Модульное тестирование кода C# с использованием NUnit и .NET Core

Этот учебник описывает пошаговую процедуру по созданию примера решения для изучения концепций модульного тестирования. Если при изучении учебника вы предпочитаете использовать готовое решение, просмотрите или скачайте пример кода перед началом работы. Инструкции по загрузке см. в разделе Просмотр и скачивание примеров.

Предварительные требования

Пакет SDK для .NET Core. Текстовый редактор или редактор кода по вашему выбору.

Создание исходного проекта

Откройте окно оболочки. Создайте каталог с именем `unit-testing-using-nunit` для хранения решения. В этом каталоге выполните следующую команду, чтобы создать файл решения для библиотеки классов и тестового проекта:

```
dotnet new sln
```

Затем создайте каталог `PrimeService`. Ниже приведена актуальная структура каталогов и файлов:

```
/unit-testing-using-nunit
    unit-testing-using-nunit.sln
    /PrimeService
```

Перейдите в каталог `PrimeService` и выполните следующую команду, чтобы создать исходный проект:

```
dotnet new classlib
```

Переименуйте Class1.cs в PrimeService.cs. Создайте сбойную реализацию класса PrimeService:

```
using System;

namespace Prime.Services
{
    public class PrimeService
    {
        public bool IsPrime(int candidate)
        {
            throw new NotImplementedException("Please create a test first.");
        }
```

```
    }
}
```

Вернитесь в каталог `unit-testing-using-nunit`. Чтобы добавить проект библиотеки классов в решение, выполните следующую команду:

```
dotnet sln add PrimeService/PrimeService.csproj
```

Создание тестового проекта

Затем создайте каталог `PrimeService.Tests`. Ниже представлена структура каталогов:

```
/unit-testing-using-nunit
    unit-testing-using-nunit.sln
    /PrimeService
        Source Files
        PrimeService.csproj
    /PrimeService.Tests
```

Перейдите в каталог `PrimeService.Tests` и создайте проект, выполнив следующую команду:

`dotnet new nunit`

Команда `dotnet new` создает тестовый проект, который использует NUnit в качестве библиотеки тестов. Созданный шаблон настраивает средство запуска тестов в файле *PrimeServiceTests.csproj*:

```
<ItemGroup>
  <PackageReference Include="nunit" Version="3.13.3" />
  <PackageReference Include="NUnit3TestAdapter" Version="4.5.0" />
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="17.7.2" />
</ItemGroup>
```

Тестовый проект требует других пакетов для создания и выполнения модульных тестов. Команда Команда `dotnet new` на предыдущем шаге добавила пакет Microsoft Test SDK, платформу тестирования NUnit и адаптер тестирования NUnit. Теперь добавьте в проект библиотеку классов `PrimeService` в качестве еще одной зависимости. Использование команды `dotnet add reference`:

```
dotnet add reference ../PrimeService/PrimeService.csproj
```

Ниже показан окончательный макет решения:

```
/unit-testing-using-nunit
    unit-testing-using-nunit.sln
    /PrimeService
        Source Files
        PrimeService.csproj
    /PrimeService.Tests
        Test Source Files
        PrimeService.Tests.csproj
```

Выполните следующую команду в каталоге *unit-testing-using-nunit*:

```
dotnet sln add ./PrimeService.Tests/PrimeService.Tests.csproj
```

Создание первого теста

Вы пишете один неудачный тест, делаете его пройденным, а затем повторяете процесс. В каталоге *PrimeService.Tests* переименуйте файл *UnitTest1.cs* в *PrimeService_IsPrimeShould.cs* и замените его содержимое следующим кодом:

```
using NUnit.Framework;
using Prime.Services;

namespace Prime.UnitTests.Services
{
    [TestFixture]
    public class PrimeService_IsPrimeShould
    {
        private PrimeService _primeService;

        [SetUp]
        public void SetUp()
        {
            _primeService = new PrimeService();
        }

        [Test]
        public void IsPrime_InputIs1_ReturnFalse()
        {
            var result = _primeService.IsPrime(1);

            Assert.IsFalse(result, "1 should not be prime");
        }
    }
}
```

Атрибут `[TestFixture]` обозначает класс, содержащий модульные тесты. Атрибут `[Test]` указывает, что метод — это метода теста.

Сохраните этот файл и выполните `dotnet test` команду , чтобы создать тесты и библиотеку классов и запустить тесты. Средство запуска тестов NUnit содержит точку входа в программу для выполнения тестов. `dotnet test` запускает средство выполнения тестов с помощью проекта модульного теста, который вы создали.

Тест не будет пройден. Вы еще не создали реализацию. Сделайте тест пройденным, написав самый простой `PrimeService` код в классе, который работает:

```
public bool IsPrime(int candidate)
{
    if (candidate == 1)
    {
        return false;
    }
    throw new NotImplementedException("Please create a test first.");
}
```

Выполните команду `dotnet test` еще раз в каталоге `unit-testing-using-nunit`. Команда `dotnet test` запускает сборку для проекта `PrimeService` и затем для проекта `PrimeService.Tests`. После сборки обоих проектов выполняется этот один тест. Он выполняется.

Добавление дополнительных возможностей

Теперь, когда тест проходит успешно, пора создать дополнительные тесты. Есть еще ряд элементарных случаев с простыми числами: 0, -1. Можно добавить новые тесты с помощью атрибута `[Test]`, но это скоро станет утомительным. Есть другие атрибуты NUnit, которые позволяют создавать наборы похожих тестов. Атрибут `[TestCase]` используется для создания набора тестов, которые выполняют один и тот же код, но имеют разные входные аргументы. С помощью атрибута `[TestCase]` можно указать значения для этих входных аргументов.

Вместо создания тестов примените этот атрибут для создания одного управляемого данными теста. который проверяет несколько значений меньше 2, то есть наименьшего простого числа.

```
[TestCase(-1)]
[TestCase(0)]
[TestCase(1)]
public void IsPrime_ValuesLessThan2_ReturnFalse(int value)
{
    var result = _primeService?.IsPrime(value);

    Assert.IsFalse(result, $"{value} should not be prime");
}
```

Выполните команду `dotnet test`, и два из этих тестов завершаются ошибкой. Для успешного выполнения всех тестов нужно изменить предложение `if` в начале метода `Main` в файле *PrimeService.cs*:

```
if (candidate < 2)
```

Продолжайте выполнять итерацию, добавляя дополнительные тесты, теории и код в библиотеку `main`. В результате вы получите готовую версию тестов и полную реализацию библиотеки.

Вы создали небольшую библиотеку и набор модульных тестов для нее. Вы также структурировали решение таким образом, что добавление новых пакетов и тестов является частью стандартного рабочего процесса. и получить возможность сосредоточиться на задачах приложения.