**Лабораторная работа №3.**

**Тестирование взаимодействия и подставные объекты**

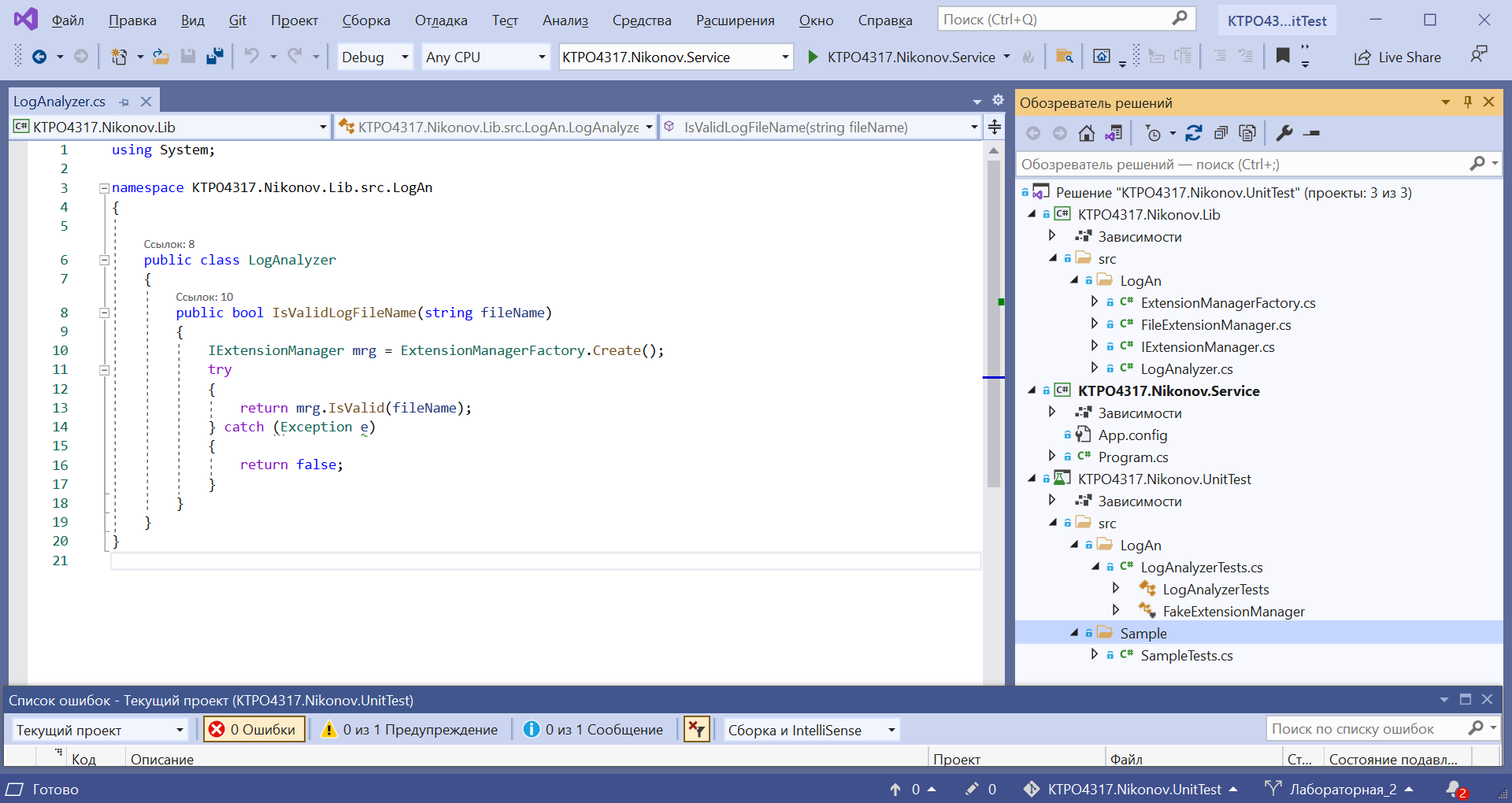
**Цель работы:**

Приобретение практических навыков использования подставных объектов, для тестирования взаимодействия при автономном тестировании модулей, практика использования тестового каркаса NUnit, практика внедрения зависимости.

