Модульное тестирование C# в .NET Core с использованием dotnet test и xUnit

В этом руководстве описано, как создать решение с проектом модульного теста и проектом исходного кода. Чтобы работать с руководством на примере готового решения, просмотрите или скачайте этот пример кода.

Создание решения

В этом разделе описано, как создать решение, которое содержит проект исходного кода и тестовый проект. Полное решение имеет следующую структуру каталогов:

```
/unit-testing-using-dotnet-test
    unit-testing-using-dotnet-test.sln
    /PrimeService
        PrimeService.cs
        PrimeService.csproj
    /PrimeService.Tests
        PrimeService_IsPrimeShould.cs
        PrimeServiceTests.csproj
```

В инструкциях далее описано, как создать тестовое решение. См. команды для быстрого создания тестового решения и дополнительные инструкции.

- Откройте окно оболочки.
- Выполните следующую команду:

```
dotnet new sln -o unit-testing-using-dotnet-test
```

Команда `dotnet new sln` создает новое решение в каталоге *unit-testing-using-dotnet-test*.

- Теперь перейдите в папку *unit-testing-using-dotnet-test*.
- Выполните следующую команду:

```
dotnet new classlib -o PrimeService
```

Команда `dotnet new classlib` создает проект библиотеки классов в папке *PrimeService* . Новая библиотека классов будет содержать код для тестирования.

- Переименуйте `Class1.cs` в `PrimeService.cs`.
- Замените код файла `PrimeService.cs` на код, приведенный ниже.

```
using System;

namespace Prime.Services
{
    public class PrimeService
    {
        public bool IsPrime(int candidate)
        {
            throw new NotImplementedException("Not implemented.");
        }
    }
}
```

- Предыдущий код:
- - Создает исключение NotImplementedException с сообщением о том, что код не реализован.
- - Мы обновим его позже.

- Выполните приведенную ниже команду в каталоге *unit-testing-using-dotnet-test*, чтобы добавить в решение проект библиотеки классов.

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

- Создайте проект *PrimeService.Tests*, выполнив следующую команду

```
dotnet new xunit -o PrimeService.Tests
```

- Предыдущая команда позволяет:

Создает проект *PrimeService.Tests* в каталоге *PrimeService.Tests*. В тестовом проекте используется библиотека тестов `xUnit`. Настраивает средство выполнения тестов, добавляя следующие элементы `<PackageReference />` в файл проекта:

- - Microsoft.NET.Test.Sdk
- - xunit
- - xunit.runner.visualstudio
- - coverlet.collector

- Добавьте тестовый проект в файл решения, выполнив следующую команду:

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

- Добавьте в проект PrimeService.Tests библиотеку классов PrimeService в качестве зависимости:

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

Команды для создания решения

В этом разделе перечислены все команды, описанные в предыдущем разделе. Пропустите этот раздел, если вы уже выполнили действия из предыдущего раздела.

Следующие команды создают тестовое решение на компьютере Windows

```
dotnet new sln -o unit-testing-using-dotnet-test
cd unit-testing-using-dotnet-test
dotnet new classlib -o PrimeService
ren .\PrimeService\Class1.cs PrimeService.cs
dotnet sln add ./PrimeService/PrimeService.csproj
dotnet new xunit -o PrimeService.Tests
dotnet add ./PrimeService.Tests/PrimeService.Tests.csproj reference
./PrimeService/PrimeService.csproj
dotnet sln add ./PrimeService.Tests/PrimeService.Tests.csproj
```

Выполните инструкции по замене кода в файле *PrimeService.cs*, предложенные в предыдущем разделе.

Создание теста

Распространенный подход к разработке на основе тестирования (TDD) заключается в написании теста (сбой) перед реализацией целевого кода. В этом руководстве используется подход TDD. Метод `IsPrime` является вызываемым, но он пока не реализован. Тестовый вызов `IsPrime` завершается ошибкой. При использовании TDD мы создаем тест, который ожидаемо завершается ошибкой. Затем мы обновляем целевой код, чтобы пройти только созданный тест. Этот подход повторяется циклически: сначала вы создаете тест, который не выполняется, а затем обновляете целевой код, чтобы выполнить тест.

Обновите проект *PrimeService.Tests*.

- Удалите *PrimeService.Tests/UnitTest1.cs*.
- Создайте файл *PrimeService.Tests/PrimeService_IsPrimeShould.cs*.
- Замените код файла *PrimeService_IsPrimeShould.cs* кодом, приведенным ниже.

```
using Xunit;
using Prime.Services;

namespace Prime.UnitTests.Services
{
    public class PrimeService_IsPrimeShould
    {
        [Fact]
```

```
        public void IsPrime_InputIs1_ReturnFalse()
        {
            var primeService = new PrimeService();
            bool result = primeService.IsPrime(1);

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

Атрибут `[Fact]` объявляет метод теста, который выполняется средством выполнения тестов. В папке *PrimeService.Tests* выполните `dotnet test`. Команда `dotnet test` компилирует оба проекта и выполняет тесты. Средство запуска тестов `xUnit` содержит точку входа в программу для выполнения тестов. `dotnet test` запускает средство выполнения тестов, используя проект модульного тестирования.

Тест возвращает ошибку, так как мы еще не реализовали `IsPrime`. Согласно концепции TDD, нужно создавать только такой код, который позволит пройти этот тест. Обновите `IsPrime`, включив в него следующий код.

```
public bool IsPrime(int candidate)
{
    if (candidate == 1)
    {
        return false;
    }
    throw new NotImplementedException("Not fully implemented.");
}
```

Запустите `dotnet test`. Тест проходит.

Добавьте дополнительные тесты

Добавьте проверку простых чисел для 0 и -1. Вы можете скопировать тест, созданный на предыдущем шаге, и создать копии следующего кода для проверки значений 0 и -1. Но лучше этого не делать, так как доступен более оптимальный способ.

```
var primeService = new PrimeService();
bool result = primeService.IsPrime(1);

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

Копирование кода теста, в котором изменяется только один параметр, приводит к дублированию кода и раздуванию теста. Следующие атрибуты xUnit позволяют создавать набор сходных тестов.

`[Theory]` представляет набор тестов, которые выполняют один и тот же код, но имеют разные входные аргументы. Атрибут `[InlineData]` задает значения для этих входных данных. Чтобы не создавать новые тесты, примените указанные выше атрибуты `xUnit` для создания единой теории. Замените представленный ниже код.

```
[Fact]
public void IsPrime_InputIs1_ReturnFalse()
{
    var primeService = new PrimeService();
    bool result = primeService.IsPrime(1);

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

следующим кодом:

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

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

В приведенном выше коде `[Theory]` и `[InlineData]` позволяют тестировать несколько значений, не превышающих 2. Число 2 является наименьшим простым числом.

Добавьте следующий код после объявления класса и перед атрибутом `[Theory]`:

```
private readonly PrimeService _primeService;

public PrimeService_IsPrimeShould()
{
    _primeService = new PrimeService();
}
```

Выполните команду `dotnet test`, и два из этих тестов завершатся ошибкой. Поместите следующие код в метод `IsPrime`, чтобы успешно пройти все тесты:

```
public bool IsPrime(int candidate)
{
    if (candidate < 2)
```

```
        {
            return false;
        }
        throw new NotImplementedException("Not fully implemented.");
    }
```

Согласно концепции TDD, добавьте новые тесты, которые завершаются ошибкой, а затем обновите целевой код. Вы также можете изучить готовую версию тестов и полную реализацию библиотеки.

Составленный нами метод `IsPrime` не является эффективным алгоритмом для тестирования простых чисел.