**Практическая работа 4**

**на тему: «Работа с заглушками и имитаторами взаимодействия»**

**Задание:** Для приложения типа Nunit Test Project разработать Unit-тест, который тестирует работу класса BoxService. Для теста добавить Stub или Mock объекты в зависимости от того, что тестируем, состояние или взаимодействие.

В тестируемой программе класс BoxService реализует с помощью метода Analize запись ошибок в локальное хранилище в формате строки. Ошибки могут возникать при несоответствии имен файлов определенным требованиям.

Метод тестирования Assert должен проверить, что метод Analize класса BoxService правильно вызывает метод LogError Mock объекта.

**Ход работы:**

**1. Создать приложение по шаблону****Nunit Test Project.**

Запустить на выполнение MS Visual Studio 20xx. Чтобы создать проект по шаблону Nunit Test Project нужно вызвать следующую последовательность команд

**File -> New -> Project...**

В результате откроется окно New Project. В окне выбрать шаблон Nunit Test Project как изображено на рисунке 1.

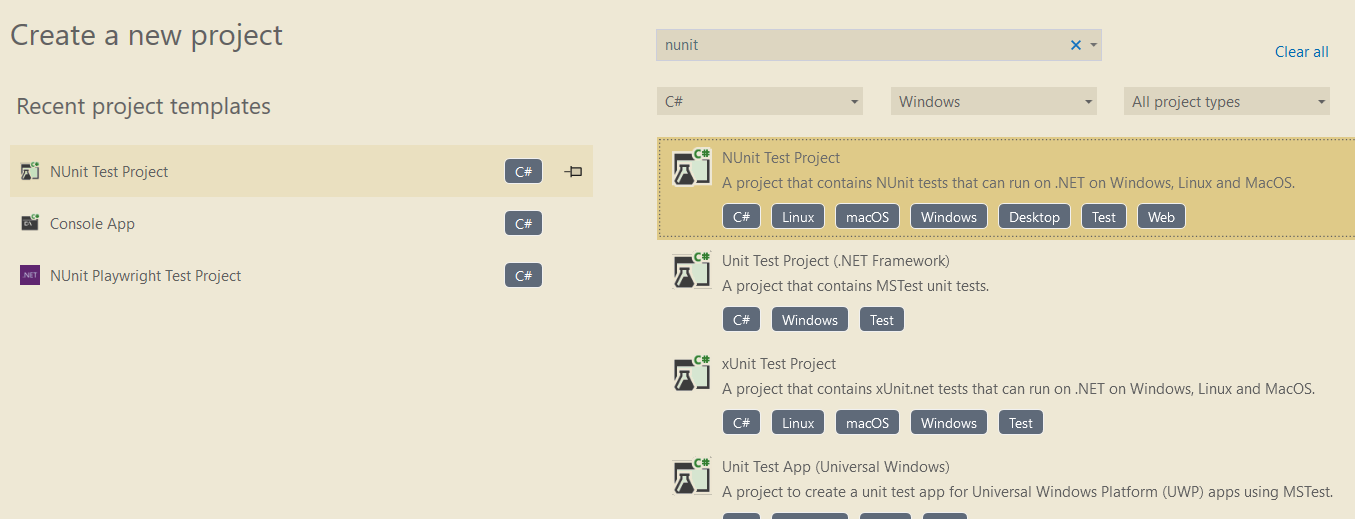


Рис. 1. Окно «New Project». Выбор приложения типа **Nunit Test Project**

**2. Начальные установки**

**2.1 Реализация шаблона проекта.**

Название проекта по выбору. В примере проект назван MocksAndStubsUnitTest.

Тесты пишем в файле UnitTest1.cs, потом будет переименован в BoxServiceTest.cs с одноименным тестирующим классом в нем.

Бизнес логику приложения пишем в файлах-классах в проекте по другому шаблону.

Пример начального кода изображен на рисунке 2.

