MISA®
TIN CẬY - TIỆN ÍCH - TẬN TÌNH

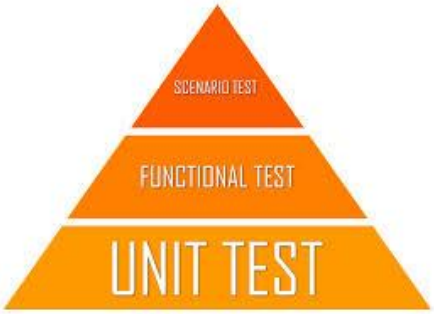
# Giới thiệu Unit Test

Nguyễn Thanh Tùng – CTO MISA

# Nội dung

1. Unit Test là gì?
2. Ưu điểm
3. NUnit Framework
4. Các kỹ thuật để break dependency
5. Isolation Framework: Nsubstitute
6. Best practices

# Unit Test là gì?




# Unit Test là gì?

- Unit Test là một test case test một phần chức năng của code

**Trả về true khi...**



**Trả về false khi...**

```
Public bool IsLoginOK(string user, string password)
{
//.....
}
```

# Ưu điểm của Unit Test?

- Giúp dễ tìm bug hơn
- Giúp code dễ bảo trì hơn
- Giúp code dễ hiểu hơn
- Giúp code dễ phát triển, mở rộng hơn

# Unit Test Framework

- MSTest
- Nunit
- Xunit
- …

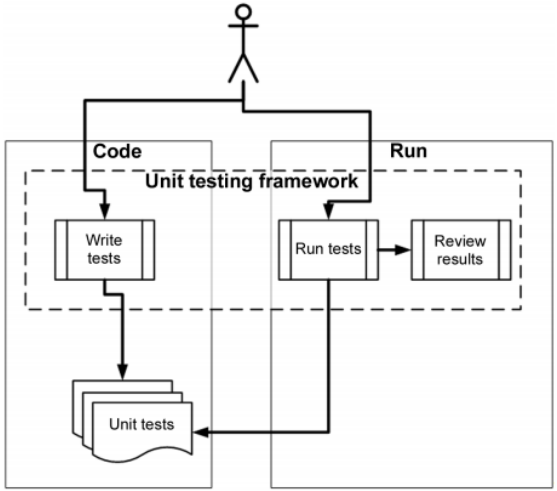
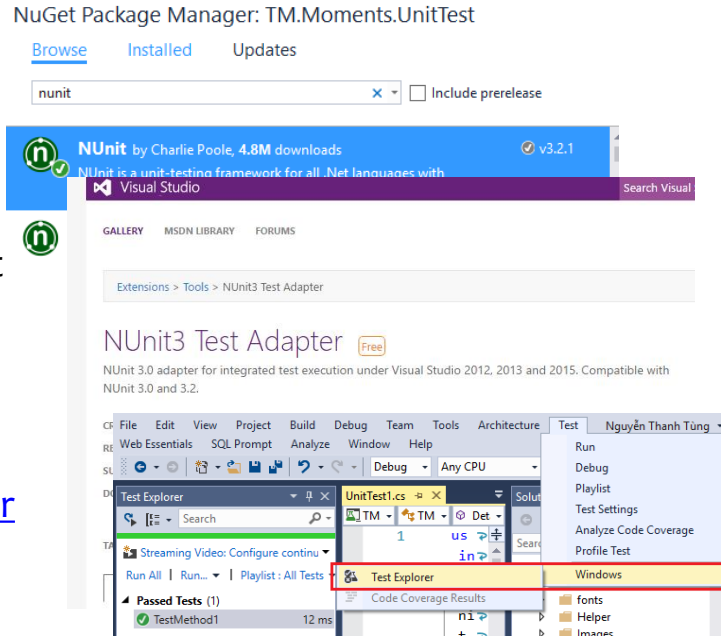



**Figure 2.1 Unit tests are written as code, using libraries from the unit testing framework. Then the tests are run from a separate unit testing tool or inside the IDE, and the results are reviewed (either as output text, the IDE, or the unit testing framework application UI) by the developer or an automated build process.**

