, $body) {
        $conn = $this->createConnection();
        $resp = $conn->request($uri, $body);
        return $resp;
     }
}

class HTTPRestAdatper extends RESTAdapter {
    protected function createConnection() {
        // create HTTP connection
    }
}

class HTTPSRestAdatper extends RESTAdapter {
    protected function createConnection() {
        // create HTTPS connection
    }
}
```

流程固定
Override 不同

Don’t Repeat Yourself (DRY)

Cohesion 凝聚力

多少相關連的
member data / function

Liskov Substitution Principle

- 定義
  - 使用父類別的 reference 或 pointer 操作 function 時，不需要知道實際指向的類別
  - Design by Contract
    - 父類別跟子類別的「行為」
    - Function Prototype

Liskov Substitution Principle

```php
interface PersistentResource
{
    public function load();
}

class PostData implements PersistentResource
{
    // ...
}

class AuthData implements PersistentResource
{
    // ...
}

$postData = new PostData();
$loadPostData = $postData->load();

$authData = new AuthData();
$loadAuthData = $authData->load();
```

Design by Contract

- Design by Contract
  - 講定的介面與行為
  - Interface / Function Prototype
  - 行為
    - pre-condition
    - post-condition
    - Side Effects

LSP Violation

```
interface PersistentResource
{
    public function load();
}

class PostData implements PersistentResource
{
    public function load()
    {
        return $this->db->load('post_data');
    }
}

class AuthData implements PersistentResource
{
    public function load()
    {
        $authData = $this->db->load('auth_data');
        $this->authenticator->auth($authData);
        return $authData;
    }
}
```

Design by Contract?

- 沒有預期的 Side Effect
- 來看看之前的範例
  - Pre-condition
    - `$this->db` is ready
  - Post-condition
    - 把資料讀取出來
  - Unexpected Side Effect
    - Authentication

Function Prototype

- Function Prototype

```
class Foo
{
    public function bar(array $bar) {}
}

class Baz extends Foo
{
    public function bar(array $bar, array $beer) {}
}

Strict Standards: Declaration of Baz::bar() should be compatible
with Foo::bar(array $bar)
```

Explicit is better than implicit

– PEP20 The Zen of Python

明確比隱誨更好

Interface Segregation Protocol

- Interface 的使用者不該被強迫使用自己不需要的 function
- 把介面切割乾淨
- 不該是我的，就不要放進來，想想 SRP

ISP Violation

```
interface Bird
{
    public function fly();
    public function run();
}


class Dove implements Bird
{
    public function fly() {}
    public function run() {}
}

class Penguin implements Bird
{
    public function fly() {
        // Ahhh, I cannot fly
    }
    public function run() {}
}
```

Java 與 ISP

- Java 的介面設計
  - Serializable
  - AuthClosable
  - Readable
  - DataOutput

Dependency Inversion Principle

- Principle
    - 依賴抽象類別，而非實體類型
    - 低耦合 decoupling
- 好處
    - Open Close Principle
    - 擴充性 / 測試性 / Reuse

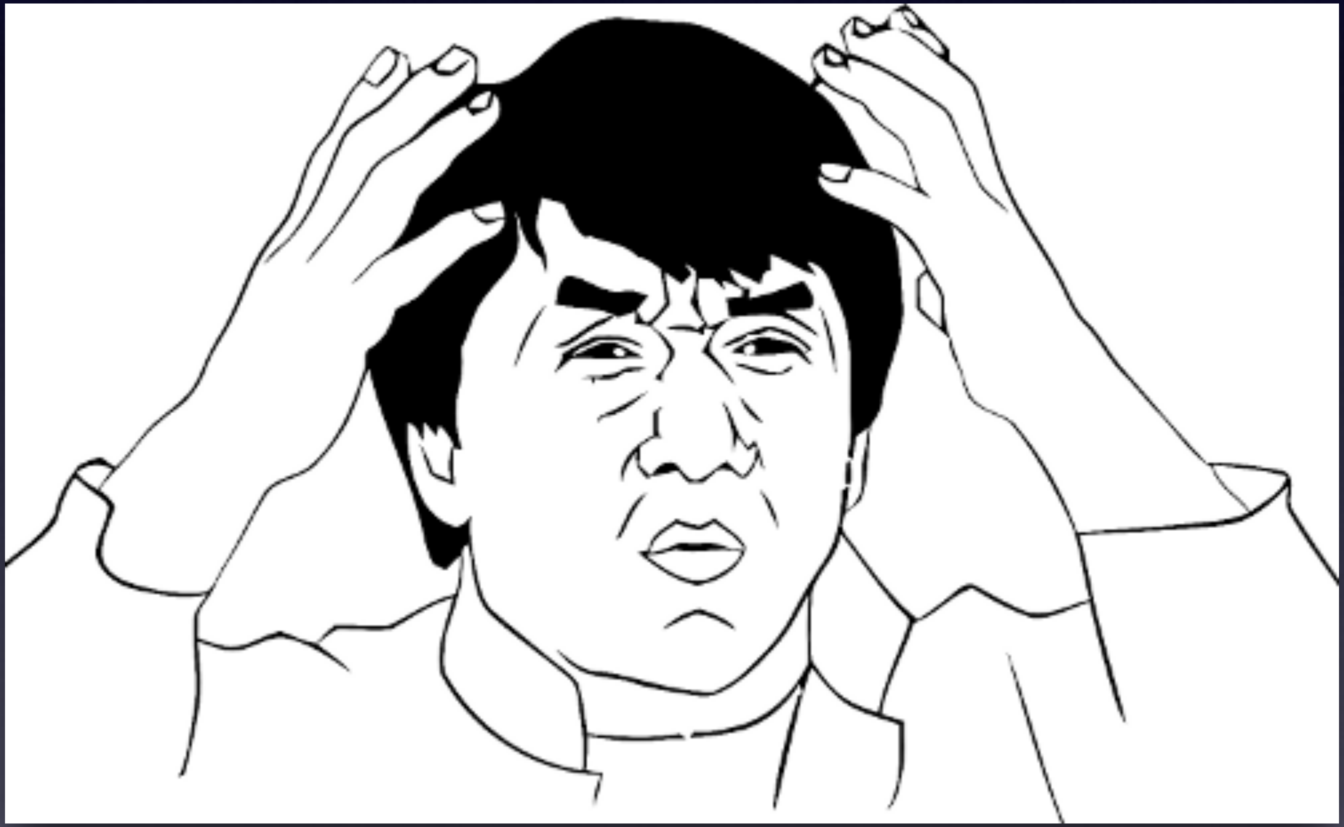
DIP Violation

- Singleton 不好測試