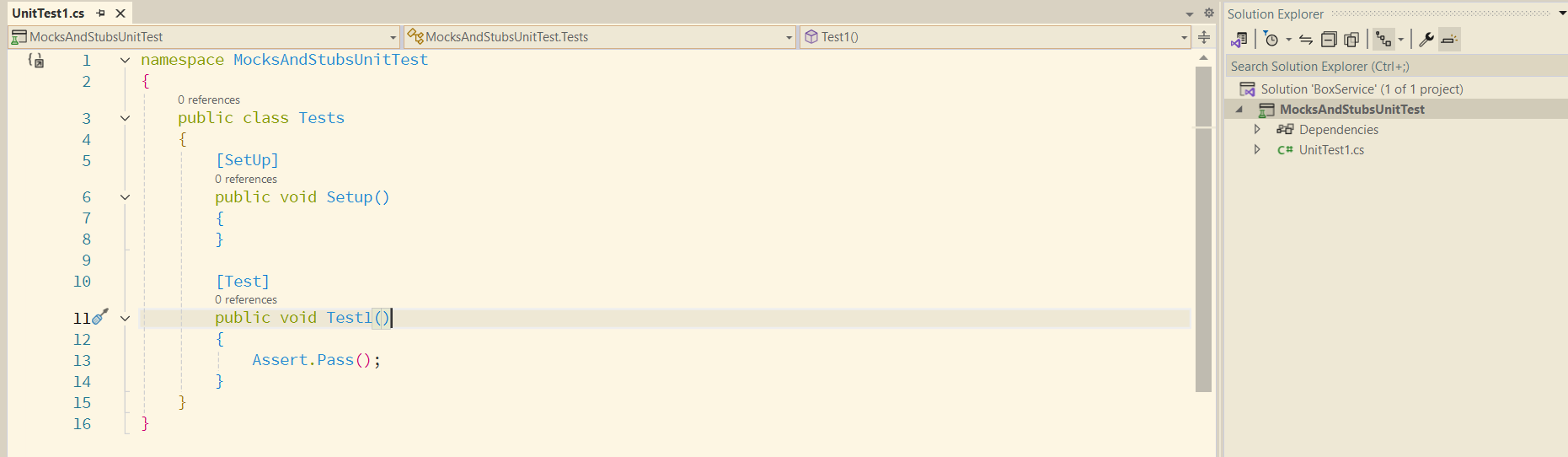
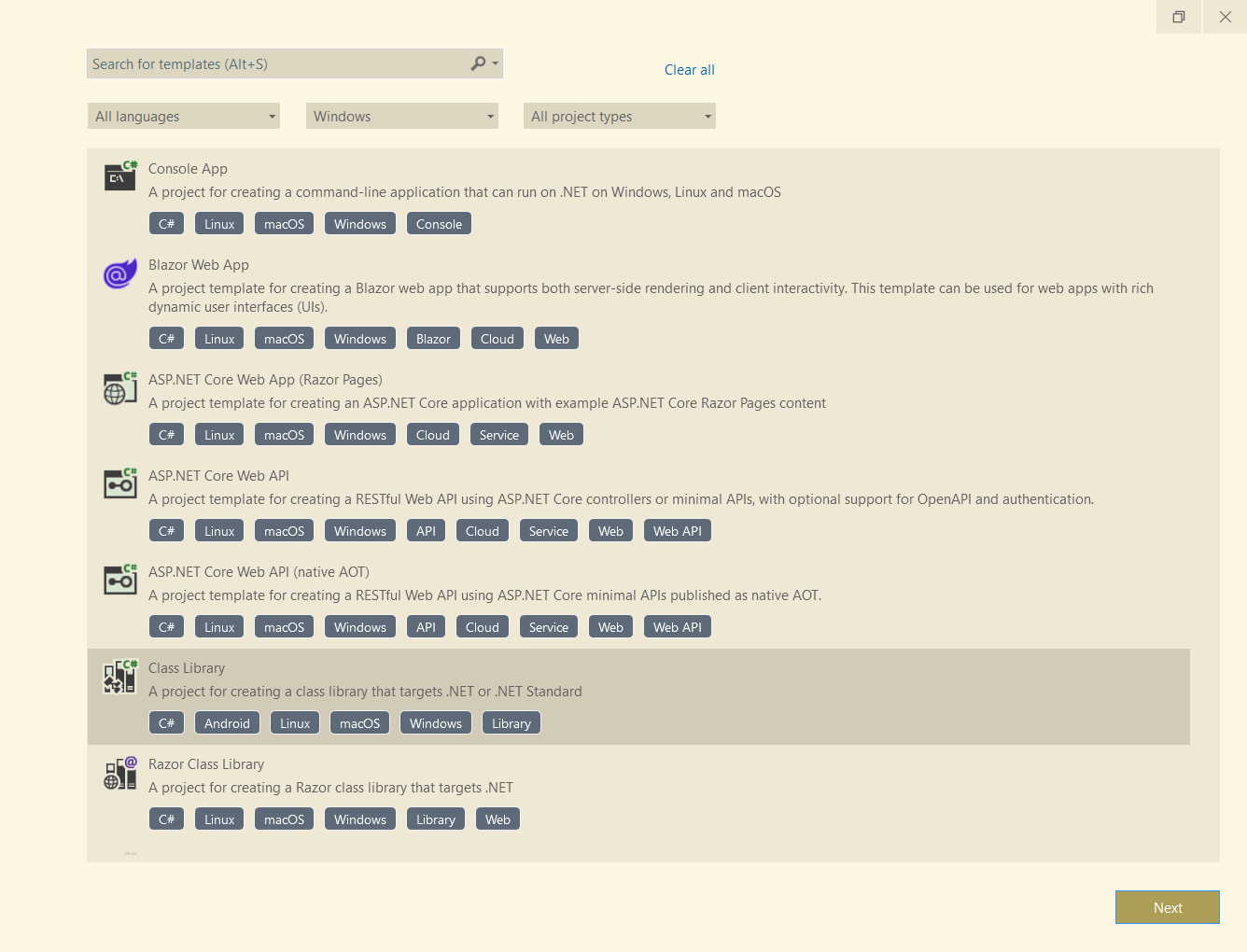
****

Рис. 2. Вид окна MS Visual Studio 2022, модуль « UnitTest1.cs »

**2.2. Добавим в решение проект с тестируемой программой BoxService.**

(примечание: в отдельных случаях в учебных целях допускается реализация приложения с тестирующим классом в рамках одного проекта)

например, библиотеку классов на основе .NETCoreApp:



В приложении будут модули BoxService.cs, ILogService и MockLogService. См. рис.3

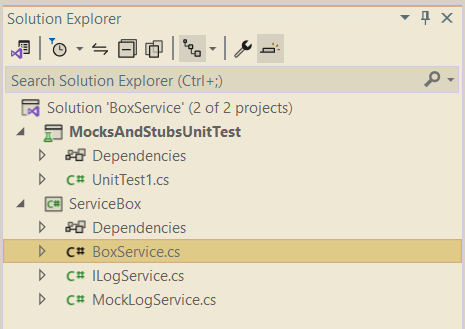
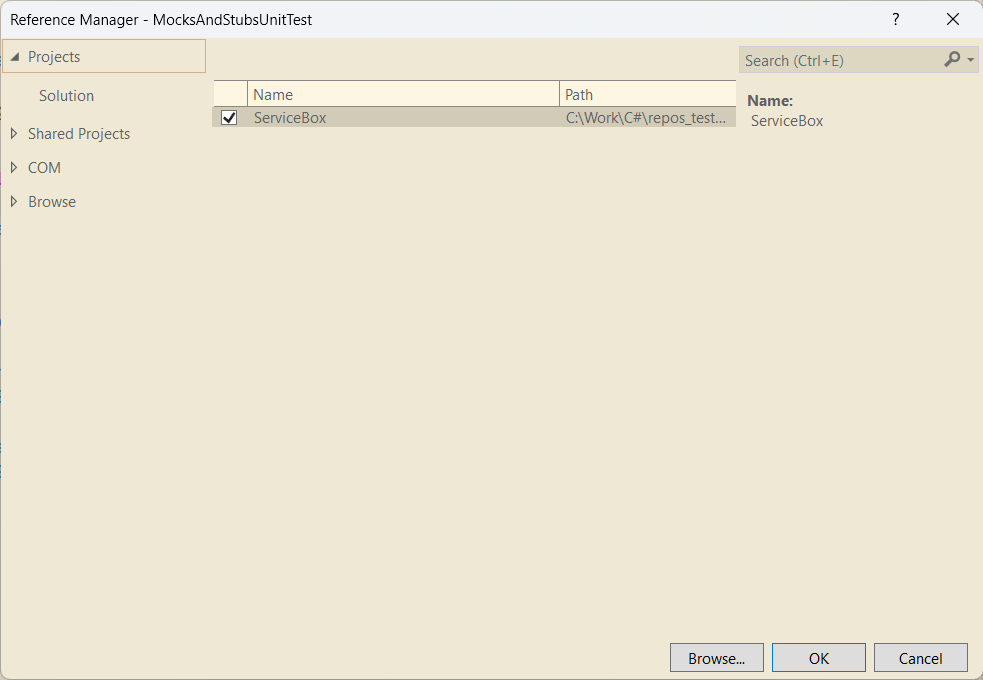


