Unit Testing and TDD

Lesson 02 :
Introduction to TDD

Capgemini

Lesson Objectives

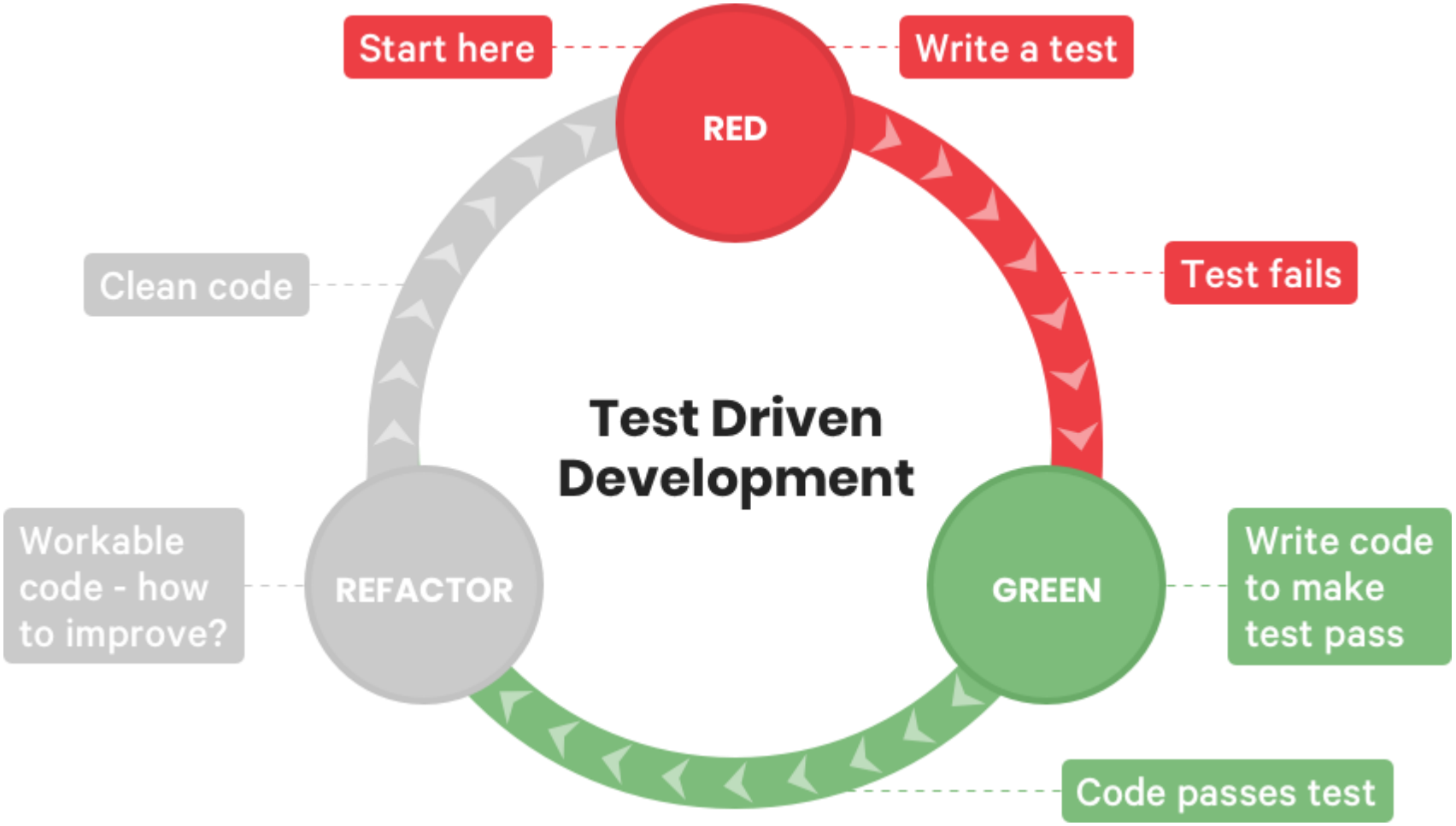
- In this lesson, you will learn:
  - What is TDD?
  - TDD Cycle
  - Advantages of TDD
  - Mocking with Rhino.Mock
  - Creating a Mock
  - Mock Objects

What is Test Driven Development (TDD)?

- TDD is a technique whereby you write your test cases before you write any implementation code
- Tests drive or dictate the code that is developed
- An indication of "intent"
- Tests provide a specification of "what" a piece of code actually does
- Some might argue that "tests are part of the documentation"
- TDD is done at Unit level - i.e. testing the internals of a class
- Tests are written for every function
- Mostly written by developers using one of the tool specific to the application

> "Before you write code, think about what it will do. Write a test that will use the methods you haven't even written yet."