```php
class UserRepository {
    const BY_USERNAME_SQL = "Select ...";

    public function loadUser($user) {
        $db = Database::getInstance();
        return $db->query(self::BY_USERNAME_SQL, $user);
    }
}
```

老闆說這邊想放在Redis？



DIP Example

- Use ISP in practice

```
class UserRepository {
    const BY_USERNAME_SQL = "Select ...";

    private $_db = null;

    function __construct($db) {
        $this->_db = $db;
    }

    public function loadUser($user) {
        return $this->_db->query(self::BY_USERNAME_SQL, user);
    }
}
```

從測試的角度出發

```
class UserRepository {
    const BY_USERNAME_SQL = "Select ...";

    private $_db = null;

    function __construct($db) {
        $this->_db = $db;
    }
    public function loadUser($user) {
        return $this->_db->query(self::BY_USERNAME_SQL, $user);
    }
}

class UserRepositoryTest extends PHPUnit_Framework_TestCase {
    public function testLoadUser() {
        $stubDB = $this->getMock('FakeDB');
        $stubDB->expects($this->any())
               ->method('query')
               ->will($this->returnValue(array('name' => 'hubert')));
        $repo = new UserRepository($stubDB);
        $repo->loadUser('hubert');
    }
}
```

DIP Patterns

- Dependency Injection
  - Constructor injection
  - Setter injection
- Service Locator
  - IoC Framework

Is TDD Dead?

- 我自己的看法
  - 透過測試來改善軟體架構
  - 幫助你思考 SOLID 跟責任分離
  - 先有 SOLID，然後 Design Pattern
  - TDD 不是萬能
  - 為了測試而測試？

Tell, Don’t Ask

- 結帳的時候，你會把自己交給櫃臺嗎？

```
class Clerk {
    Store store;
    void SellGoodsTo(Client client) {
        money = client.GetWallet().GetMoney();
        store.ReceiveMoney(money);
    }
};
```

Tell, Don’t Ask

- 媽媽，我要錢，只要你的錢

```
class Clerk {
    Store store;
    void SellGoodsTo(money) {
        store.ReceiveMoney(money);
    }
};
```

Law of Demeter

- Law of Demeter 建議，物件 O 中的函示 m 只能使用哪些物件
    1. O 自己
    2. m 的 parameter
    3. m 當中 new 出來的物件
    4. O 所能直接存取的成員函示及物件
    5. 全域變數
- Law of Demeter

```
final String outputDir =
ctxt.getOptions().getScratchDir().getAbsolutePath();
```

良好的封裝

結語

- SOLID 是物件導向的基本
- 因為有限制，才得以出類拔萃
- 好的設計 / 好的測試
- 物件的封裝與權責
- Design Pattern

Q & A

Global Variables?

```
int printf(
  const char *format, ...);




int main() {
  printf("Hello world.\n");
  return 0;
}
```

```
int fprintf(
  FILE* stream,
  HEAP* heap,
  int* errno,
  const char *format, ...);

int main(GLOBALS *globals)
{
  int errno = 0;
  fprintf(
    globals->stdio->out,
    globals->heap,
    &errno,
   "Hello world.\n”);
}
```

https://docs.google.com/presentation/d/17Hg3-42fygPd_JUJHorjs1Ah2KCYnjBUBEQJGZgpT6E/edit?usp=sharing