**Ход работы:**

**1. Подготовка проекта**

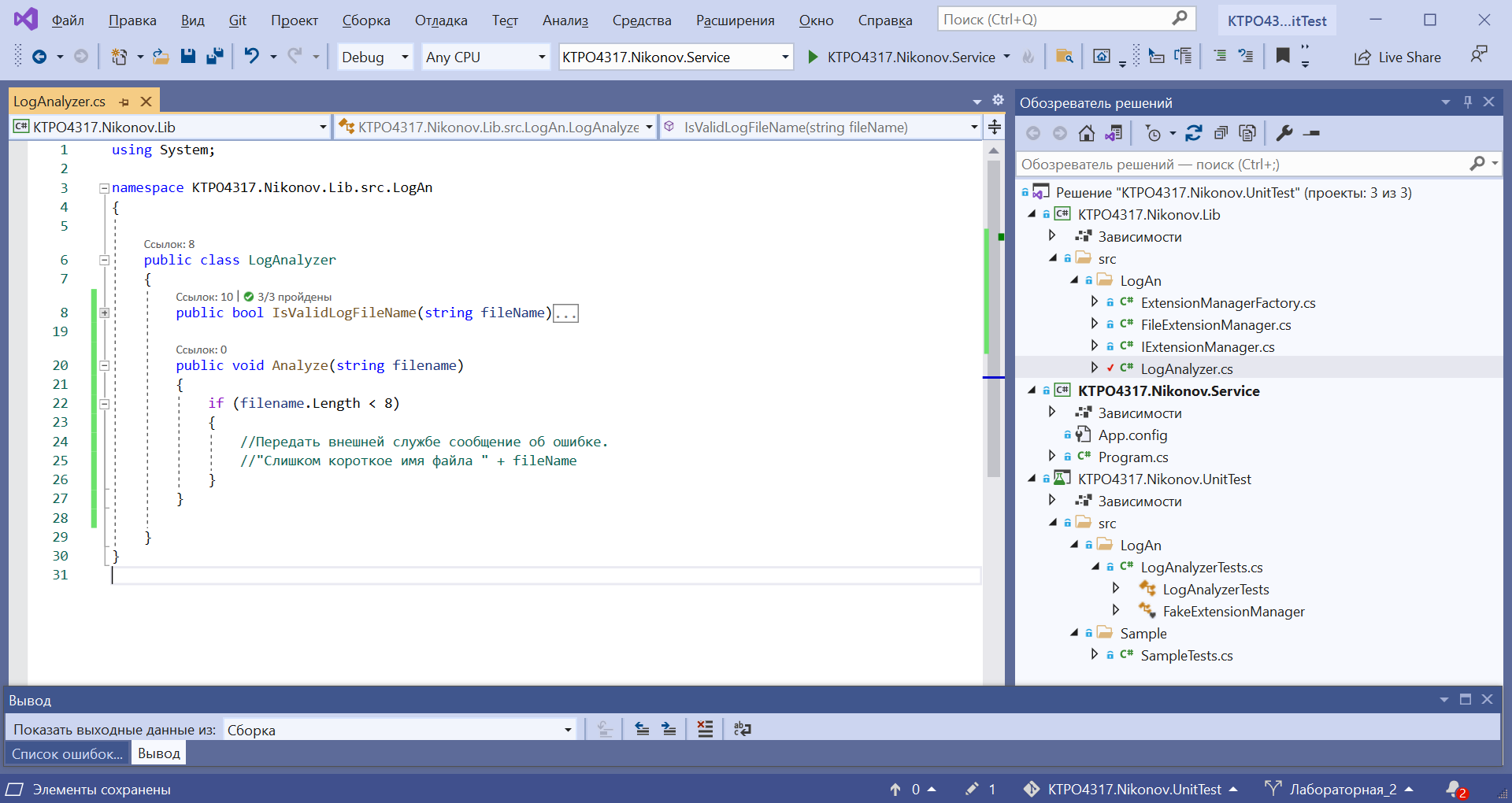
Для выполнения данной лабораторной работы возьмите решение, полученной в результате выполнения лабораторной