Рис. 3. Файловая структура проекта в MS Visual Studio 2022

Подключим проект ServiceBox к проекту MocksAndStubsUnitTest:



**3. Работа модулей в тесте.**

**3.1. Текст программы, которую нужно протестировать**

На данный момент текст программы, которую нужно протестировать, имеет вид:

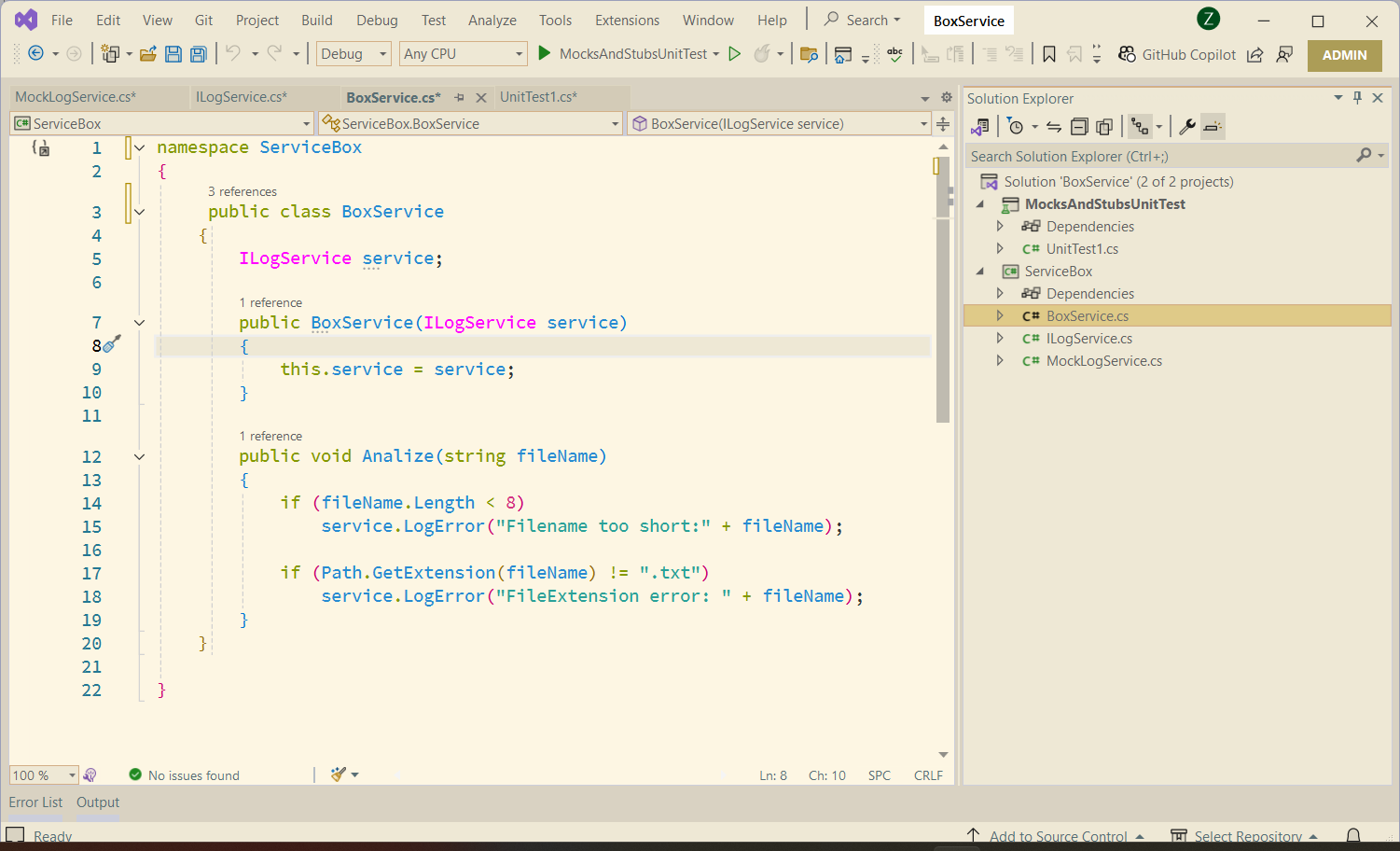


Рис. 3. Код модуля BoxService.cs

В этом примере мы изначально наполним класс BoxService следующим функционалом:

Добавим метод Analize, который:

* принимает имя какого-то файла, проверяет, если у этого файла расширение fileName != ".txt" ;
* используя некий сервис записывает в некий журнал ошибок сообщения ошибки "FileExtension error: " + fileName (ошибка разрешения файла);
* аналогично проверяет длину файла, если меньше 8 символов, пишет в логи "Filename too short:" + fileName

При этом метод Analize использует внешнюю зависимость – экземпляр **service,** который реализует интерфейс **ILogService.** Егонадо заменить неким подставным объектом. Это может быть **MOCK** объект или другой.

Внешняя зависимость **ILogService service** внедряется в качестве параметра конструктора **BoxService**. В таких случаях говорят, что внедрение происходит «через конструктор».

**3.2. Работа класса теста ServiceBoxTests.cs при тестировании модуля BoxService**

Если разобраться, то мы понимаем, что для тестирования метода Analize нужно использовать тестирование взаимодействия, которое проверит, что метод Analize правильно вызывает метод **LogError** объекта **service**, поэтому для замены объекта **service** мы будем использовать **MOCK** объект.