TDD Cycle


Start here
Write a test
RED
Test fails
Clean code
Test Driven Development
Workable code - how to improve?
REFACTOR
GREEN
Write code to make test pass
Code passes test

Advantages of TDD

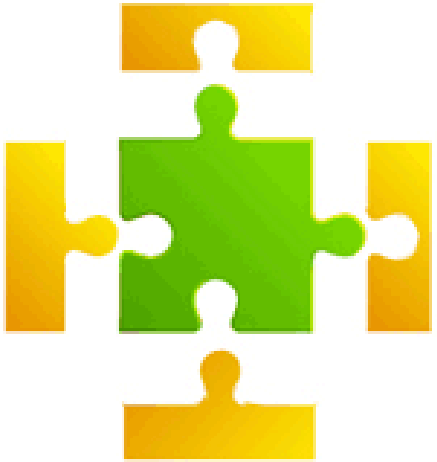
- Better program design and higher code quality
- When writing tests for particular requirements, programmers immediately create a strict and detailed specification
- TDD reduces the time required for project development
- Code flexibility and easier maintenance
- Refactoring becomes easier
- Save project costs in the long run

Mocking

REAL SYSTEM

Green = class in focus
Yellow = dependencies
Grey = other unrelated classes

CLASS IN UNIT TEST



Green = class in focus
Yellow = mocks for the unit test

Mocking

- Mock objects allow you to mimic the behaviour of classes and interfaces
- This isolates the code you're testing, ensuring that it works on its own and that no other code will make the tests fail.
- With mocks, you can set up the object, including giving parameters and return values on method calls and setting properties.
- You can also verify that the methods you set up are being called in the tested code.

Mocking with Rhino.Mocks

- Rhino.Mocks is a dynamic mock object framework for .NET.
- Its purpose is to ease testing by allowing the developer to create mock implementations of custom objects and verify the interactions using unit testing.

Creating a Mock

- To generate mocks, you'll use the static factory Rhino.Mocks.MockRepository.
- From this factory, you'll create your mock using one of a few generic methods:
  - GenerateMock<T> (for DynamicMocks)
  - GeneratePartialMock<T>
  - GenerateStrictMock<T>
- where the T is the class/interface being mocked. The method parameters, if you provide any, will be passed to the object's constructor.
- Rhino.Mocks can only mock/stub virtual members of a real class
- Generally better way is mock/stub the interface

Types of Mock Objects

- Rhino.Mocks supports three basic types of mock objects:
- Strict Mock
  - A strict mock requires you to provide alternate implementations for each method/property that is used on the mock.
  - If any methods/properties are used which you have not provided implementations for, an exception will be thrown.
- Dynamic Mock
  - With a dynamic mock, any methods/properties which are called by your tests for which you have not provided an implementation will return the default value for the data type of the return value.
- Partial Mock
  - A partial mock will use the underlying object's implementation if you don't provide an alternate implementation.
  - So if you're only wanting to replace some of the functionality (or properties), and keep the rest, you'll want to use this.
  - For example, if you only want to override the method IsDatabaseActive(), and leave the rest of the class as-is, you'll want to use a partial mock and only provide an alternate implementation for IsDatabaseActive().

Test Doubles

- In Rhino Mocks we create implementation of dependencies that act as stand-in for the real implementations. Such implementation are called Test Doubles.
- Test Doubles is an object that is used as fake object in place of dependent object.
- Test Doubles are never used in final application.
- These objects are only used in unit testing of an application.
- There are various terms related to test doubles.
  - Dummy Object
  - Stub Object
  - Mock Object

Dummy Object

- These objects are simple objects that stand-in for dependent objects.
- They usually return a predefined response that are not vary based on input parameters.
- In the above example, we directly return 4 as discount without using the quantity parameter.

```
public interface IOrder
{
    int GetDiscount(int quantity);
}

public class DummyObject : IOrder
{
    public int GetDiscount(int quantity)
    {
        return 4;
    }
}
```

Stub

- A stub is simply an alternate implementation
- Stub object is used for State verification.
- Stub object vary their response based on input parameters.

```
public interface IOrder
{
    int GetDiscount(int quantity);
}
public class StubObject : IOrder
{
    public int GetDiscount(int quantity)
    {
        if (quantity < 4)
            return 1;
        else if (quantity < 10)
            return 2;
        else
            return 4;
    }
}
```

Mock

- Mock object is used for Behavior verification.
- Mock object is a step up from Stub object.
- Mock object are also stand-in for dependent object.
- They are also pre-programmed with expectations.
- Mock objects also check the implementation of the test method.
- It also tracks how many times a particular method was called and in what order a sequence of methods were called.

Summary

- In this lesson, you have learnt about:
  - TDD is a technique whereby you write your test cases before you write any implementation code
  - Mock objects allow you to mimic the behaviour of classes and interfaces
  - Rhino.Mocks is a dynamic mock object framework for .NET.