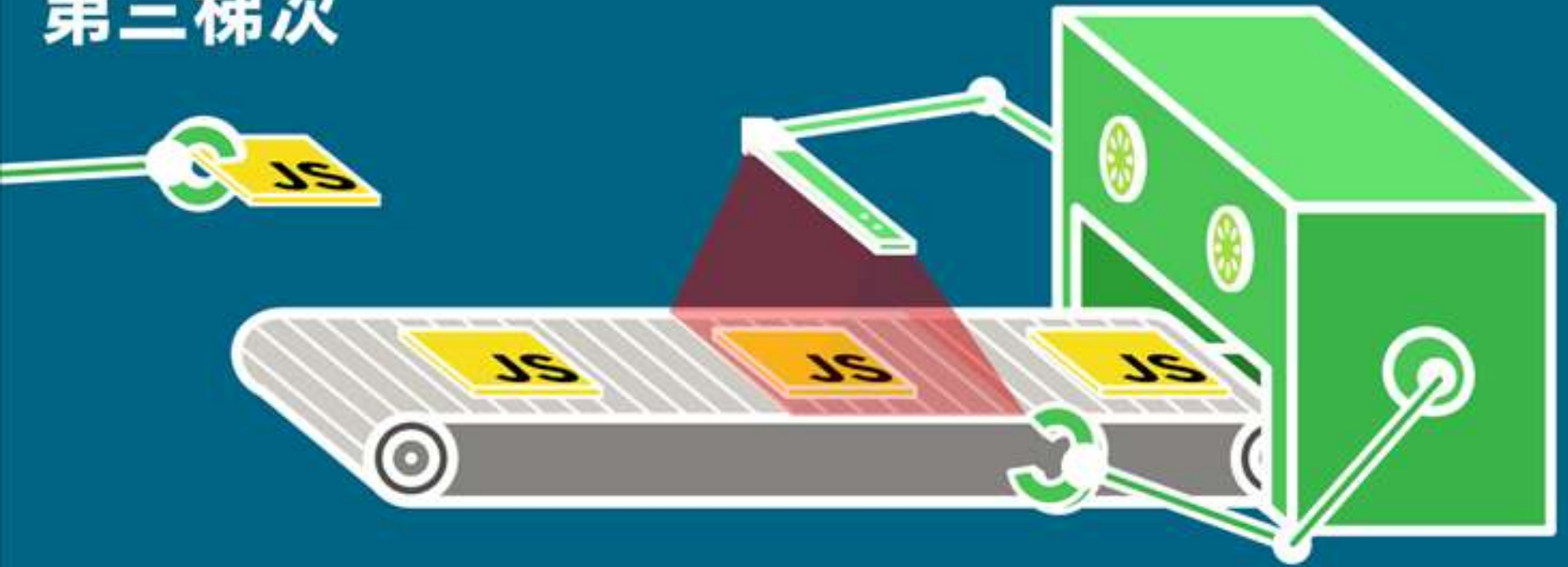


JavaScript 實務測試新手班

第三梯次



 SKILLTREE

2017/09/16 - 2017/09/23 14H

陳鋒逸 (陳小風)

自我介紹

- 陳鋒逸 (陳小風)
- 講師經歷
 - 微軟最有價值專家 (MVP)
 - SkillTree 兼任講師
 - 社群研討會講師
 - AgileCommunity.tw
 - Javascript.tw
 - twMVC



粉絲團：愛流浪的小風



現代化網頁開發

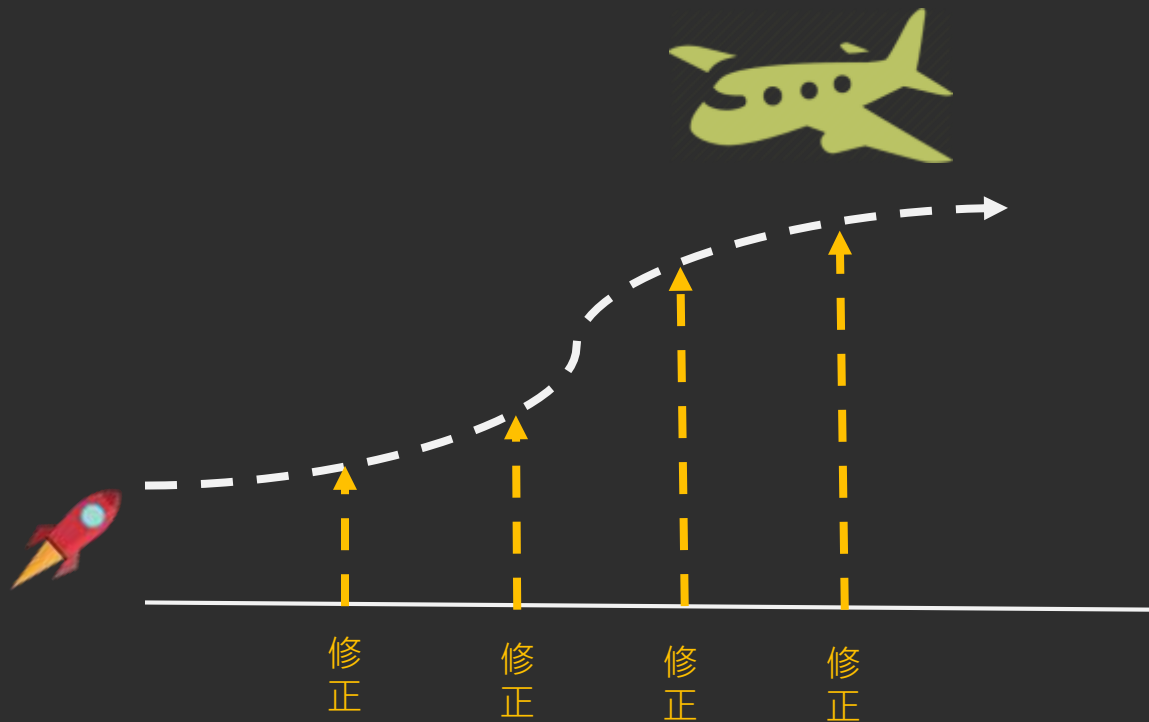
Software in 2017

- 趨勢變化
- 競爭激烈
- 時間就是金錢
- 市場難以預測



適應變化

- 關注目標變化
- 持續修正方向



產品團隊的挑戰

持續交付新產品



```
graph TD; A[持續交付新產品] --> B[保持系統穩定]; B --> C[瞭解服務狀態]; C --> D[應付意外狀況];
```

