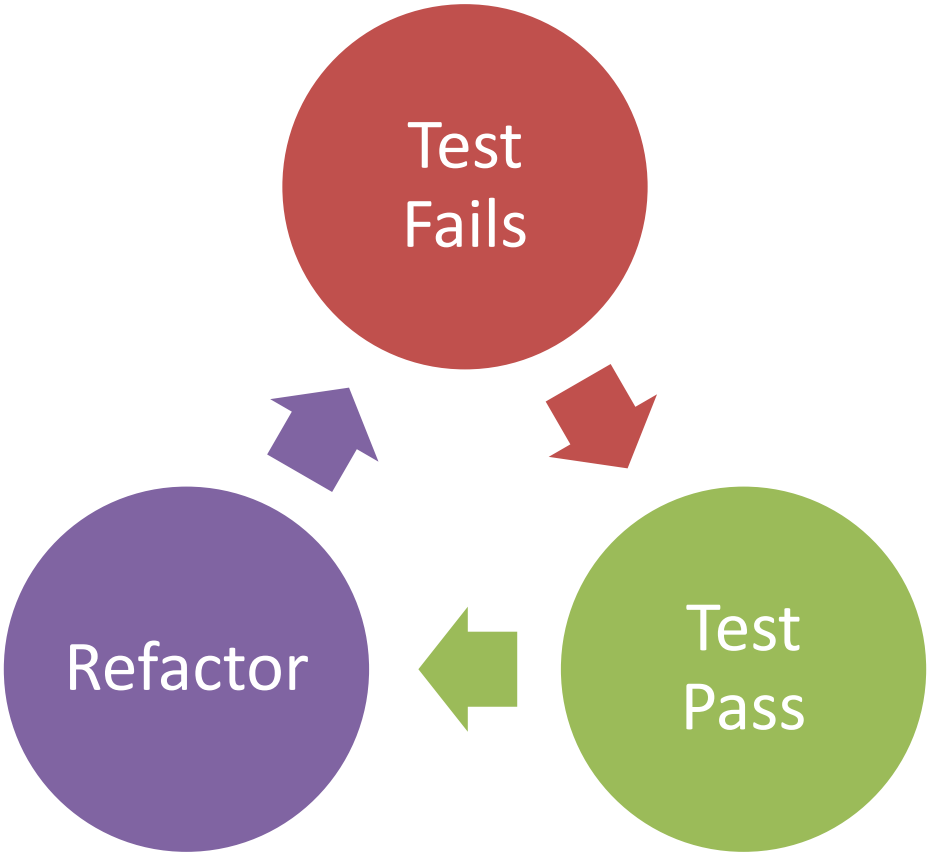
TDD Hands-On <1>

Wei Xiao

June 5, 2015

TDD Cycle



Exercise

Factorize a positive integer number into its prime factors.
找出一个正整数的所有质因数

Problem Domain Analysis

Input	Expected output
2	[2]
3	[3]
4	[2, 2]
6	[2, 3]
8	[2, 2, 2]
9	[3, 3]
10	[2, 5]

Arrange – Act - Assert

```
[Test]
public void should_be_able_to_add_two_numbers_together()
{
        int firstNumber = 1;
        Calculator calculator = new Calculator();
        int secondNumber = 2;
        var result = calculator.Add(firstNumber, secondNumber);
        result.ShouldEqual(3);
}
```

```
[Test] public void should_be_able_to_add_two_numbers_together()
{
        // Arrange
        int firstNumber = 1;
        int secondNumber = 2;
        Calculator calculator = new Calculator();
        // Act
        var result = calculator.Add(firstNumber, secondNumber);
        //Assert
        result.ShouldEqual(3);
}
```

DEMO

https://vimeo.com/97516288
Start from 32’50

Transformation Priority Premise (TPP)

http://blog.8thlight.com/uncle-bob/2013/05/27/TheTransformationPriorityPremise.html

- ({}–>nil) no code at all->code that employs nil
- (nil->constant)
- (constant->constant+) a simple constant to a more complex constant
- (constant->scalar) replacing a constant with a variable or an argument
- (statement->statements) adding more unconditional statements.
- (unconditional->if) splitting the execution path
- (scalar->array)
- (array->container)
- (statement->recursion)
- (if->while)
- (expression->function) replacing an expression with a function or algorithm
- (variable->assignment) replacing the value of a variable.

TDD is design tool or testing tool ??

Classicist TDD

1. Design happens during the refactoring phase.
2. During the refactoring phase, the unit under test may grow to multiple classes.
3. Mocks are rarely used, unless when isolating external systems.
4. No up-front design considerations are made. Design completely emerges from code.
5. It's a great way to avoid over-engineering.
6. Often used in conjunction with the 4 Rules of Simple Design.
7. Good for exploration, when we know what the input and desired output are but we don't really know how the implementation looks like.
8. Great for cases where we can't rely on a domain expert or domain language (data transformation, algorithms, etc.)

http://codurance.com/2015/05/12/does-tdd-lead-to-good-design/

Outside-In TDD

1. Prescribes a direction in which we start test-driving our code: from outside to the inside.
2. Design starts in the *red* phase, while writing the tests.
3. Design is refined during the *refactoring* phase.
4. We normally start with an acceptance test which verifies if the feature as a whole works. With a failing acceptance test informing why the feature is not yet complete. we start writing unit tests.
5. Starts from classes that are closer to the input of the system (outside) and move towards the inside of our application
6. Refactoring phases are much smaller, when compared to the classicist approach.

Which One Is Better?

TDD becomes much easier
when we understand what
good design looks like.

Good Design Guidelines

1. 4 Rules of Simple Design
2. Domain Driven Design
3. SOLID
4. Patterns
5. Law of Demeter
6. Tell Don't Ask
7. Design by Contract
8. Feature Envy
9. Cohesion and Coupling
10. Balanced Abstraction Principle

TDD is workflow

- Starts with confidence
- Deals with specific cases then
  - Concludes generic cases
- Prompts for code improvement

Homework

Give number between 0 to 99, return its English word, E.g.
给出任意个0-99的数字，返回数字的英文。比如

0 → Zero
10 → Ten
21 → Twenty One
99 → Ninety Nine