# NUnit Framework

- Chọn Nunit thay vì MSTest framework vì:
  - Nunit open source nên nhiều tính năng tốt hơn, performance cao hơn
  - Cập nhật thường xuyên hơn MSTest
- Cài đặt:
  - Nunit Framework từ NuGet
  - Nunit Test Adapter từ Gallery: https://visualstudiogallery.msdn.microsoft.com/0da0f6bd-9bb6-4ae3-87a8-537788622f2d



# NUnit Framework

- Một số Attribute hay dùng
  - TestFixture
  - Test
  - Ignore
  - Description
  - Category
- Một số Assert hay dùng
  - AreEqual
  - IsTrue/False
  - Contains
  - IsNull/NotNull
  - Throws

```
[TestFixture]
public class LogAnalyzerTests
{
    [Test]
    public void IsValidFileName_BadExtension_ReturnsFalse()
    {

    }
}
```

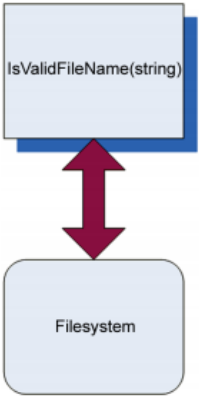
```
[Test]
public void IsValidFileName_BadExtension_ReturnsFalse()
{
    LogAnalyzer analyzer = new LogAnalyzer();

    bool result = analyzer.IsValidLogFileName("filewithbadextension.foo");

    Assert.False(result);
}
```

# Các loại hàm unit test

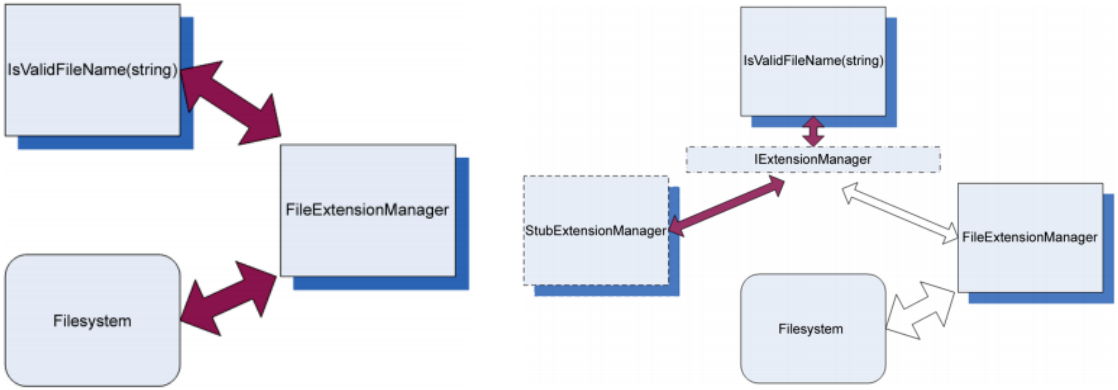
- Có 3 loại hàm cần test
  - Hàm trả về giá trị (Value-based)
  - Hàm không trả về giá trị nhưng thay đổi thuộc tính của đối tượng (State-based)
  - Hàm không trả về đối tượng và gọi sang một hàm khác (Interaction-based)

# Dependency (Tight couple)

- Muốn unit test được tự động và độc lập môi trường -> cần loại bỏ các phụ thuộc này (break dependency – loose couple)



# Break Dependency




- Tạo base class với các hàm virtual để override lại
- Extract interface để tạo ra các class implement khác nhau

# Break Dependency

- Sử dụng Dependency injection
  - **Truyền vào hàm khởi tạo (constructor injection)**
  - **Truyền vào property (property injection)**
  - Truyền vào parameter của hàm (parameter injection)

# Break Dependency

```
public class LogAnalyzer
{
        private IExtensionManager manager;

        public LogAnalyzer(IExtensionManager mgr)
        {
            manager = mgr;
        }
        public bool IsValidLogFileName(string fileName)
        {
            return manager.IsValid(fileName);
        }
}
```