保持系統穩定

瞭解服務狀態

應付意外狀況

持續交付的痛點



穩定性

相容性

可回復性

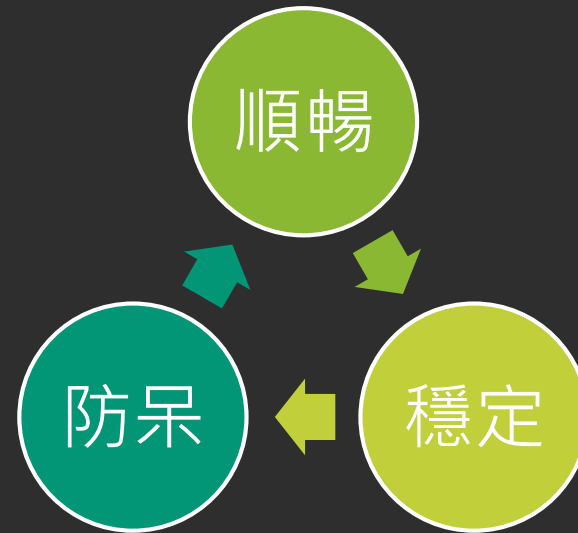
應變性

測試的目的

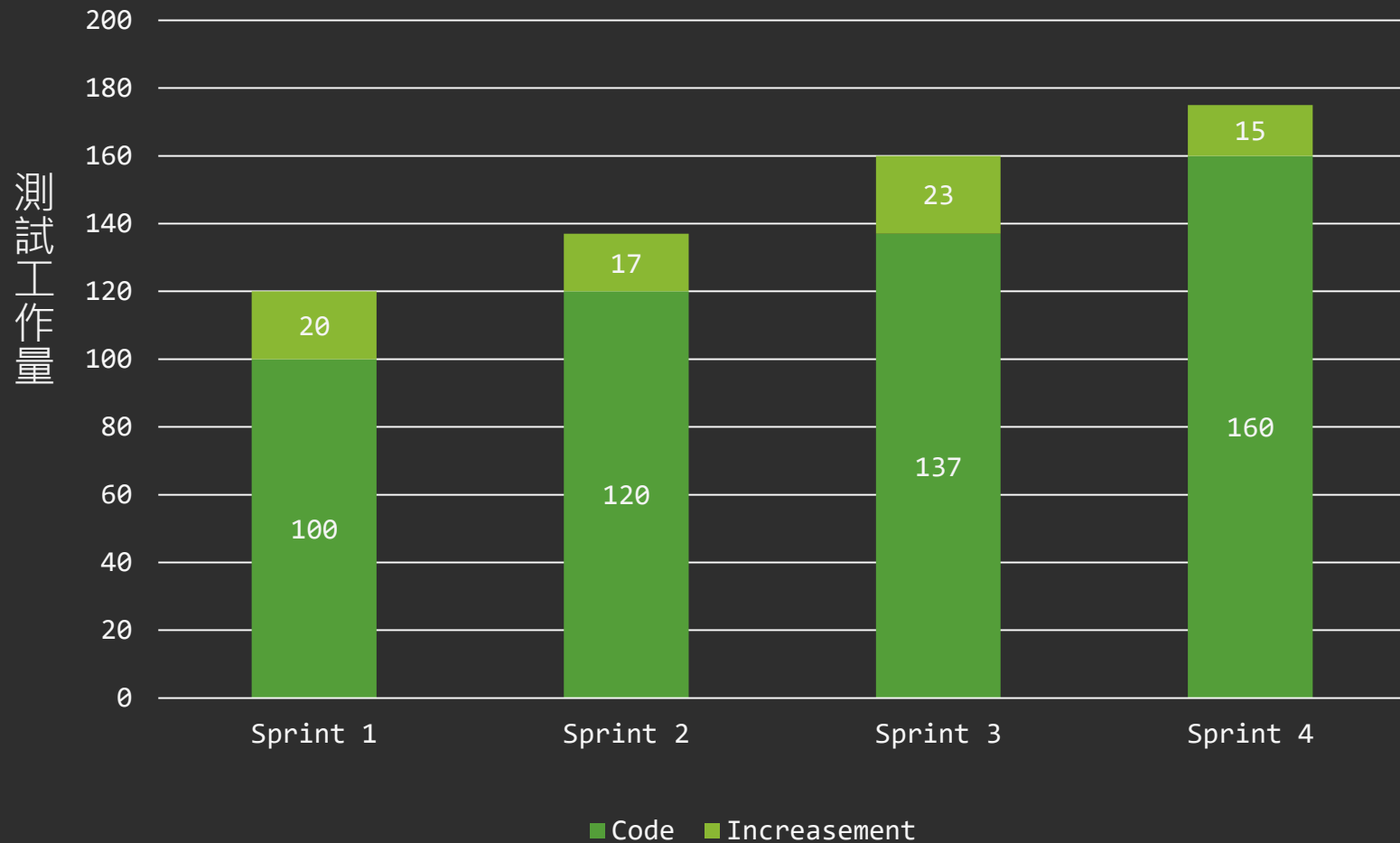
- 驗證結果符合預期
- 找出潛在問題
- 安全防護網
- 有重構的機會

你做的是測試嗎？

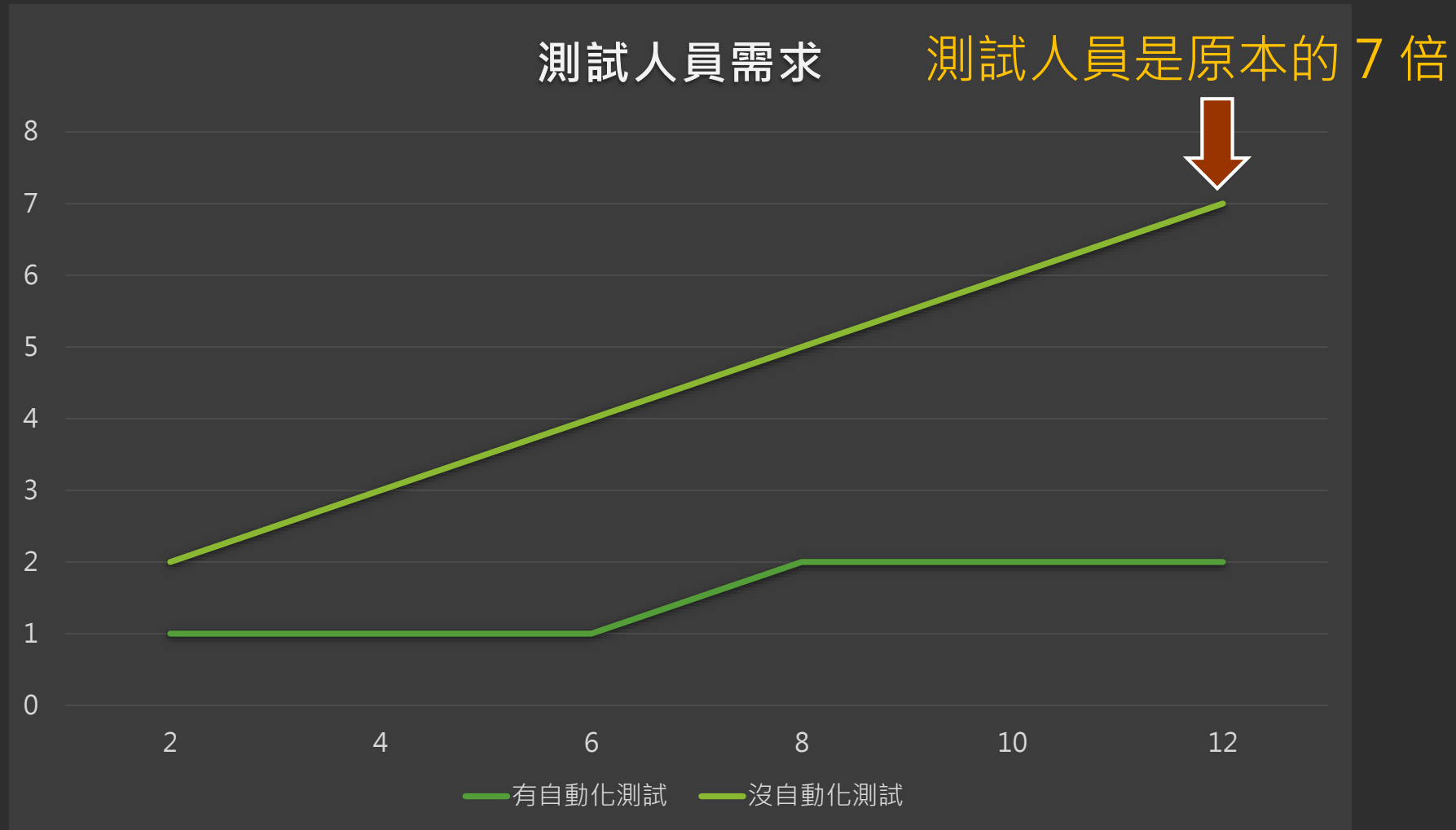
- 預期會發生的行為 => 驗證
- 找出意料外的行為 => 測試
- 讓 驗證 自動化