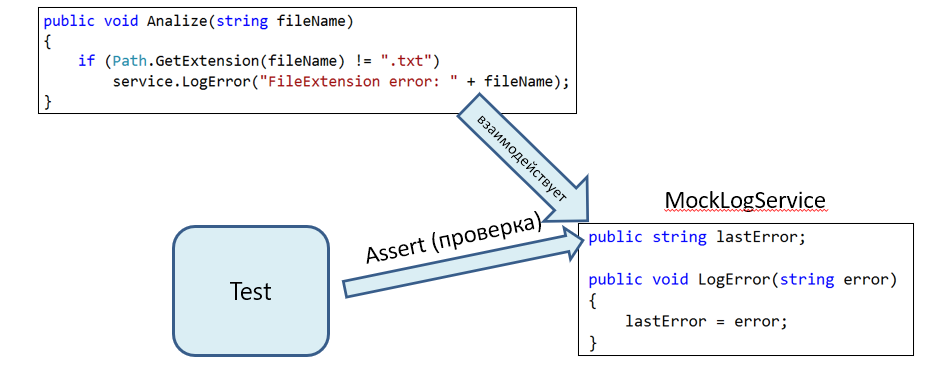


Рис. 5. Схема работы теста ServiceBoxTest

В качестве Mock объекта выступает класс MockLogService, который содержит метод LogError.

Суть метода LogError состоит в сохранении параметра, который передается в метод LogError при его использовании.

Тесты, которые используют объекты Mock, должны быть направлены на проверку состояния именно Mock объекта.

В нашем случае Mock объект будет выглядеть так:

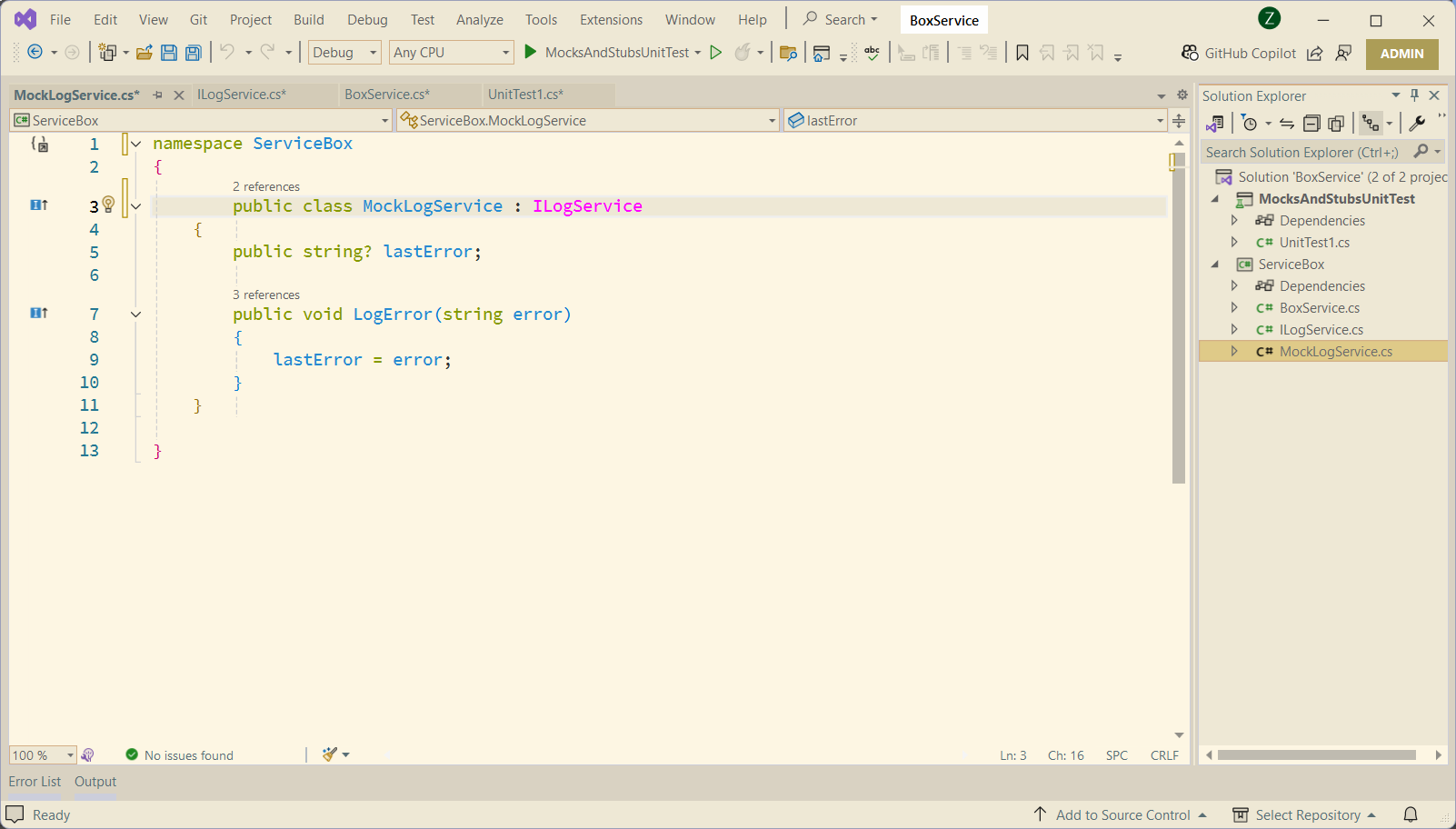


Рис. 6. Код модуля MockLogService.cs

Класс MockLogService реализует интерфейс ILogService (см рис.7). За счет интерфейса мы можем без риска подменить настоящий логгер на Mock объект в конструкторе класса BoxService.

