Test Driven Development (TDD)

Learning concepts

We LEARN more when we LEARN TOGETHER

Learning concepts

We LEARN more when we LEARN TOGETHER

The heart of learning is to focus on the concepts, which is more important than focusing on the details 🥰

Learning concepts

We LEARN more when we LEARN TOGETHER

The heart of learning is to focus on the concepts, which is more important than focusing on the details 🥰

I can **EXPLAIN** it to you, but I can't **UNDERSTAND** it for you 💪

Agenda

1. The World of Software Testing
2. Code Refactoring
3. Isolated Test Environments
4. High Level Testing
5. Workshop

Code

Refactoring

Code Refactoring

“It is the process of restructuring existing computer code without changing its external behavior. Refactoring is intended to improve the design, structure, and/or implementation of the software, while preserving its functionality. Potential advantages of refactoring may include improved code readability and reduced complexity; these can improve the source code's maintainability and create a simpler, cleaner, or more expressive internal architecture or object model to improve extensibility.” — Wikipedia

`Code design`  `Maintainability`

💖 Clean Coding

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.” — Martin Fowler

 Challenge 05

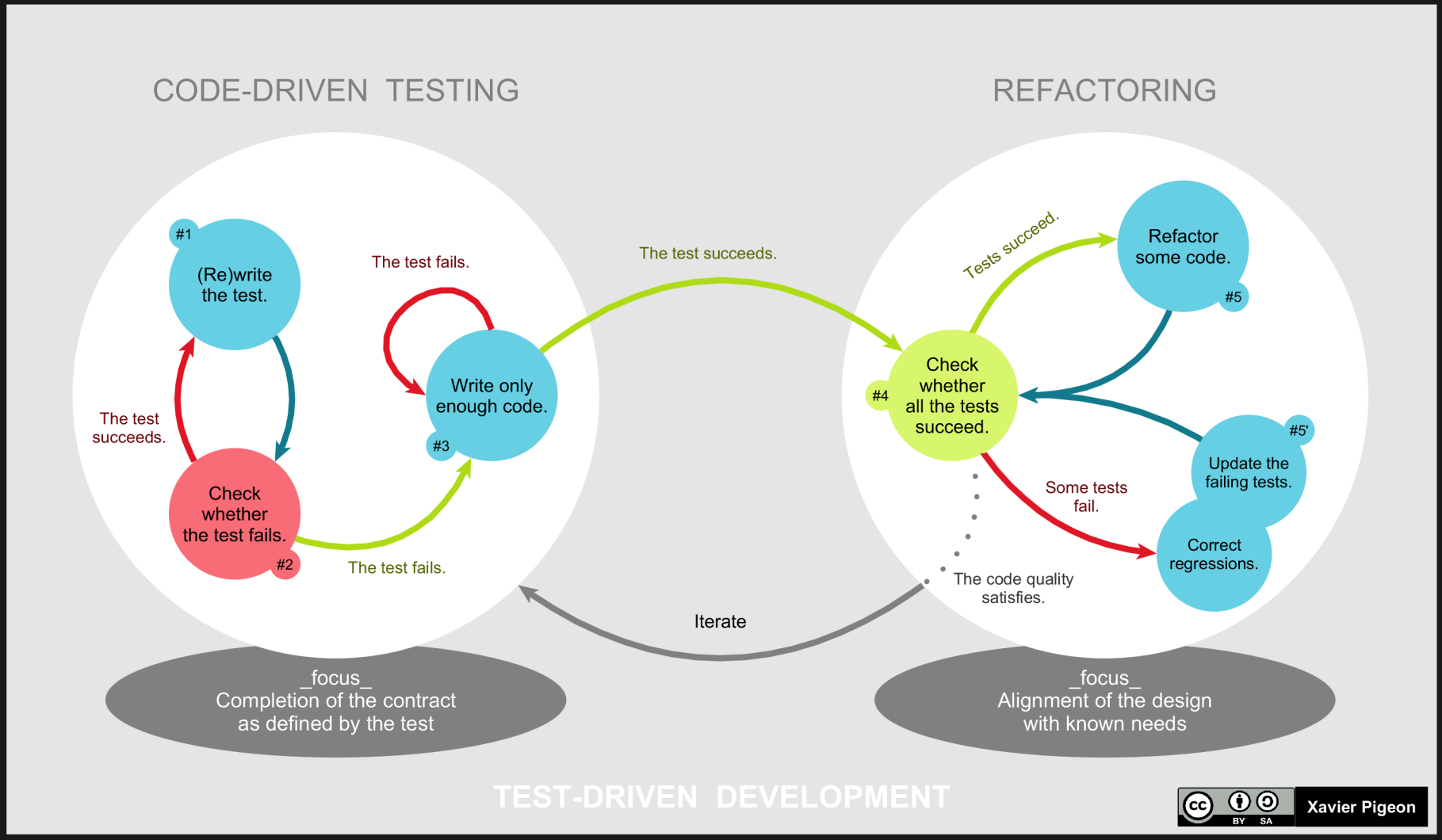
 หยุดก่อนอานนท์ เราร้อนวิชาแล้ว

โปรแกรมตัดเกรด

ช่วงคะแนน	85+	70+	60+	50+	อื่นๆ
เกรดที่ได้	A	B	C	D	F

แก้เกณฑ์การให้คะแนนได้ — เลือกแสดงผลลัพท์ให้ออกทางหน้าจอ หรือ จะพิมพ์ลงกระดาษได้

TDD Cycle



Design Principles

Single-Responsibility Principle

ของต่างๆควรมีหน้าที่รับผิดชอบแค่เรื่องเดียว เพื่อความชัดเจนในขอบเขตหน้าที่รับผิดชอบ เพราะเมื่อมี Requirement change เกิดขึ้น ของที่กระทบจะอยู่ในวงจำกัด ไม่ลามไปหาตัวอื่นที่ไม่เกี่ยวข้อง

💖 Design Principles