程式碼行數與測試需求



為什麼需要自動化測試



惡性循環

開發時間少

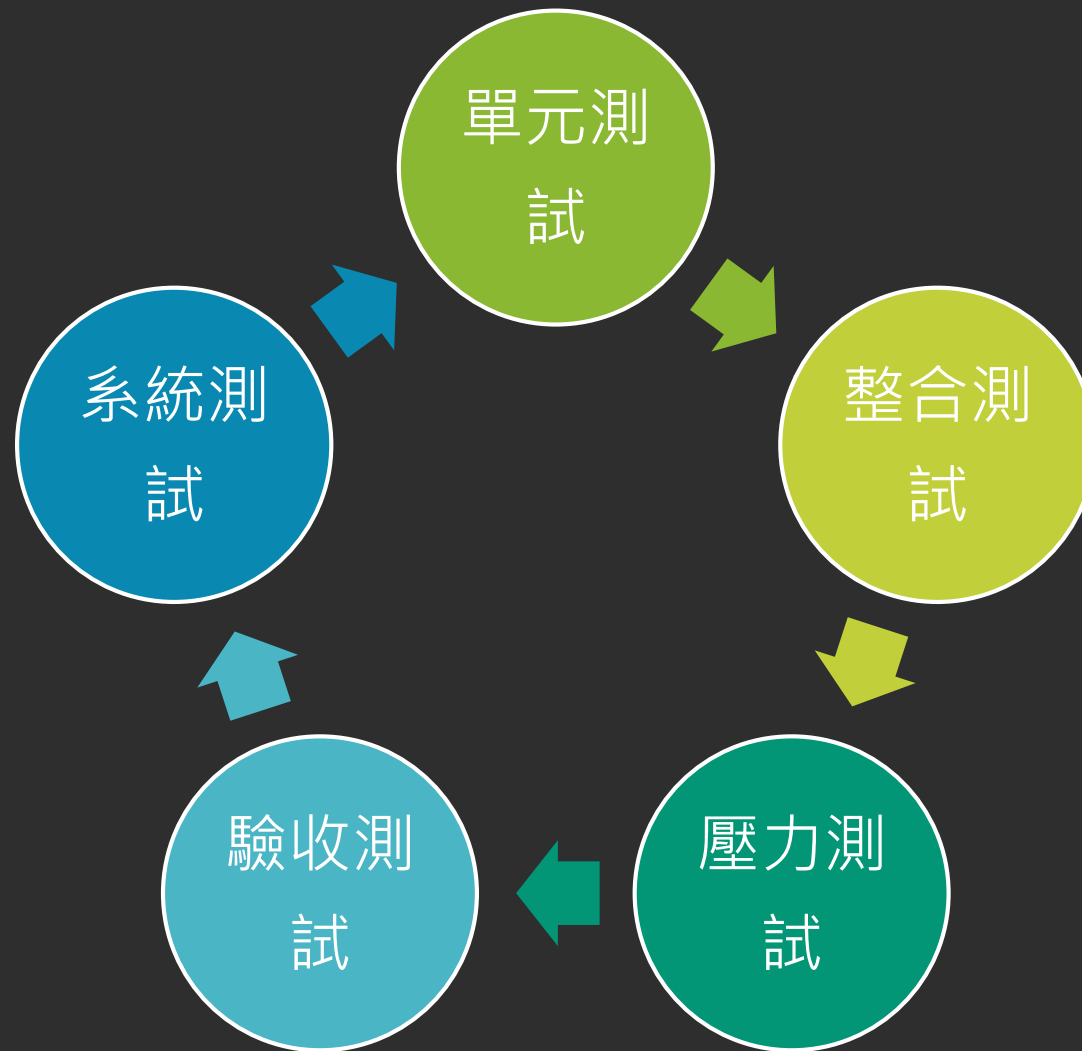
測試問題多

加班趕工上線

發現程式很難改

投入更多資源

讓測試自動化



單元測試

單元測試

- 測試的最小方法
- 沒有相依性
- 一個測試一個方法
- 一次只做一件事情

SOLID

- **Single Responsibility** – 單一職責原則
一個 **Function** 只負責一件事情
需要多個職責時，使用 **組合**
- **Open/ Close** – 開放封閉原則
對擴展開放，對修改封閉

SOLID

- **Liskov Substitution** – 里式替換原則
使用父類的地方，必須可以使用子類替代
- **Interface Segregation** – 介面隔離原則
Class 之間的依賴應該建立在最小的介面上

SOLID

- **Dependency Inversion** – 依賴反轉原則
透過抽象介面依賴

FIRST

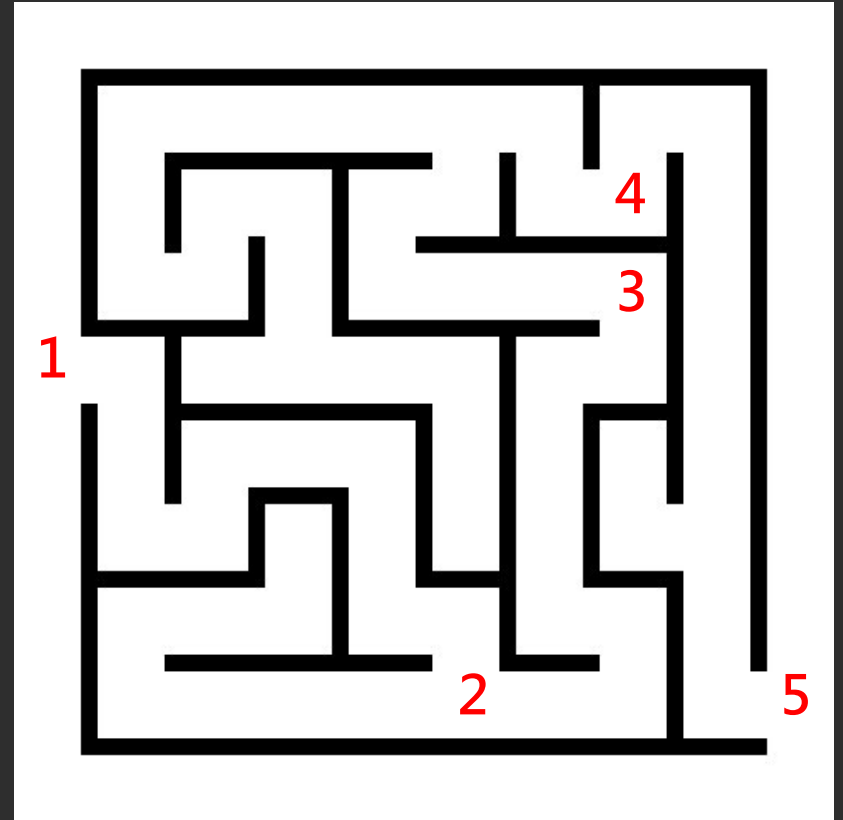
- FAST
- INDEPENDENT
- REPEATABLE
- SELF VALIDATION
- TIMELY

要測試什麼?