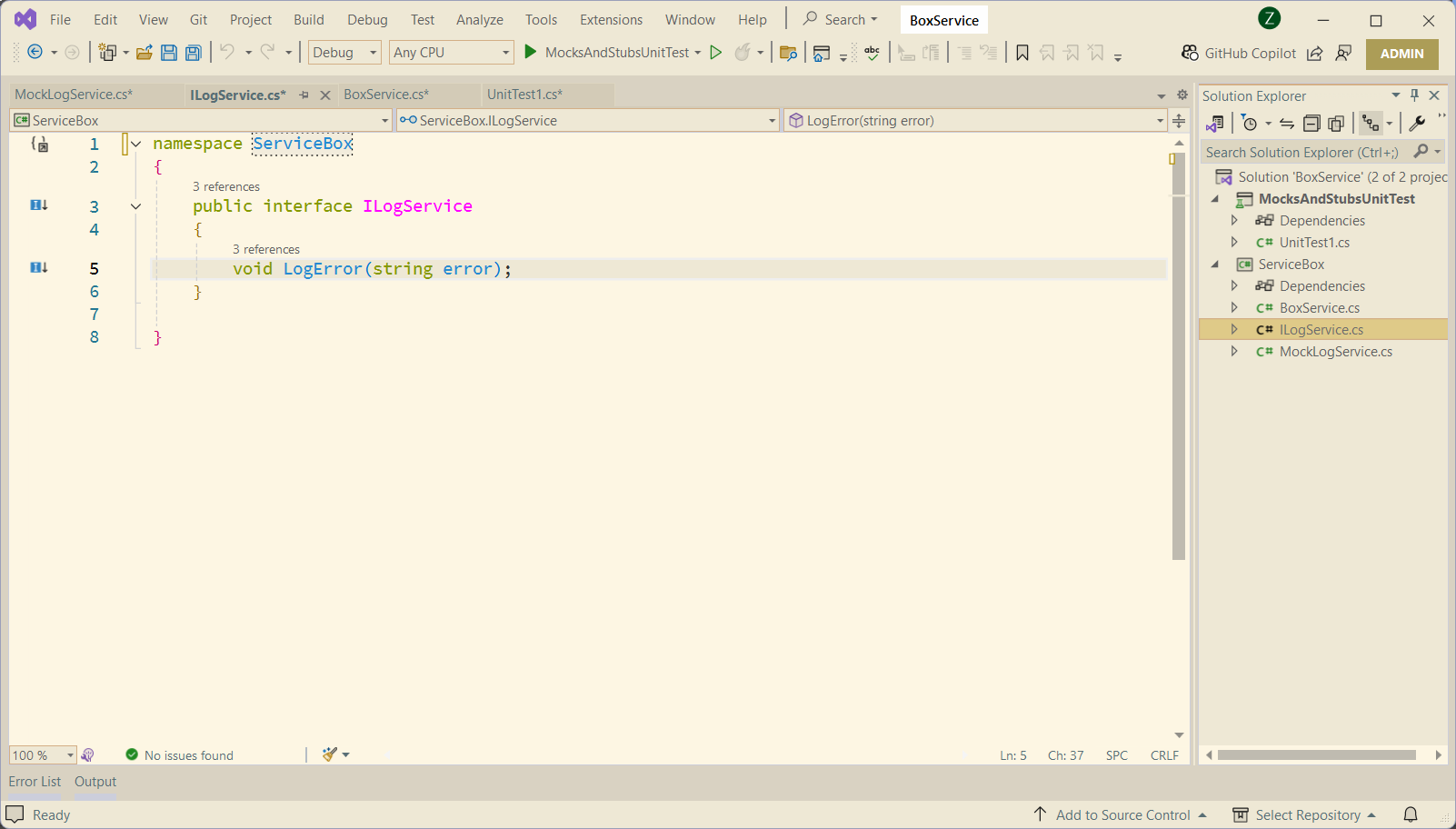


Рис. 7. Код модуля ILogService.cs

**Структура тестирующего класса.**

Поменяем название модуля-теста на ServiceBoxTest и заменим имя пространства имен на ServiceBox.Test.

Теперь проверка взаимодействия класса BoxService с Mock объектом класса MockLogService будет происходить в тестовом классе модуля ServiceBoxTest.cs.

Здесь создается mockService-экземпляр класса MockLogService, а затем box-экземпляр класса BoxService, который принимает зависимость от Mock объекта mockService.

Далее вызываем метод Analize объекта класса BoxService и передаем ему имя файла.