	❌ห้าม	✅ให้
1	ยัด Principle หรือ Design ต่างๆเข้ามาตั้งแต่แรก	รอเหตุผลที่ดีพอเข้ามาก่อน ค่อยตัดสินใจนำมาใช้อีกที (การแก้เชื่อกมันยากกว่าการผูกเชือก)
2	เพิ่ม Coupling ในระบบ	เพิ่ม Cohesion โดยต้องไม่เพิ่มความซับซ้อน

xUnit Attributes

- `Fact` - เทสเคสหลักของสิ่งที่ต้องการจะเทส ข้อมูลที่ใช้ในการทดสอบตายตัว
- `Theory` - เทสเคสที่ใช้เป็นข้อพิสูจน์เทสเคสหลัก ซึ่งสามารถมีข้อมูลทดสอบได้หลายชุด

Demo D01

Refactor Test Cases

xUnit → Theory

1. `InlineData`
2. `MemberData`
3. `ClassData`

xUnit → Theory

1. `InlineData`

```
[Theory]
[InlineData(3, 5, 8)]
[InlineData(2, 8, 10)]
public void Add_TwoPositiveValues_MustBeWorkingCorrectly(int input1, int input2, int expect
{
    var sut = new Calculator();
    var result = sut.Add(input1, input2);
    Assert.Equal(expected, result);
}
```

2. `MemberData`
3. `ClassData`

xUnit → Theory

1. `InlineData`
2. `MemberData`

```
[Theory]
[MemberData(nameof(PositiveValueCases))]
public void Add_TwoPositiveValues_MustBeWorkingCorrectly(int input1, int input2, int expect
{
    var sut = new Calculator();
    var result = sut.Add(input1, input2);
    Assert.Equal(expected, result);
}

public static IEnumerable<object[]> PositiveValueCases => new List<object[]>
{
    new object[] { 3, 5, 8 },
    new object[] { 2, 8, 10 },
};
```

3. `ClassData`

xUnit → Theory

1. `InlineData`
2. `MemberData`
3. `ClassData`

```
[Theory]
[ClassData(typeof(PositiveValueCases))]
public void Add_TwoPositiveValues_MustBeWorkingCorrectly(int input1, int input2, int expecte
{
    var sut = new Calculator();
    var result = sut.Add(input1, input2);
    Assert.Equal(expected, result);
}

public class PositiveValueCases : TheoryData<int, int, int>
{
    public PositiveValueCases()
    {
        Add(3, 5, 8);
        Add(2, 8, 10);
    }
}
```