- 如何使用這個方法
- 方法會提供什麼樣的功能
- 方法具有什麼樣的行為
- 有哪些可能的例外狀況

TDD

- 設定目標
- 容易達成
- 進度透明
- 不容易迷路
- 釐清思路
- 有效交付



撰寫第一個測試

- 使用 `mocha.js`
- 設定使用 `BDD` 模式

```
mocha.setup('bdd')
```

- 執行測試

```
mocha.run();
```

- 使用指令建立

```
mocha init [directory]
```

- 練習
 - 練習撰寫第一個測試程式

小結

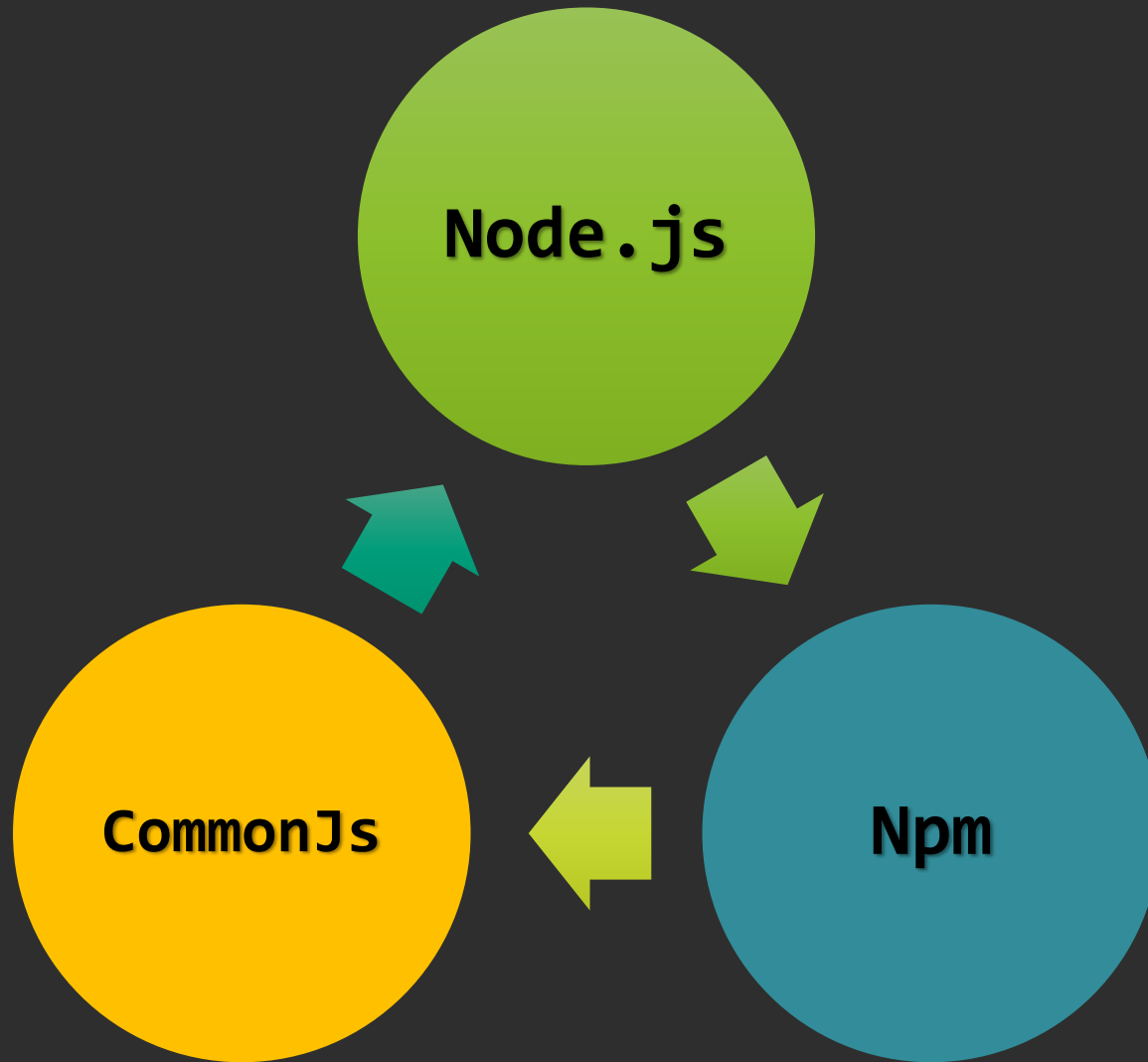
- 單元測試
- TDD

讓開發更順手

前端開發的困難

- 使用的套件很多
- 套件版本不一致
- 建立環境需花時間
- 沒有統一的開發流程
- 需要打包、壓縮檔案

前端工具箱



Node.js



- 基於 Google 的 **V8** 引擎開發
- 讓 JavaScript 可以在 **Server Side** 執行
- 可以用來 打包、壓縮程式碼
- 可以用來**預編譯** React.js
- 執行單元測試、整合測試



NPM (Node Package Manager)

- NPM 是 node.js 下所提供的 套件管理工具
- 方便我們 安裝 第三方 Library
- 可控管我們程式所需要的套件版本
- 可以透過 package.json 還原 專案所需的套件

管理套件



npm install



常用的NPM指令



- 初始化專案

```
npm init
```

- 安裝套件

```
npm install ( --save or --save-dev )
```

- 安裝的套件清單

```
npm list
```

- 搜尋套件

```
npm search
```

- 練習
 - 練習使用 `node.js` 初始化開發環境

模組化

修改程式碼時

```
var DatePicker = React.createClass({
  componentDidMount: function () {
    var datePicker = React.findDOMNode(this.refs.datePicker);
    var $datePicker = $(datePicker);

    $datePicker.datepicker();
  },
  render: function () {
    return <input type="text" ref="datePicker"/>;
  }
});

React.render(<DatePicker />, document.getElementById('content'));
```

請問一共用了幾個Library?

CommonJs

- Node.js 所採用的規範
- 模組化
- 每一個文件獨立模組
- 使用 `require` 載入文件
- 使用 `module.exports` 匯出功能

模組化

```
var $ = require('jquery');  
var ui = require('jquery-ui');  
var React = require('react');
```

所依賴的Library

```
var DatePicker = React.createClass({  
  componentDidMount: function () {  
    var datePicker = React.findDOMNode(this.refs.datePicker);  
    var $datePicker = $(datePicker);  
  
    $datePicker.datepicker();  
  },  
  render: function () {  
    return <input type="text" ref="datePicker"/>;  
  }  
});  
  
module.exports = DatePicker;
```

模組化



```
var React = require('react');
```

```
var DatePicker = require('./DatePicker.js');
```

 使用自己撰寫的模組

```
React.render(<DatePicker />, document.getElementById('content'));
```

- 容易管理相依性
- 容易打包檔案

```
browserify "app.js" -t babelify --outfile "bundle.js"
```

撰寫一個簡單的Module

- 撰寫一個加法功能

```
function Add(a, b){  
    return a + b;  
}
```

```
module.exports = Add;
```