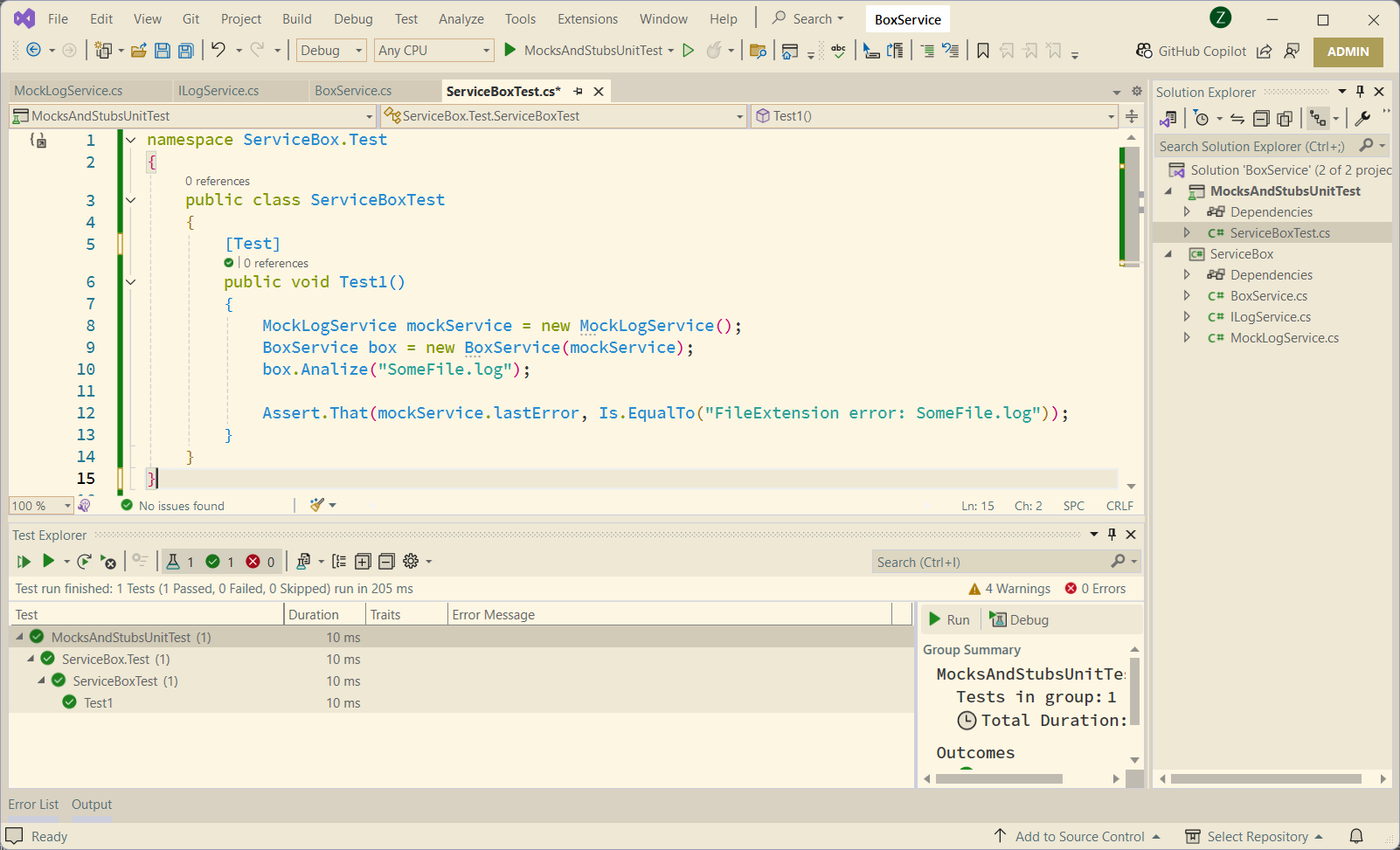


Рис. 7. Код модуля ServiceBoxTest.cs

Имя файла "SomeFile.log" не соответствует нашим условиям в методе Analize, генерируются сообщения об ошибке, которые мы проверяем методом Assert.That() из фреймворка Nunit.

На данный момент наша программа готова к тестированию.

**4. Запуск теста на выполнение и проверка результата тестирования**

В Microsoft Visual Studio 2022 для работы с Unit-тестами реализовано специальное меню команд, которое называется Test.

Чтобы запустить тест на выполнение, нужно выбрать одну из команд в меню, например: **Test -> Run All Tests** как изображено на рисунке 8.

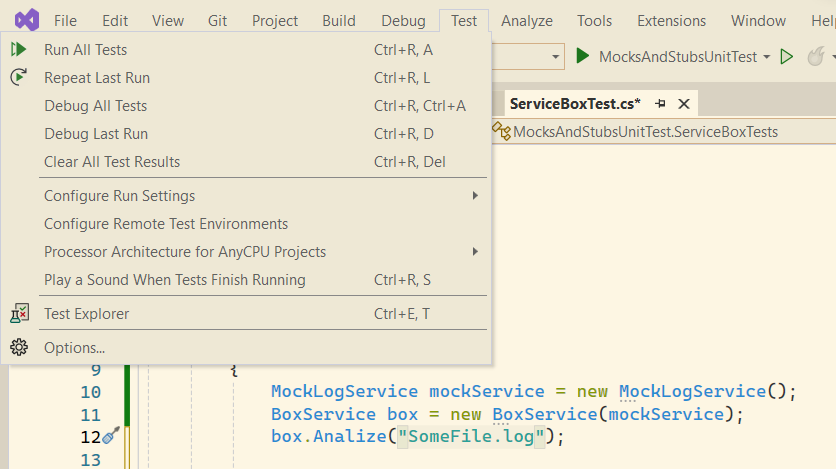


Рис. 8. Вызов команды запуска тестирования

После запуска теста, результат можно просмотреть в окне Test Explorer.