🎮 Challenge 06

Palindrome Password

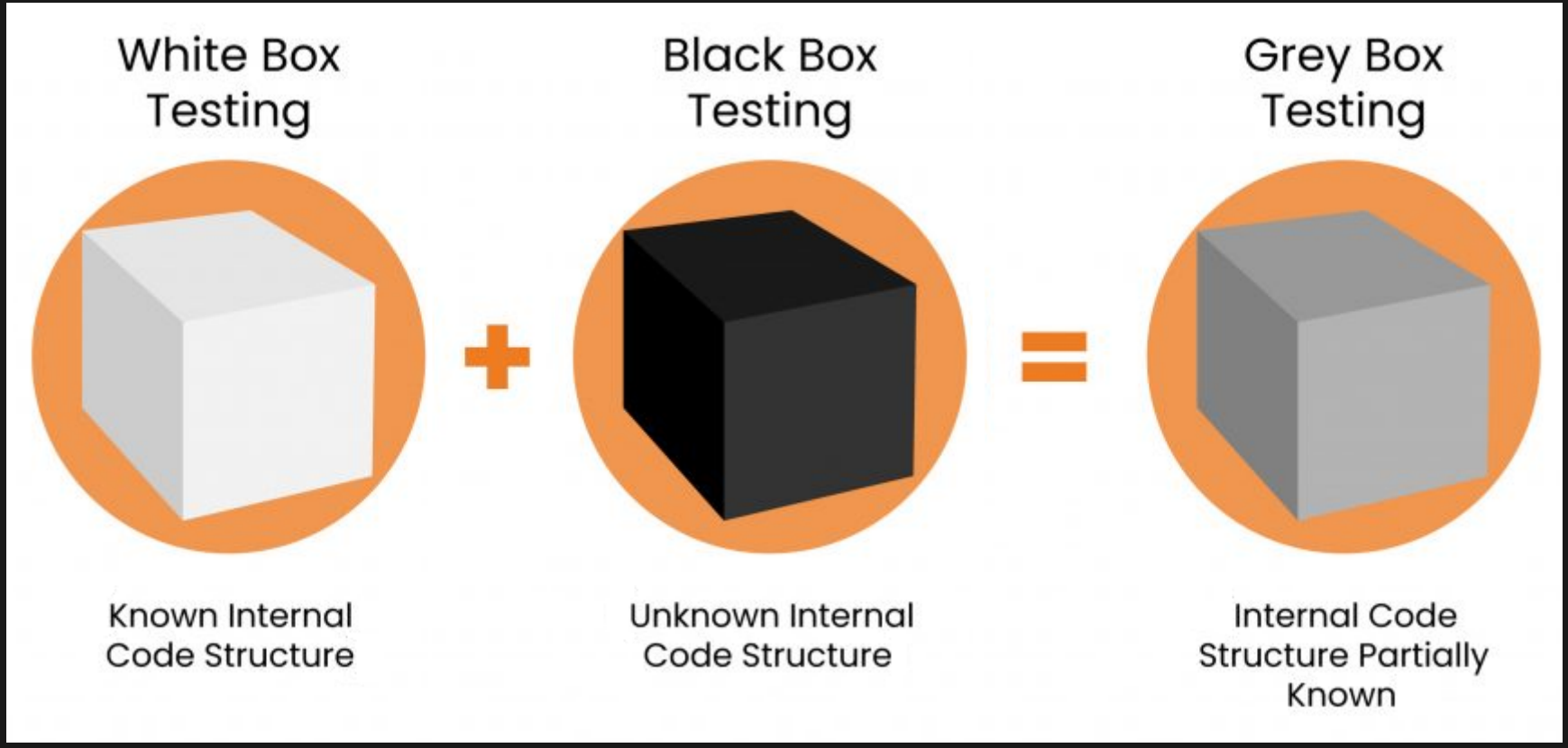
Username	Password	Result
Saladpuk	123456	❌
Hello	olleH	✅
Selles	selles	✅

Test data sources

1. `Developers` — การฝังเทสเคสเข้าไปในตัวโค้ดโดยตรง (เดฟเท่านั้นที่ทำได้)
2. `Non-developers` — การเขียนเทสเคสจากภายนอกตัวระบบ แล้วโหลดไฟล์เหล่านั้นกลับเข้าไปในระบบเทสเพื่อทำการทดสอบ (ใครก็ทำได้)

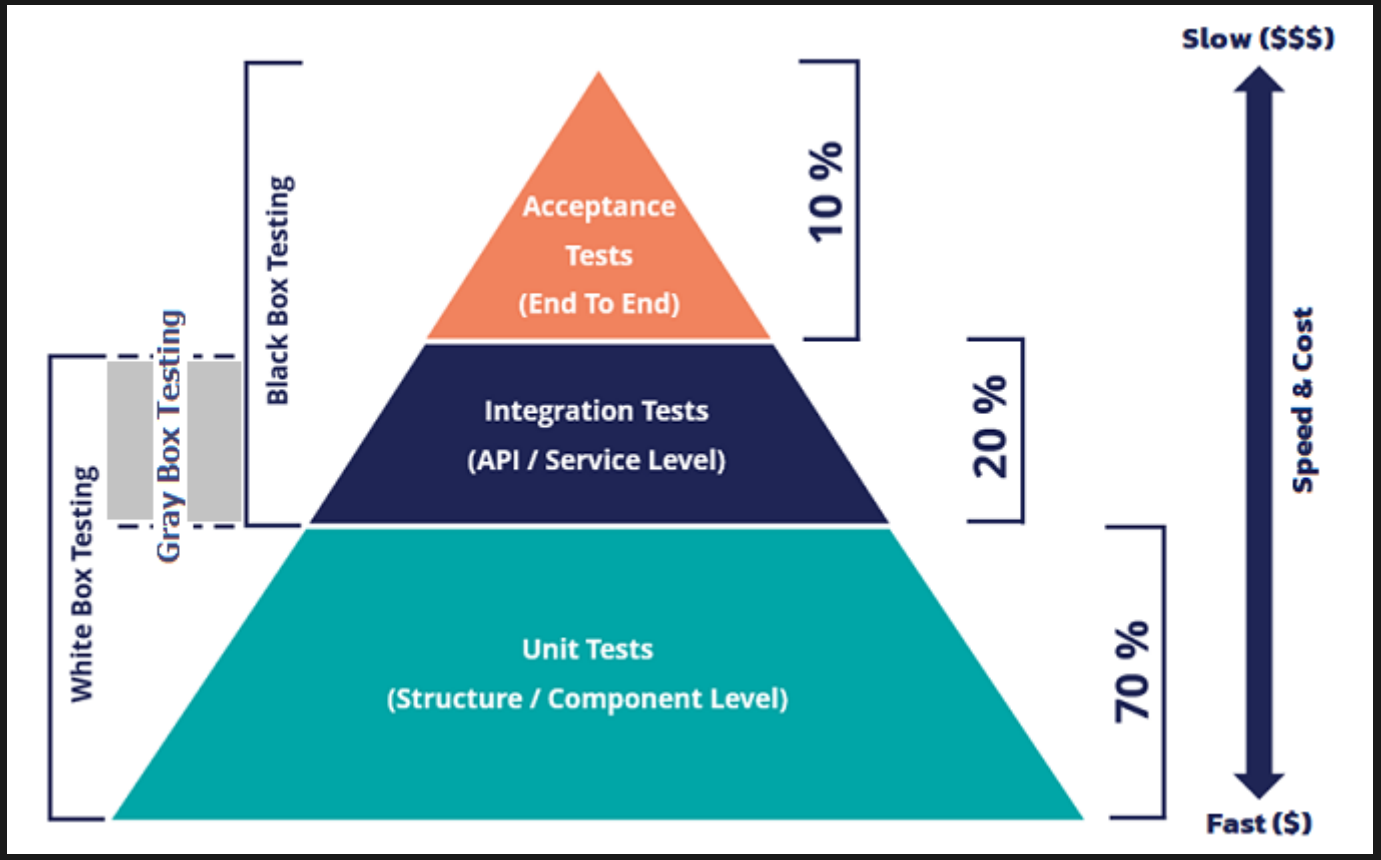
`Local data source` — `Shared data source`

Testing methods



รูปแบบ	เป้าหมายในการตรวจสอบ
White Box	Output & Object's states
Black Box	Output & Behaviour
Gray Box	White Box & Black Box

Testing methods



Demo D02

Black Box Testing

Serilog .net

 Unit Testing Tips

1. การย่อยเทสใหญ่ๆให้กลายเป็นเทสเล็กๆ
2. หมดความสำคัญก็เตะออก (Guard Clauses)
3. การพิมพ์ผลลัพท์ในเทสด้วย (ITestOutputHelper)
4. เขียนเทสเพื่อตรวจสอบผลลัพท์และพฤติกรรม อย่าตรวจสอบรายละเอียดลึกๆ เพื่อลด (Brittle code)
5. Code coverage - การมีเทสครอบคลุมไม่ได้หมายถึงมันจะไม่มี bug (มันอาจจะซ่อนอยู่แต่เราไม่รู้ตัวก็ได้)

 Challenge 07

Password Validation

ขั้นต่ำ 8 ตัวอักษร สูงสุด 256 ตัวอักษร ต้องมีตัวเลขอย่างน้อย 1 ตัว ต้องมีอักขระพิเศษอย่างน้อย 1 ตัว ห้ามมีช่องว่าง ต้องมีตัวพิมพ์เล็กอย่างน้อย 1 ตัว ต้องมีตัวพิมพ์ใหญ่อย่างน้อย 1 ตัว ห้ามใช้ Regular expression เนื่องจากปัญหาด้าน performance

Fluent Assertions

“Nothing is more annoying than a unit test that fails without clearly explaining why. More than often, you need to set a breakpoint and start up the debugger to be able to figure out what went wrong.” — Fluentassertions

Demo D03

Fluent Assertions

Fluent Assertions

```
[Theory(DisplayName = "นำตัวเลขที่บวกสองตัวไปรวมกัน ผลลัพท์ต้องเป็นบวก")]
[InlineData(3, 5, 8)]
[InlineData(2, 8, 10)]
public void Add_TwoPositiveValues_TheResultMustBePositive(int input1, int input2, int exp
{
    var calculator = new Calculator();
    calculator.Add(input1, input2).Should().Be(expected);
}
```

❌ นำตัวเลขที่บวกสองตัวไปรวมกัน ผลลัพท์ต้องเป็นบวก(input1: 3, input2: 5, expected: 8)
Message: Expected result to be positive, but found -2.

❌ นำตัวเลขที่บวกสองตัวไปรวมกัน ผลลัพท์ต้องเป็นบวก(input1: 2, input2: 8, expected: 10)
Message: Expected result to be positive, but found -6.

Best Practices

1. ไม่ต้องเสียงเวลาเขียนเทสให้กับทุกอย่างๆ แต่ให้ทุ่มเวลาเขียนเทสกับ `Core business value` ก่อน ส่วนลำดับถัดมาค่อยเขียนเทสให้กับ `ของที่คลุมเครือ`, `ของที่ส่งผลกระทบสูง` ถึงจะคุ้มกับเวลาที่ลงทุนไป
2. จงดูแลเทสเหมือน `Production code`
3. ของที่เขียนในเทสต้องไม่หลุดไปที่ Production code `Isolation project`
4. ลำดับของเทสไม่ควรมีผลต่อกัน เทสไม่ควรขึ้นต่อกัน `Random failed`
5. เทสควรจะ `ทำงานซ้ำได้เรื่อยๆ` และ `ต้องทำงานได้เร็ว`
6. หลีกเลี้ยงการใช้ static เพื่อป้องกัน `Global state`
7. เทสควรจะมองในมุมของ What ไม่ใช่ How ไม่อย่างนั้นจะทำให้เกิด `Brittle code`

🎮 Challenge `08`

Largest Prime Factor

ตัวเลข	ตัวประกอบ
15	`3` X `5`
13,195	`5` X `7` X `13` X `29`
600,851,475,143	🤔

ต้องหาผลลัพท์ได้ภายใน 1 วินาที

🎮 Challenge 08

Largest Prime Factor

ตัวเลข	ตัวประกอบ
15	3 X 5
13,195	5 X 7 X 13 X 29
600,851,475,143	🤔 6,857

ต้องหาผลลัพท์ได้ภายใน 1 วินาที

🎮 Challenge 09

10001st Prime

ตัวเลขจำนวนเฉพาะ 20 ตัวแรกได้แก่

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71

ลำดับที่	เลขจำนวนเฉพาะคือ
6	13
19	67
10001	🤔

ต้องหาผลลัพท์ได้ภายใน 1 วินาที

🎮 Challenge 09

10001st Prime

ตัวเลขจำนวนเฉพาะ 20 ตัวแรกได้แก่

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71

ลำดับที่	เลขจำนวนเฉพาะคือ
6	13
19	67
10001	🤔 104,743

ต้องหาผลลัพท์ได้ภายใน 1 วินาที

Challenge 10

Number To Text

Input	Expected
1	หนึ่ง
53	ห้าสิบสาม
571	ห้าร้อยเจ็ดสิบเอ็ด
3,234	สามพันสองร้อยสามสิบสี่
620,000	หกแสนสองหมื่น
1,234,567	หนึ่งล้านสองแสนสามหมื่นสี่พันห้าร้อยหกสิบเจ็ด
58,000,341	ห้าสิบแปดล้านสามร้อยสี่สิบเอ็ด

Summary

1. Refactoring process
2. Design Principles & Pitfall
3. Fact & Theory
4. Testing methods `WhiteBox` `BlackBox` `GrayBox`
5. Tips `ย่อยเทส` `Guard clause` `Custom test output`
6. Fluent Assertions
7. Best practices