- 引用加法功能

```
var Add = require('./Add');
```

```
var result = Add(1, 2);  
console.log(result);
```

- 練習
 - 練習撰寫 CommonJs 的 Module



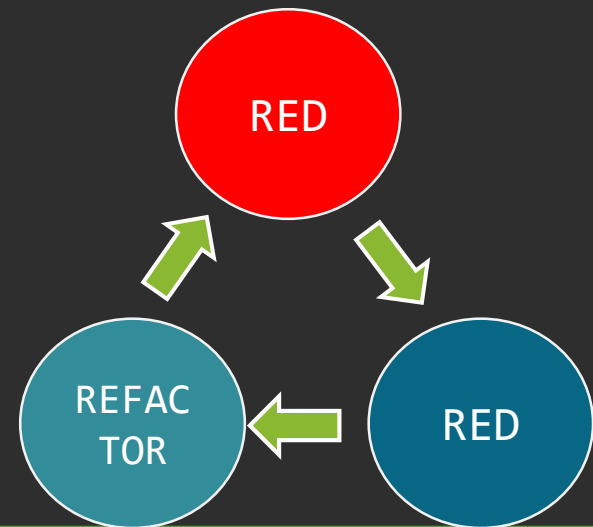
小結

- Node.js
- Npm
- CommonJs

TDD

TDD

- 設定目標
- 達成目標
- 重新整理程式碼 （ 重構 ）
- 滿足所有需求



mocha

- Open Source
- 單元測試 Library
- 支援 TDD、BDD
- 使用簡單
- 同時支援 browser 和 node.js

mocha

- 安裝 mocha

```
npm install mocha --save-dev
```

- 使用 mocha

```
require('mocha');
```

```
describe('Calculator', function () {  
  describe('Add', function () {  
    it('1 add 1 should be 2', function(){  
      // Test  
    });  
  });  
});
```

描述測試案例

- **describe**

描述物件或方法

```
describe('Calculator', function () {  
  describe('#Add()', function () {  
  }  
})
```

- **it**

描述測試情境

```
it('1 add 1 should be 2', function(){  
})
```

3A 原則

- Arrange

準備資料

```
// Arrange
```

```
var first = 1;
```

```
var second = 1;
```

```
var expected = 2;
```

```
var actual = 0;
```

```
var calculator = new Calculator();
```

- Act

執行動作

```
// Act
```

```
actual = calculator.Add(first, second);
```

- Assert

驗證需求

```
// Assert
```

```
actual.should.equal(expected);
```

測試範圍

- 三正一反
- 標準期望值
- 邊界值
- 例外狀況
- 盡可能涵蓋所有情境

- 練習
 - 練習使用 `mocha` 撰寫單元測試

Hooks

- 準備測試使用的資料

- **before**

所有測試之前

```
before(function(){  
  // Preparation  
});
```

- **after**

所有測試之後

```
after(function(){  
  // Cleanup  
});
```

- **beforeEach**

每個測試之前

```
beforeEach(function(){  
  // Preparation  
});
```

- **afterEach**

每個測試之後

```
afterEach(function(){  
  // Cleanup  
});
```

Inclusive and Exclusive

- **only**

只執行某個測試

```
it.only('1 add 1 should be 2', function(){  
  // Test  
});
```

- **skip**

跳過某個測試

```
it.skip('1 add 1 should be 2', function(){  
  // Test  
});
```

Dynamic Generating Test

- 使用資料集進行測試

```
var dataSets = [  
    { args: [1, 1], expected: 2 },  
    { args: [1, 2], expected: 3 },  
    { args: [0, 0], expected: 0 },  
];  
  
dataSets.forEach((data) => {  
    it(`${data.args[0]} add ${data.args[1]} should be ${data.expected}`, () => {  
        // Arrange  
        var calculator = new Calculator();  
  
        // Act  
        var actual = calculator.Add.apply(null, data.args);  
  
        // Assert  
        actual.should.equal(data.expected);  
    });  
});
```

Time

- **slow**

找出緩慢程式碼

```
describe('#Add()', function () {  
  this.slow(10000);  
})
```

- **timeout**

允許執行時間

```
describe('#Add()', function () {  
  this.timeout(500);  
})
```

Reporter

- 支援多種格式的 Reporter
 - Spec
 - Markdown
 - Dot Matrix
 - Html
 - Nyan
 - Doc
 - Progress
 - JSON

```
# 產生自訂 Report
mocha test REPORTER=doc \
  | cat docs/head.html - docs/tail.html \
  > docs/test.html
```