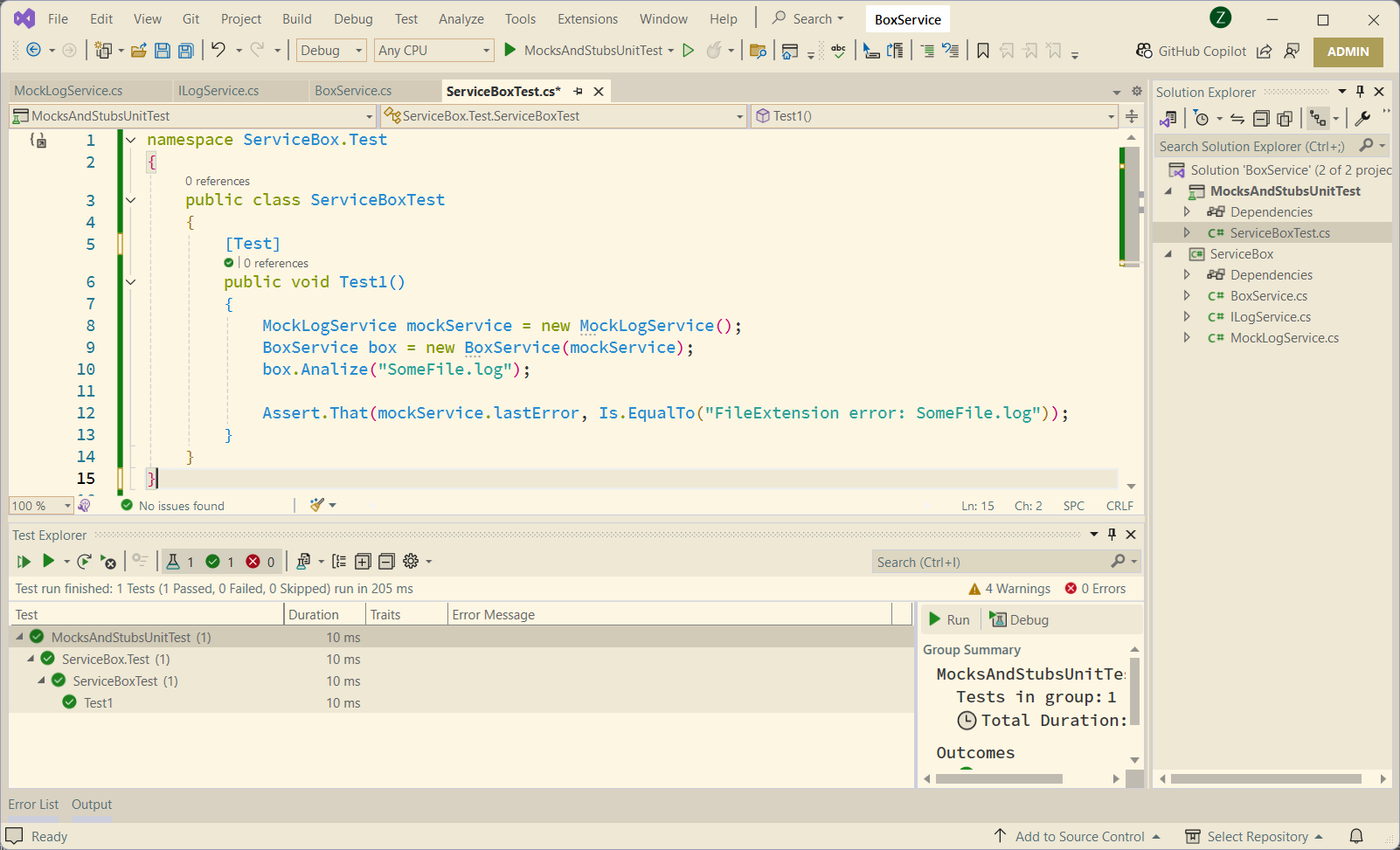


Рис. 9. Просмотр результата тестирования в Test Explorer

Соответственно в окне Test Explorer будет отображен положительный результат теста. После этого можно сделать вывод о том, что взаимодействие класса BoxService с Mock объектом класса MockLogService происходит в тестовом классе модуля ServiceBoxTest.cs правильно.

**Практическая работа № 4.1**

**Задание**

В этом задании будут параллельно использоваться Mock объект и Stub объект.

Дополним метод Analize объекта класса BoxService новым функционалом:

если метод **Analize**  попробует послать объекту **logService** какое-то сообщение и при этом **logService** не будет доступен, то сгенерируется исключительная ситуация и мы пошлём сообщение в техподдержку с сообщением об этой исключительной ситуации с помощью объекта **mailService** (см. рис 10):

* Объект **logService** у нас будет Stub объектом.
* Объект **mailService** у нас будет Mock объектом.

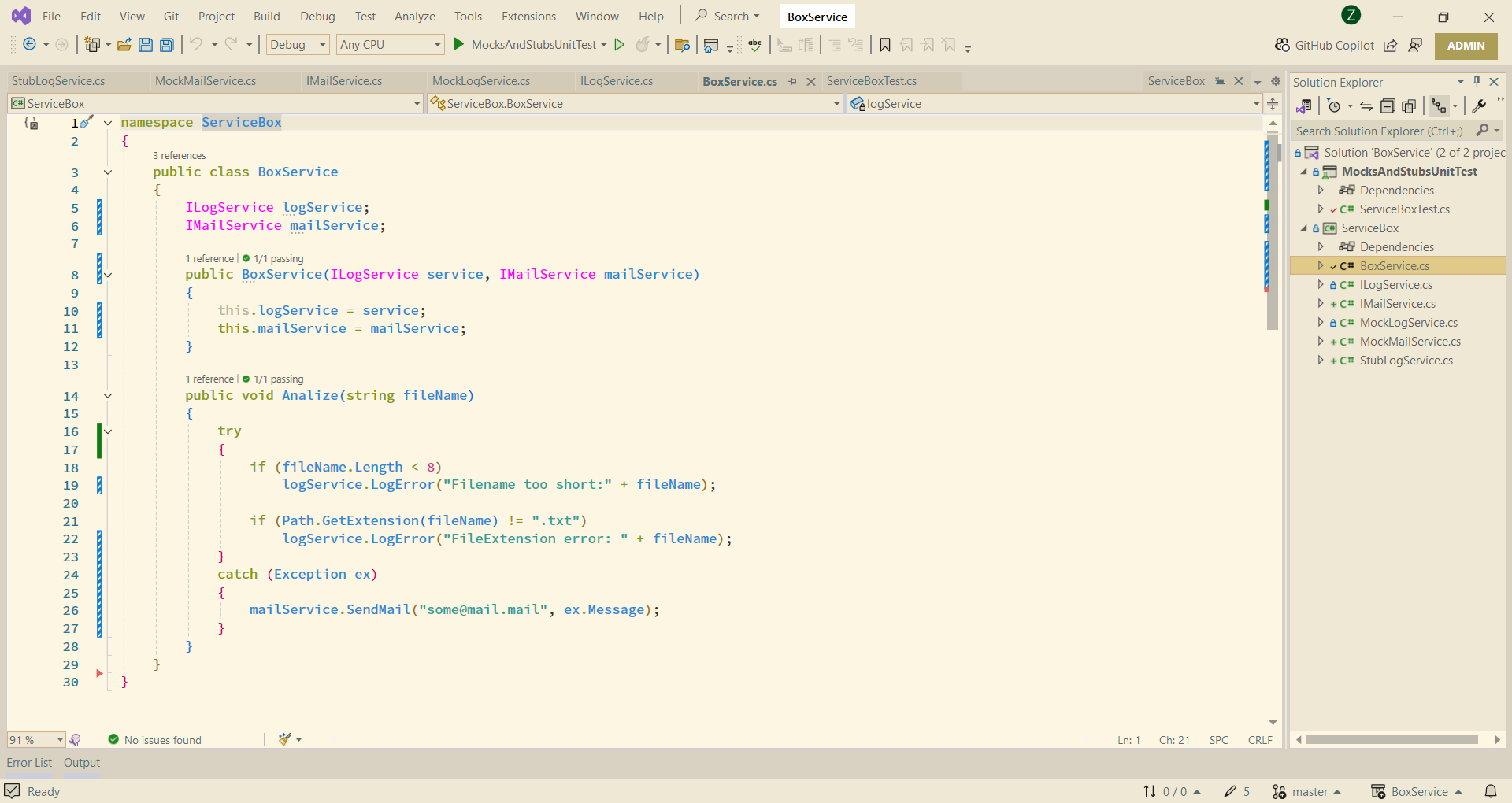


Рис. 10. Код модуля BoxService.cs

Мы будем проверять взаимодействие объекта класса **BoxService** с объектом **mailService.**

Зависимость в класс **BoxService** мы будем внедрять с помощью конструктора.