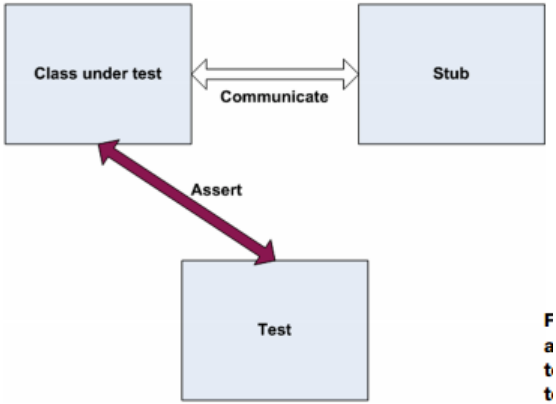
**Defines production code**

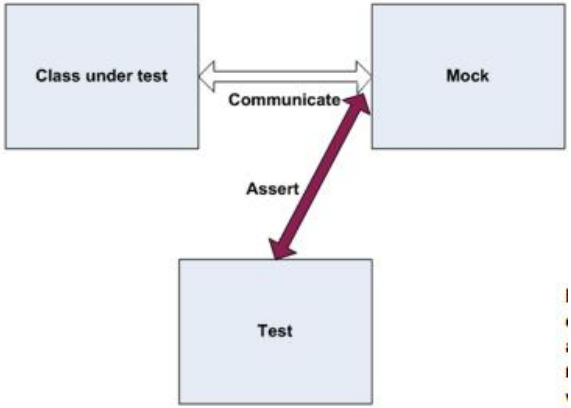
**Defines constructor that can be called by tests**

```
public class LogAnalyzer
{
    private IExtensionManager manager;

    public LogAnalyzer ()
    {
        manager = new FileExtensionManager();
    }
    public IExtensionManager ExtensionManager
    {
        get { return manager; }
        set { manager = value; }
    }

    public bool IsValidLogFileName(string fileName)
    {
        return manager.IsValid(fileName);
    }
}
```

**Allows setting dependency via a property**

# Stub & Mock


Class under test
Communicate
Stub
Assert
Test


Class under test
Communicate
Mock
Assert
Test

# Isolation Framework

- Giúp tạo các object fake (stub, mock) nhanh chóng, dễ dàng bằng config, không cần phải tạo thủ công nữa, loại bỏ các code lặp đi lặp lại.
- Sử dụng Nsubstitute
  - Document rõ ràng
  - Dễ dùng
  - http://nsubstitute.github.io/


**NSubstitute**

*A friendly substitute for .NET mocking frameworks*

# NSubstitute

```
[Test]
public void Returns_ByDefault_WorksForHardCodedArgument()
{
 IFileNameRules fakeRules =   Substitute.For<IFileNameRules>();

 fakeRules.IsValidLogFileName("strict.txt").Returns(true);

 Assert.IsTrue(fakeRules.IsValidLogFileName("strict.txt"));
}
```

```
[Test]
public void Analyze_TooShortFileName_CallLogger()
{
  ILogger logger = Substitute.For<ILogger>();
  LogAnalyzer analyzer = new LogAnalyzer(logger);

  analyzer.MinNameLength = 6;
  analyzer.Analyze("a.txt");

  logger.Received().LogError("Filename too short: a.txt");
}
```

```
[TestFixture]
class LogAnalyzerTests
{
    [Test]
    public void Analyze_TooShortFileName_CallLogger()
    {
        FakeLogger logger = new FakeLogger();

        LogAnalyzer analyzer = new LogAnalyzer(logger);

        analyzer.MinNameLength= 6;
        analyzer.Analyze("a.txt");

        StringAssert.Contains("too short",logger.LastError);
    }
}

class FakeLogger: ILogger
{
    public string LastError;

    public void LogError(string message)
    {
        LastError = message;
    }
}
```

# Best Practices

- **Mỗi một hàm test chỉ test một đơn vị code**
- **Đặt tên gồm 3 thành phần:**
  - TênHàm_ĐầuVào_ ĐầuRaMongMuốn
- **Quy tắc 3A, tránh viết nhiều code logic trong test**
  - Arrange: Chuẩn bị tất cả các dữ liệu đầu vào
  - Act: Gọi hàm cần test
  - Assert: Kiểm tra kết quả có đúng mong đợi không
- **Đảm bảo chạy code coverage kiểm tra sau khi unit test**
- So sánh object nên đưa về so sánh property
- **Khi design class hướng tới việc dùng interface**

# Thanks for your listening

# Q&A