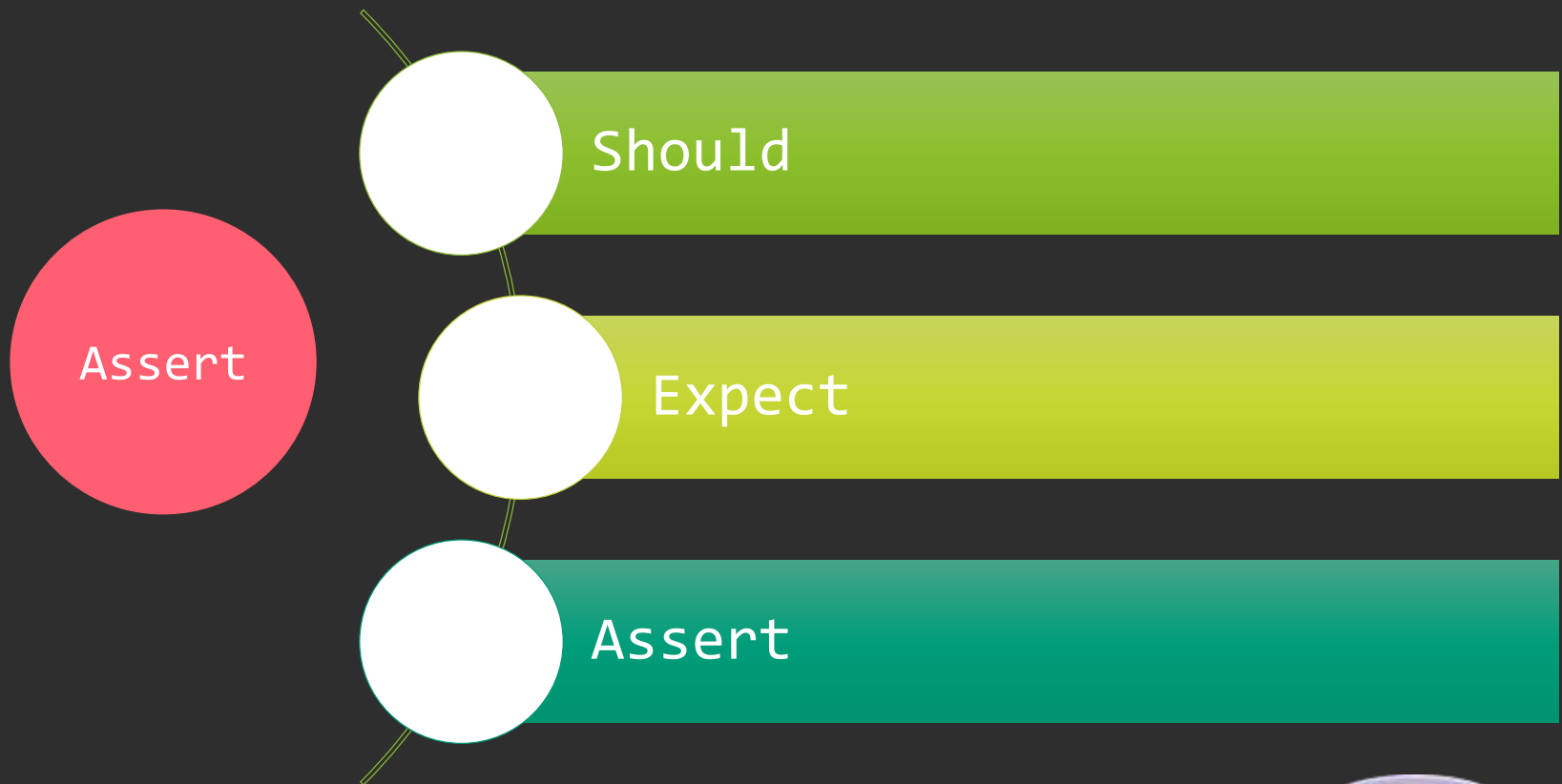
- 練習
 - 練習使用 `mocha` 進階功能

Assertion Library

Assertion

- 輔助使用的類別
- 每個測試都必須要一個
- 避免一個測試過多 Assertion
- 驗收測試正確
- 確認符合預期

Chai.js



<http://chaijs.com/>

Chai.js

- 初始化

```
var chai = require('chai')  
var should = chai.should();
```

- 使用 **should** 驗證

```
result.should.equal(parseInt(result));  
result.should.be.a('number');  
result.should.have.length(3)  
result.should.deep.equal({ count: 1});
```

Chai.js

- 是否相同

```
object.should.equal(expected)
object.should.eql
object.should.deep.equal(obj)
object.should.be.a('string')
object.should.include(val)
```

- Boolean 判斷

```
object.should.be.ok(val)
object.should.be.true
object.should.be.false
```

- 陣列

```
object.should.have.members([2, 3])
object.should.have.keys(['foo'])
object.should.have.key('foo')
```

- Reference

```
object.should.be.null
object.should.be.undefined
object.should.be.empty
object.should.be.arguments
object.should.be.function
object.should.be.instanceOf
```

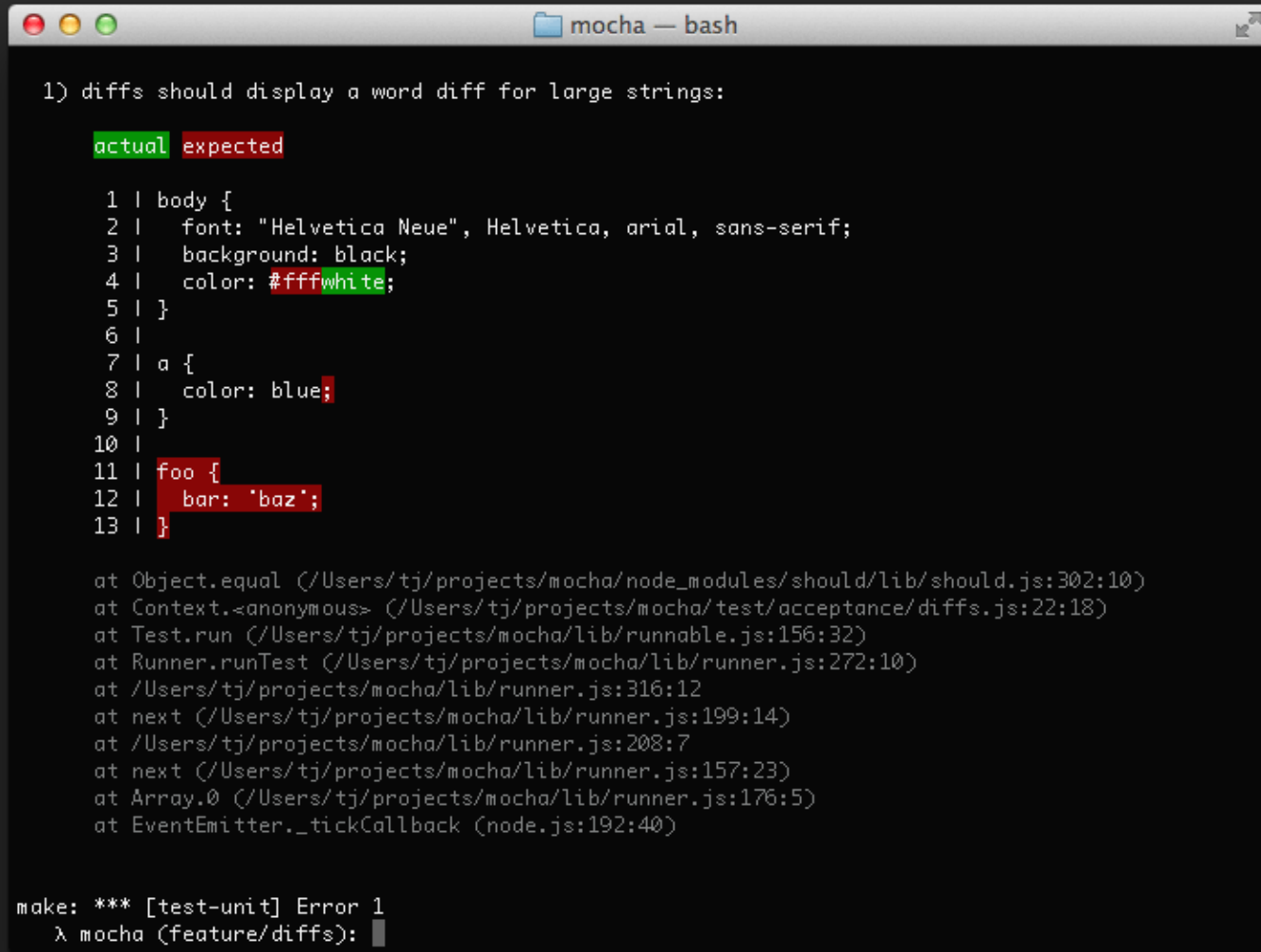
- 大小

```
object.should.gt(5)
object.should.gte
object.should.lt(5)
```

<http://chaijs.com/api/bdd/>

Object Diff

- 比較物件的詳細差異



```
mocha — bash

1) diffs should display a word diff for large strings:

  actual expected
  1 | body {
  2 |   font: "Helvetica Neue", Helvetica, arial, sans-serif;
  3 |   background: black;
  4 |   color: #fffwhite;
  5 | }
  6 |
  7 | a {
  8 |   color: blue;
  9 | }
 10 |
 11 | foo {
 12 |   bar: 'baz';
 13 | }

at Object.equal (/Users/tj/projects/mocha/node_modules/should/lib/should.js:302:10)
at Context.<anonymous> (/Users/tj/projects/mocha/test/acceptance/diffs.js:22:18)
at Test.run (/Users/tj/projects/mocha/lib/runnable.js:156:32)
at Runner.runTest (/Users/tj/projects/mocha/lib/runner.js:272:10)
at /Users/tj/projects/mocha/lib/runner.js:316:12
at next (/Users/tj/projects/mocha/lib/runner.js:199:14)
at /Users/tj/projects/mocha/lib/runner.js:208:7
at next (/Users/tj/projects/mocha/lib/runner.js:157:23)
at Array.0 (/Users/tj/projects/mocha/lib/runner.js:176:5)
at EventEmitter._tickCallback (node.js:192:40)

make: *** [test-unit] Error 1
λ mocha (feature/diffs):
```

- 練習
 - 練習使用 Assertion Library



小結

- 測試起手式
- Mocha
- Chai

測試之替身術

單元測試的重點

- 隔離相依性
- 測試時只針對目標邏輯
- KISS 原則
- 需要 假物件

常見的假物件

- **Stub**

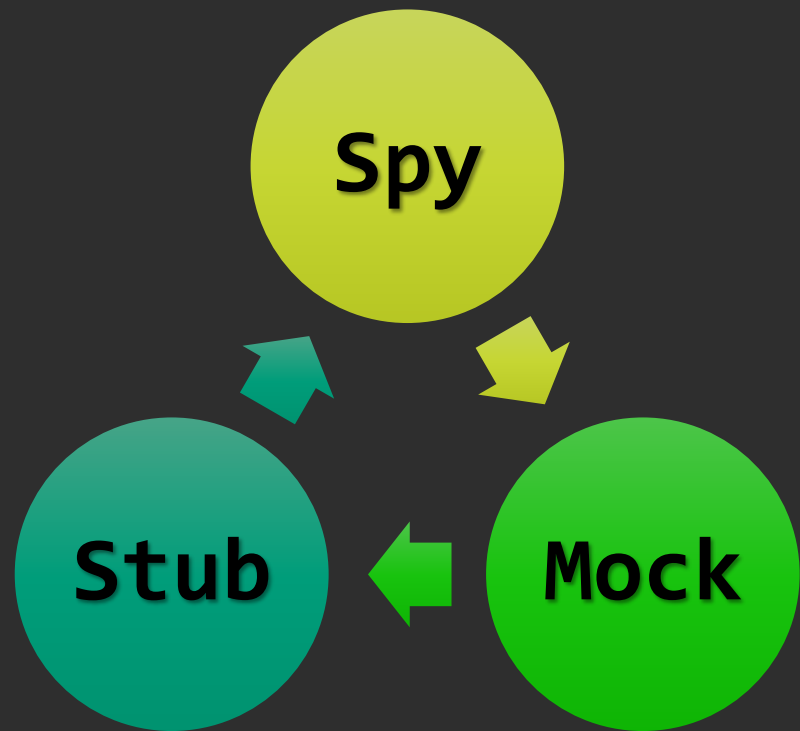
回傳指定物件

- **Mock**

嚴格檢定呼叫次數、內容

- **Spy**

確認是否有被呼叫過



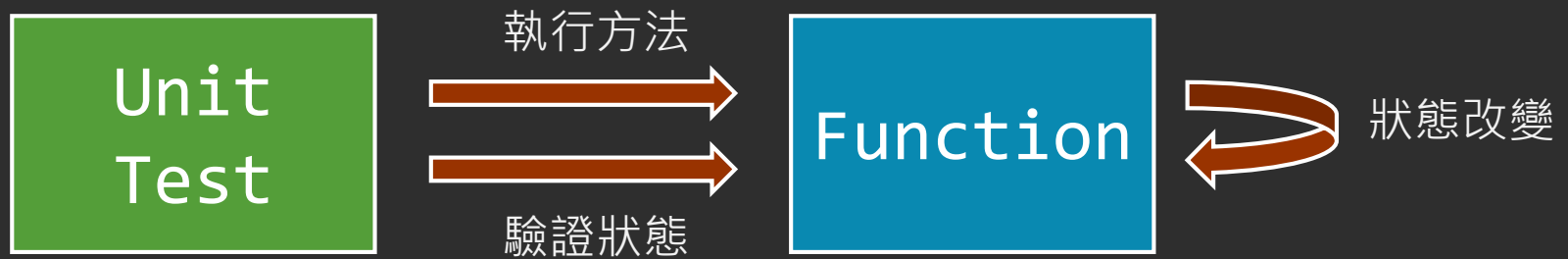
STUB

- 驗證回傳值



STUB

- 驗證狀態改變



Stub

- 假設有一個取得數字的方法

```
var service = {  
  getNumber: function(){  
  }  
}
```

- 透過 Stub 可以替換調原本的方法

```
service.getNumber = sinon.stub().returns(123);
```

```
console.log(service.getNumber());  
console.log(service.getNumber());  
console.log(service.getNumber());
```

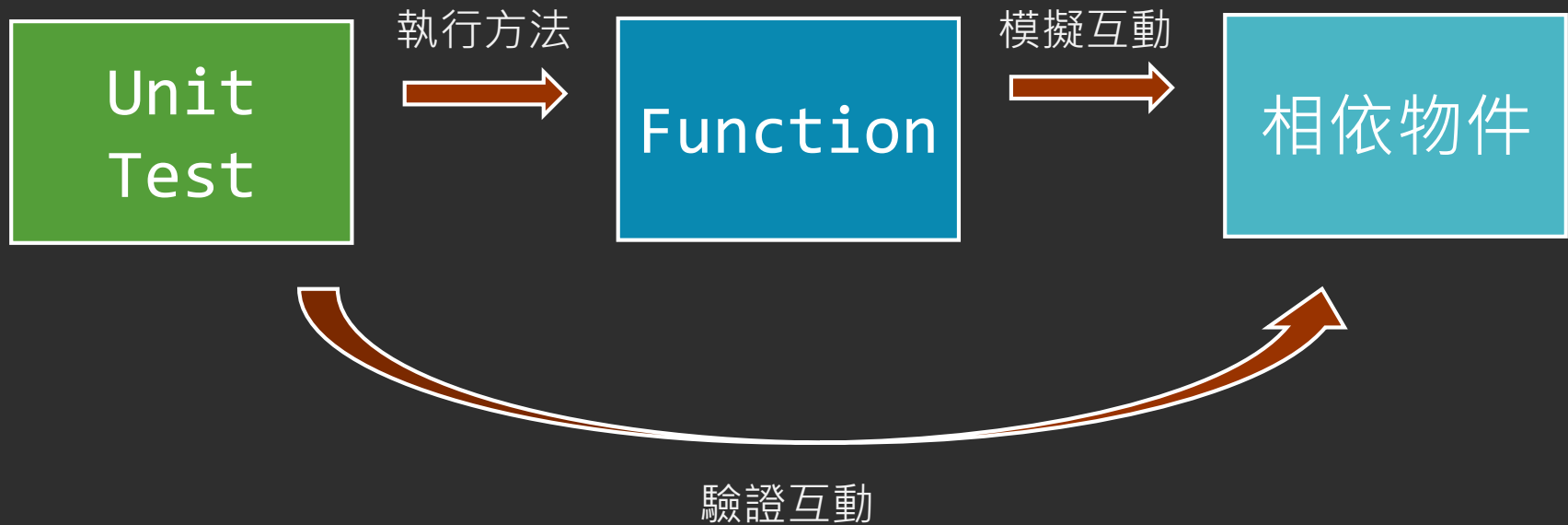
➡ 123

- 練習
 - 練習使用 Stub



MOCK

- 驗證外部互動



Mock

- 嚴謹的檢查呼叫次數、回傳值

```
var mock = sinon.mock(service);  
mock.expects('getNumber').once().returns(223);
```

```
console.log(service.getNumber());
```



123

```
console.log(service.getNumber());  
console.log(service.getNumber());  
console.log(service.getNumber());
```

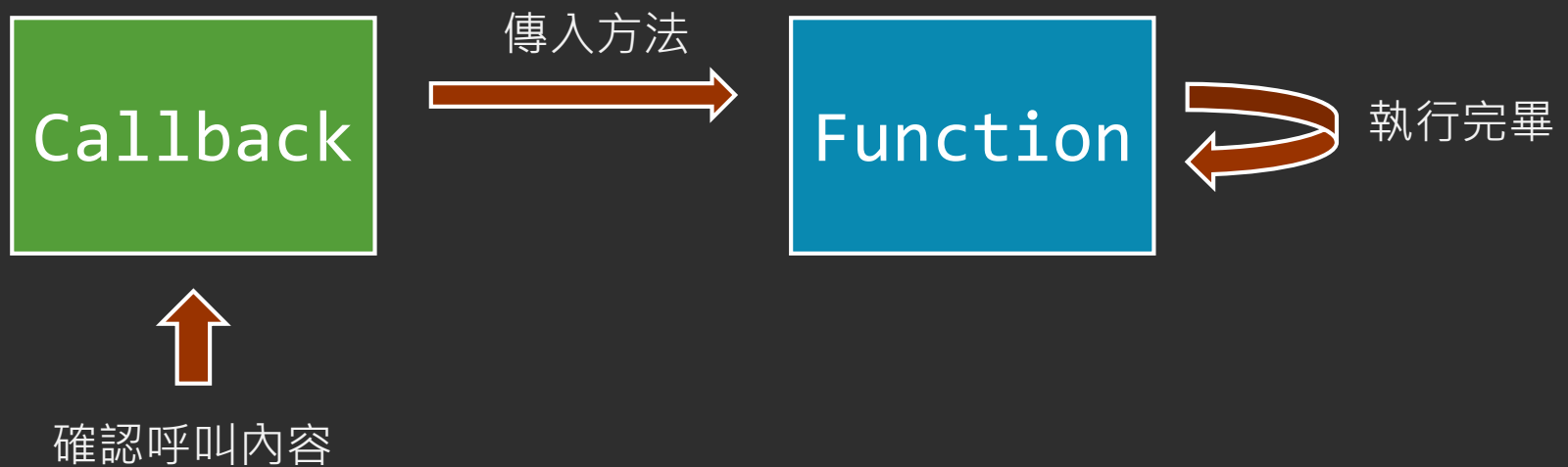


```
/Users/kirkchen/Codes/cucumber-js-sample/node_modules/sinon/lib/sinon/mock.js:452  
    throw exception;  
    ^  
  
ExpectationError: Unexpected call: getNumber()  
  Expectation met: getNumber([...]) once
```

- 練習
 - 練習使用 Mock

Spy

- 驗證觸發事件



Spy

- 處理資料的程式

```
function processData(data, callback){  
  //// long running logic  
  
  setTimeout(function(){  
    callback('success');  
  }, 5000);  
}
```

當資料處理完時才會執行

- 確認有被呼叫

```
var callback = sinon.spy();  
  
callback.called.should.be.true;
```

- 練習
 - 練習使用 Spy



讓 promise 測試更容易

Asynchronous Test

- 使用 `done` 在完成時呼叫

```
it('#GetAsync()', (done) => {  
  // Assert  
  actual.then((data)=>{  
    data.should.eql(expected);  
    done();  
  });  
});
```

Chai as Promised

- Chai 的擴充套件
- 不用額外撰寫 callback 處理
- 使用 **eventually** 及 **notify**

```
this.Then(/^取得的 post 資料應該為$/, function(result, callback) {  
  this.result.then(function(data) {  
    data.should.deep.equal(JSON.parse(result));  
    callback();  
  })  
});
```



```
this.result.should.eventually.deep.equal(JSON.parse(result))  
  .and.notify(callback);
```

使用 async/ await

```
beforeEach(async function() {  
  await db.clear();  
  await db.save([tobi, loki, jane]);  
});
```

```
describe('#find()', function() {  
  it('responds with matching records', async function() {  
    const users = await db.find({ type: 'User' });  
    users.should.have.length(3);  
  });  
});
```

- 練習
 - 練習改寫 Promise 的 Assert 方法



讓 ajax 也能測試

Ajax 怎麼測試？

- 描述使用方式

```
// Arrange
var postId = 1;
var expected = {
  id: 1,
  content: 'Test Post'
};
var actual = undefined;
var postService = new PostService();

// Act
actual = postService.GetPost(postId);

// Assert
actual.should.eventually.eql(expected);
```

Ajax 怎麼測試？

- 透過 api 取得 post 的資料

```
function PostService(){  
}  
  
PostService.prototype.GetPost = function(id){  
  return $.ajax({  
    url: root + '/posts/' + id,  
    method: 'GET'  
  });  
}
```

Ajax 怎麼測試？

- 用 假物件 替換 \$.ajax

```
beforeEach(() => {  
  this.deferred = $.Deferred();  
  
  this.deferred.resolve(data);  
  
  sinon.stub($, "ajax")  
    .returns(this.deferred.promise());  
  
});  
  
afterEach(() => {  
  $.ajax.restore();  
});
```

- 練習
 - 練習測試 Ajax 方法



小結

- Mock
- Stub
- Spy

HOMEWORK

- 建立一個 **購物車** 應用程式，必須要能夠根據會員的等級，提供不同的折扣方式。
 - 如果是 **VIP** 會員，只要購物滿 **500** 元，就一律有 **8** 折優惠
 - 如果是 **一般會員 (Normal)**，除了購物必須要滿 **1000** 元，而且購買超過 **3** 件商品才能擁有 **85** 折優惠
- 練習步驟
 - 請使用 **TDD** 的方式進行開發，一個案例，一個實現
 - 複雜度盡可能的低
 - 如果可以的話，盡量不要使用 **if**

Blog 是記錄知識的最佳平台



<https://dotblogs.com.tw>

OzCode

Your Road to Magical Debugging



```
float CalculateCost( Customer customer, string restaurant)
{
    float courseCost = GetCourseCost(restaurant);
    bool shouldTip = waiter.IsNice && courseCost > COSTLY_MEAL;
```

Exceptions Trail:



ReservationException HotelException **IndexOutOfRangeException**

Exception:

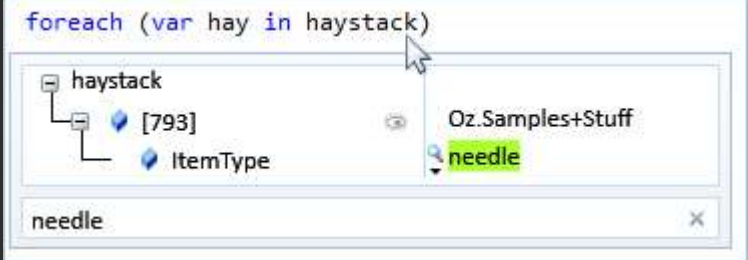
System.IndexOutOfRangeException

Message:

"Invalid customer index"

[Go to where exception was thrown](#) [Go to where exception was handled](#)

<http://www.oz-code.com/>



學員可使用 Yammer 取得優惠價

謝謝各位

<http://skilltree.my>

-
- 本投影片所包含的商標與文字皆屬原作者所有，僅供教學之用。
 - 本投影片的內容包括標誌、設計、文字、圖像、影片、聲音...等著作財產權均屬電魔小鋪有限公司所有，受到中華民國著作權法及國際著作權法律的保障。對本投影內容進行任何形式的引用、轉載、重製前，請務必取得電魔小鋪有限公司的"書面授權"，否則請勿使用，以免侵權。