Рис. 11. Код модуля MockMailService.cs

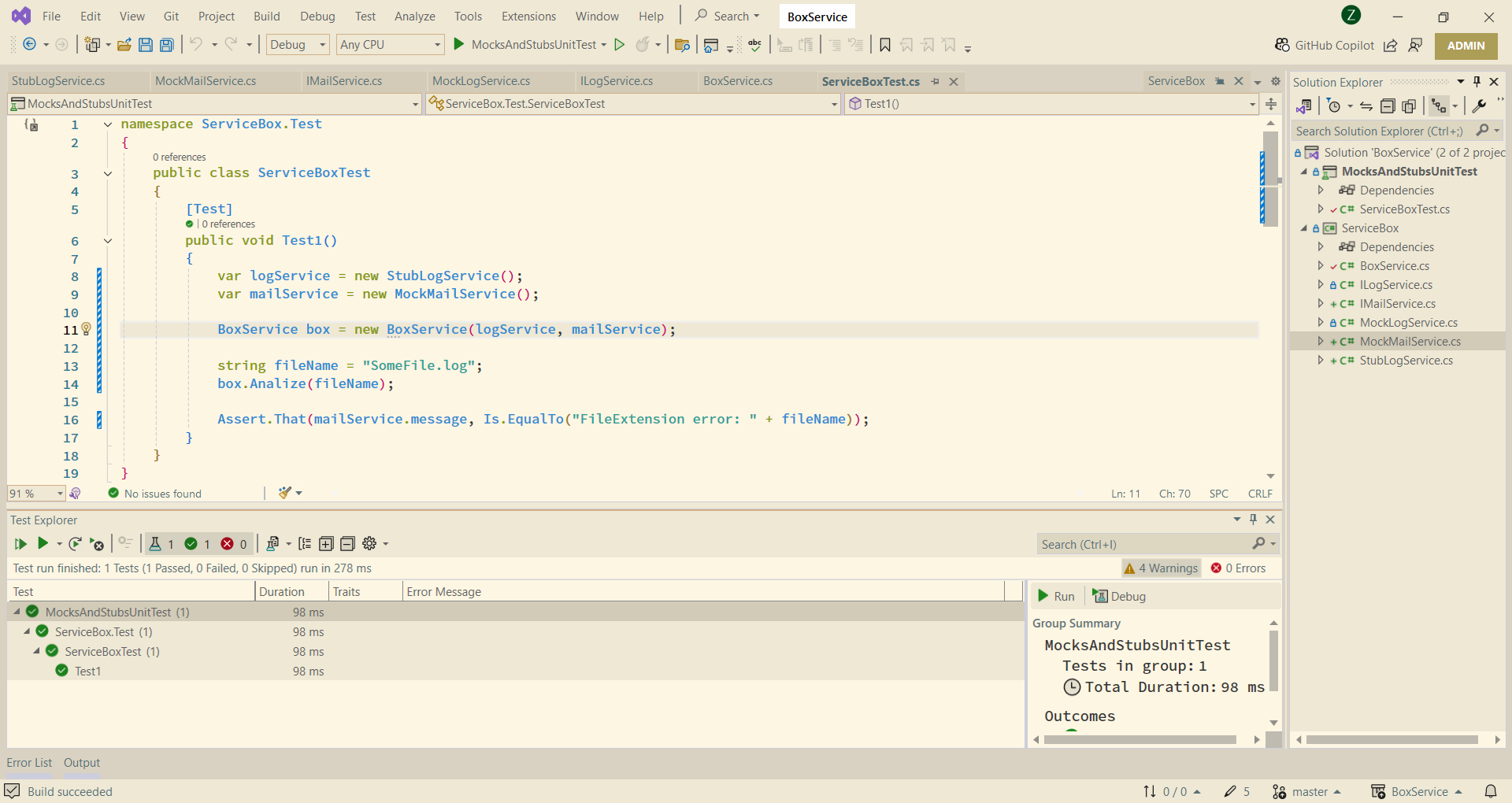


Рис. 12. Код модуля StubLogService.cs

Посмотрим на наши подставные объекты:

|  |  |
| --- | --- |
| **Stub** объект | **Mock** объект |
| **Класс StubLogService** | **Класс MockMailService** |
| оба реализуют интерфейсы | |
| **ILogService** | **IMailService** |
| оба содержат простые методы имитирующие работу | |
| генерации исключения (**LogError**)  с текстом той ошибки, которую мы передаём этому методу | отсылку сообщения (**SendMail**)  в техподдержку |

Теперь давайте перейдём к тесту методу и посмотрим как он реализован.

Рис. 13. Код модуля ServiceBoxTest.cs, просмотр результата тестирования в Test Explorer

1. создали тестовый метод Test1,
2. создали зависимости от объектов классов **StubLogService** и **MockMailService,**
3. создали экземпляр класса BoxService и внедрили в него зависимости от объектов классов StubLogService и MockMailService ,
4. подготовили имя файла, который хотим проанализировать,
5. вызвали метод Analize, который через метод LogError будет генерировать исключительную ситуацию с сообщением "FileExtension error: " и конкатенацией с именем файла (дополнительно см. рис.10), кроме того через конструкцию «try catch» мы отловим эту исключительную ситуацию и вызовем метод SendMail с текстом исключения;
6. вызываем тестовый метод Assert.That и проверяем что BoxService успешно провзаимодействовал с MailService.

Результат тестирования: Тест успешно пройден.

**Практическая работа № 4.2**

**Задание**.

1. В продолжение расширения функциональности предыдущего примера разработать программу, которая :

* реализует взаимодействие объекта класса BoxService с **StubLogService** и **MockMailService** (уже реализовано в предыдущем примере);
* реализует работу объекта класса BoxService при участии объекта класса **DataAccess**, реализующего получение коллекции имен файлов, извлекаемых из некой Базы Данных;
* реализует взаимодействие объекта класса BoxService с объектом класса **ReportManager**, реализующим сохранение на некоем файловом ресурсе отчета о полученной в предыдущем пункте данного Задания коллекции имен файлов.

Разработку производить с применением Stub и Mock объектов имитирующих работу неких реальных объектов. Самостоятельно определить в каком случае необходима заглушка, а в каком имитатор.

1. Произвести тестирование взаимодействия объекта класса BoxService с имитаторами с помощью метода Assert библиотеки Nunit.

Таким образом, для тестирования разных взаимодействий надо создать несколько методов Assert в тестовом классе. И, возможно, придется внести изменения в предыдущий код, поскольку новые объекты повлияют на реализацию объектов классов StubLogService и MockMailService.

Успешное прохождение тестов будет указывать на правильно написанный код взаимодействия объектов. Необходимые проверки:

