Tackling Basic Unit Testing Scenarios

Kevin Dockx

Architect

@KevinDockx https://www.kevindockx.com

Coming Up

Learning about assertions

Core unit testing scenarios

- Strings, collections, events, exceptions, …

Assert

An assert is a boolean expression, used to verify the outcome of a test, that should evaluate to true

Learning About Assertions

A test can contain one or more asserts

- Fails when **one or more** asserts fail
- Passes when **all** asserts pass

Learning About Assertions

xUnit provides asserts for all common core testing scenarios

“A unit test should only contain one assert”

Quote by “The strict school of thought”

```
[Fact]
public void CreateEmployee_ConstructInternalEmployee_SalaryMustBeEqualTo2500()
{
     // Arrange
     var employeeFactory = new EmployeeFactory();

     // Act
     var employee = (InternalEmployee)employeeFactory.CreateEmployee("Kevin", "Dockx");

     // Assert
     Assert.Equal(2500, employee.Salary);
}
```

Learning About Assertions

```
[Fact]
public void CreateEmployee_ConstructInternalEmployee_SalaryMustBeEqualTo2500()
{
     // Arrange
     var employeeFactory = new EmployeeFactory();

     // Act
     var employee = (InternalEmployee)employeeFactory.CreateEmployee("Kevin", "Dockx");

     // Assert
     Assert.Equal(2500, employee.Salary);
}
```

Learning About Assertions

A unit is a small piece of behavior that you want to test
Multiple assertions in one test are acceptable if they assert the same behavior

```
[Fact]
public void CreateEmployee_ConstructInternalEmployee_SalaryMustBeLargerThanOrEqualTo2500()
{
    // Arrange
    var employeeFactory = new EmployeeFactory();

    // Act
    var employee = (InternalEmployee)employeeFactory.CreateEmployee("Kevin", "Dockx");

    // Assert
    Assert.True(employee.Salary >= 2500);
}
```

Learning About Assertions

Test if salary is larger than or equal to 2500

```
[Fact]
public void CreateEmployee_ConstructInternalEmployee_SalaryMustBeSmallerThanOrEqualTo3500()
{
     // Arrange
     var employeeFactory = new EmployeeFactory();

     // Act
     var employee = (InternalEmployee)employeeFactory.CreateEmployee("Kevin", "Dockx");

     // Assert
     Assert.True(employee.Salary <= 3500);
}
```

Learning About Assertions

Test if salary is smaller than or equal to 3500

Learning About Assertions

We've split up testing one type of behavior across 2 tests

- Unnecessary code
- Costs time and money to create, maintain, manage, refactor and run

```
[Fact]
public void CreateEmployee_ConstructInternalEmployee_SalaryMustBeBetween2500And3500 ()
{
     // Arrange
     var employeeFactory = new EmployeeFactory();

     // Act
     var employee = (InternalEmployee)employeeFactory.CreateEmployee("Kevin", "Dockx");

     // Assert
     Assert.True(employee.Salary >= 2500);
     Assert.True(employee.Salary <= 3500);
}
```

Learning About Assertions

Test if salary is between or equal to 2500 and 3500

```
[Fact]
public void CreateEmployee_ConstructInternalEmployee_SalaryMustBeBetween2500And3500 ()
{
     // Arrange
     var employeeFactory = new EmployeeFactory();

     // Act
     var employee = (InternalEmployee)employeeFactory.CreateEmployee("Kevin", "Dockx");

     // Assert
     Assert.True(employee.Salary >= 2500 && employee.Salary <= 3500);
}
```

Learning About Assertions

Test if salary is between or equal to 2500 and 3500

It’s not about the amount of asserts you’re using in a test, it’s about the behavior you’re testing

```
[Fact]
public void CreateEmployee_ConstructInternalEmployee_SalaryMustBeBetween2500And3500()
{

    // Arrange
    var employeeFactory = new EmployeeFactory();

    // Act
    var employee = (InternalEmployee)employeeFactory.CreateEmployee("Kevin", "Dockx");

    // Assert
    Assert.True(employee.Salary >= 2500 && employee.Salary <= 3500);
    Assert.True(employee.SuggestedBonus > 5000);
}
```

Learning About Assertions

This test is not ok: it tests two types of behavior in one test

Demo

Asserting on booleans

```
Assert.False(...);

Assert.True(...);
```

Asserting on Booleans

Condition can be a simple property or a larger statement that evaluates to true or false

Demo

Asserting on strings

```
Assert.Equal(...) / Assert.NotEqual(...)

Assert.StartsWith(...) / Assert.EndsWith(...)

Assert.Contains(...) / Assert.DoesNotContain(...)

Assert.Matches(...) / Assert.DoesNotMatch(...)

Assert.Empty(...) / Assert.NotEmpty(...)
```

Asserting on Strings

Demo

Asserting on numeric values

Demo

Asserting on floating points with precision

```
Assert.Equal(...) / Assert.NotEqual(...)

Assert.InRange(...)
```

Asserting on Numeric Values (Including Floating Points)

Potentially pass through precision (via an overload) when comparing floating point numbers

Demo

Introducing a repository implementation with test data

Demo

Asserting on arrays and collection content

```
Assert.Equal(...) / Assert.NotEqual(...)

Assert.Contains(...) / Assert.DoesNotContain(...)

Assert.All(...)
```

Asserting on Arrays and Collection Content

Demo

Asserting asynchronous code

Demo

Asserting on exceptions

Asserting on Exceptions

Giving an internal employee a raise

- 100 is the minimum raise
- A minimum raise cannot be given twice in a row

Throws `EmployeeInvalidRaiseException`

```
Assert.Throws<T>(...) / Assert.ThrowsAsync<T>(...)

Assert.ThrowsAny<T>(...) / Assert.ThrowsAnyAsync<T>(...)
```

Asserting on Exceptions

`ThrowsAny(Async)<T>` takes derived versions into consideration, while `Throws(Async)<T>` doesn’t

Demo

Asserting on events

```
Assert.Raises<T>(...) / Assert.RaisesAsync<T>(...)

Assert.RaisesAny<T>(...) / Assert.RaisesAnyAsync<T>(...)
```

Asserting on Events

`RaisesAny(Async)<T>` takes derived event arguments into consideration, while `Raises(Async)<T>` doesn’t

Demo

Asserting on object types

```
Assert.IsType<T>(...) / Assert.IsNotType<T>(...)

Assert.IsAssignableFrom<T>(...) / Assert.IsNotAssignableFrom<T>(...)
```

Asserting on Object Types

Asserting on Private Methods

A private method is an implementation detail that doesn't exist in isolation

- Test the behavior of the method that uses the private method

Making a private method public just to be able to test it breaks encapsulation

- Use `[InternalsVisible]` as a slightly less bad alternative

Summary

Asserts allow you to evaluate and verify the outcome of a test

- Fails when one or more asserts fail
- Passes when all asserts pass

Up Next:

Setting Up Tests and Controlling Test Execution