работы №2. Выполните тесты. Зафиксируйте исходное состояние окна «Обозреватель решения», тестируемого класса и тестов окна «Результаты тестов»



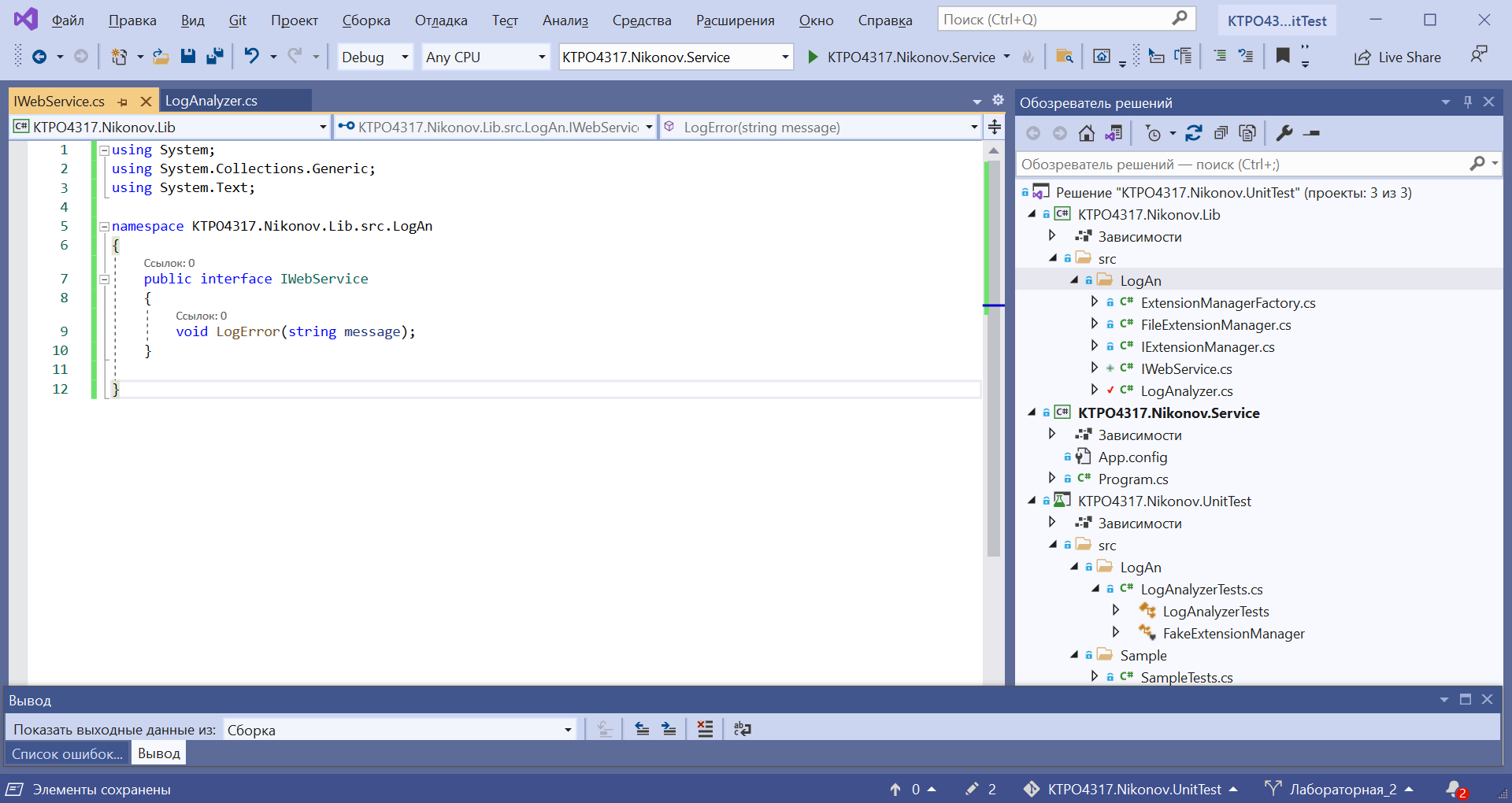


**2. Использование подставного объекта**

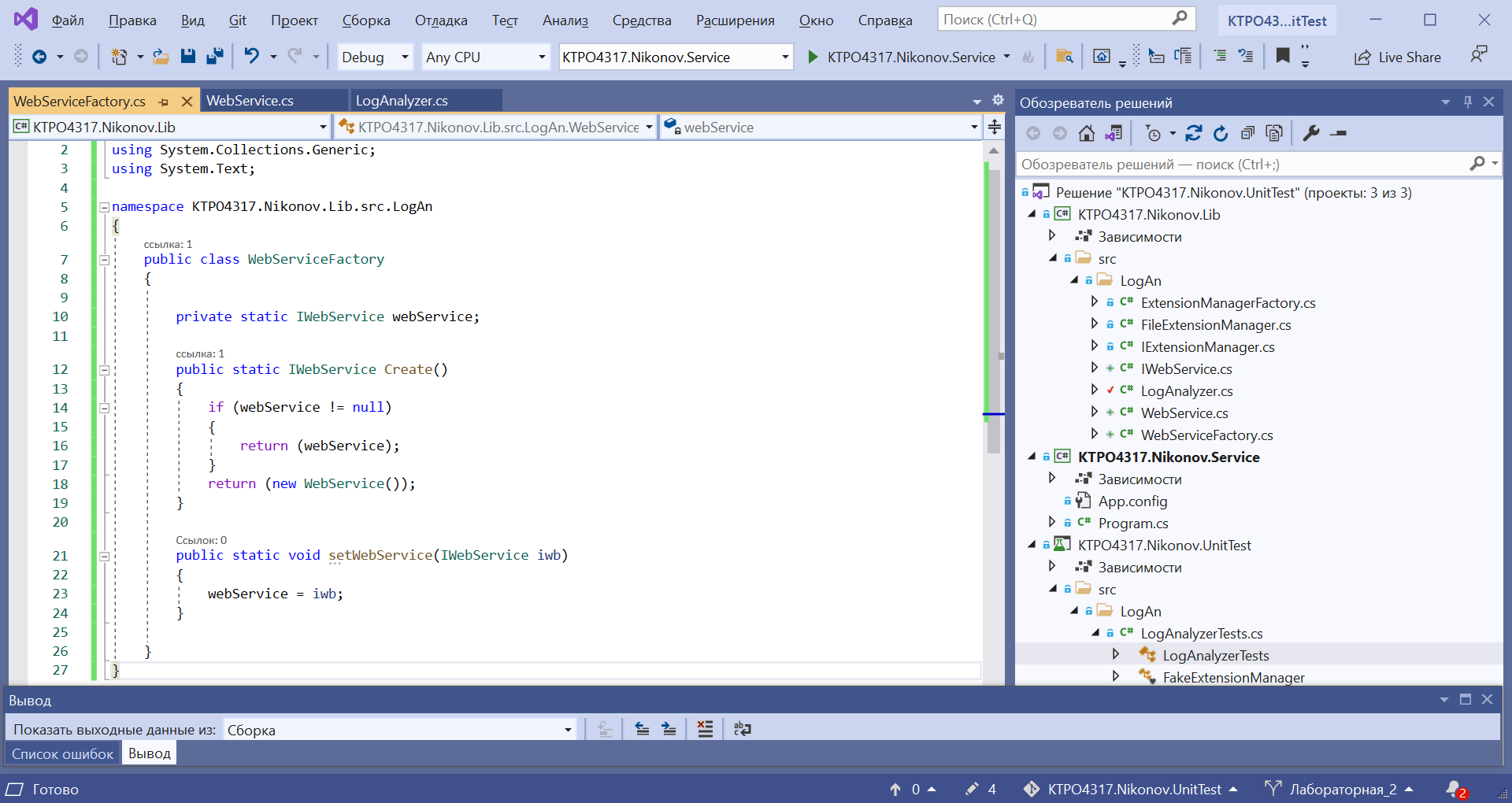
Добавьте в класс LogAnalyzer новый метод Analyze, код для проверки длины файла и взаимодействия с веб-сеорвисов.

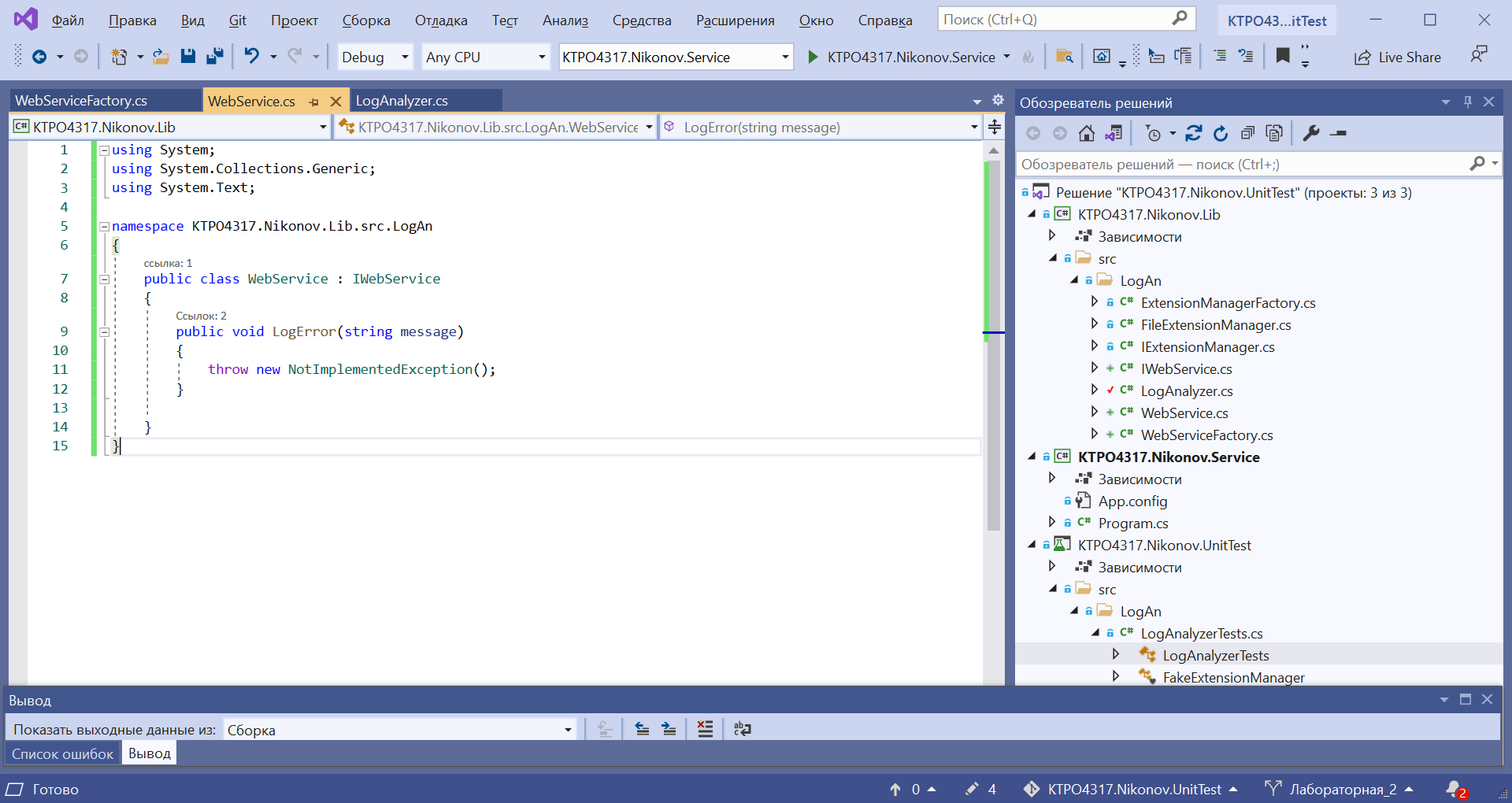


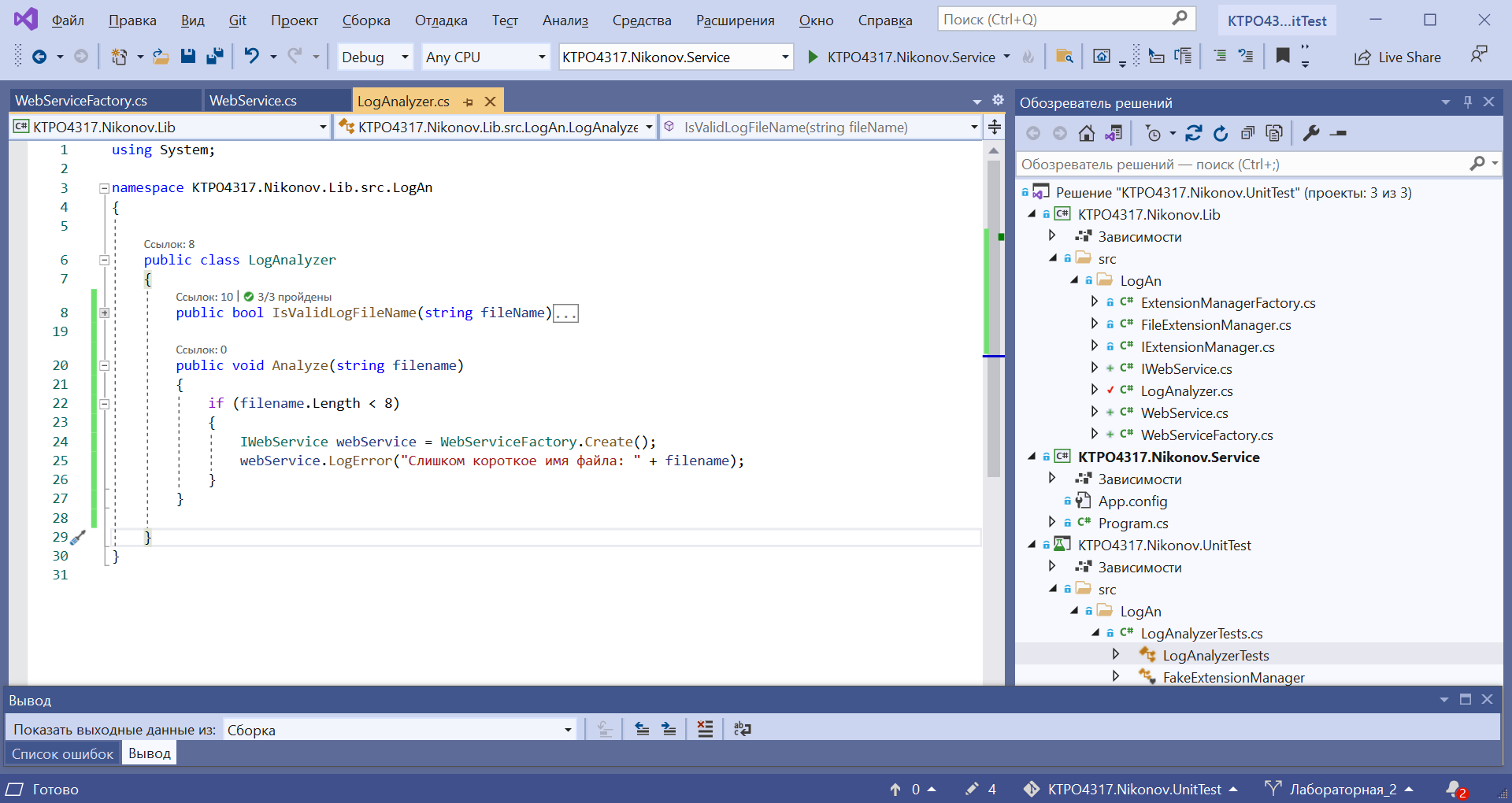
Выделите интерфейс IWebService с одним методом void LogError(string message)



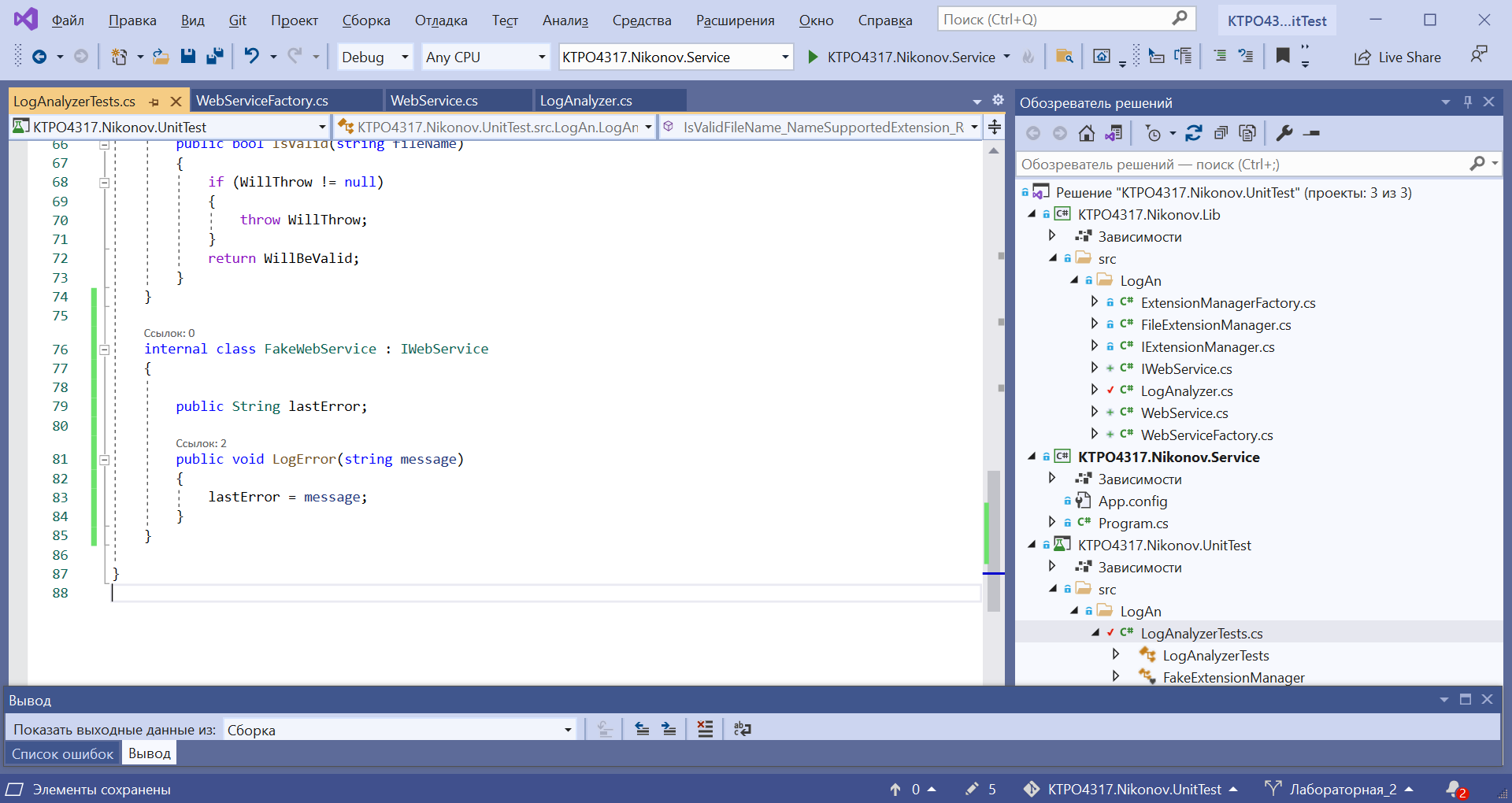
Замените псевдокод на обращение к веб-службе через интерфейс IWebService. Внедрите зависимость используя рассмотренный в лабораторной работе №2 метод внедрения зависимости через фабрику. Воспользуйтесь методом [TearDown] для восстановления исходного состояния фабрики.

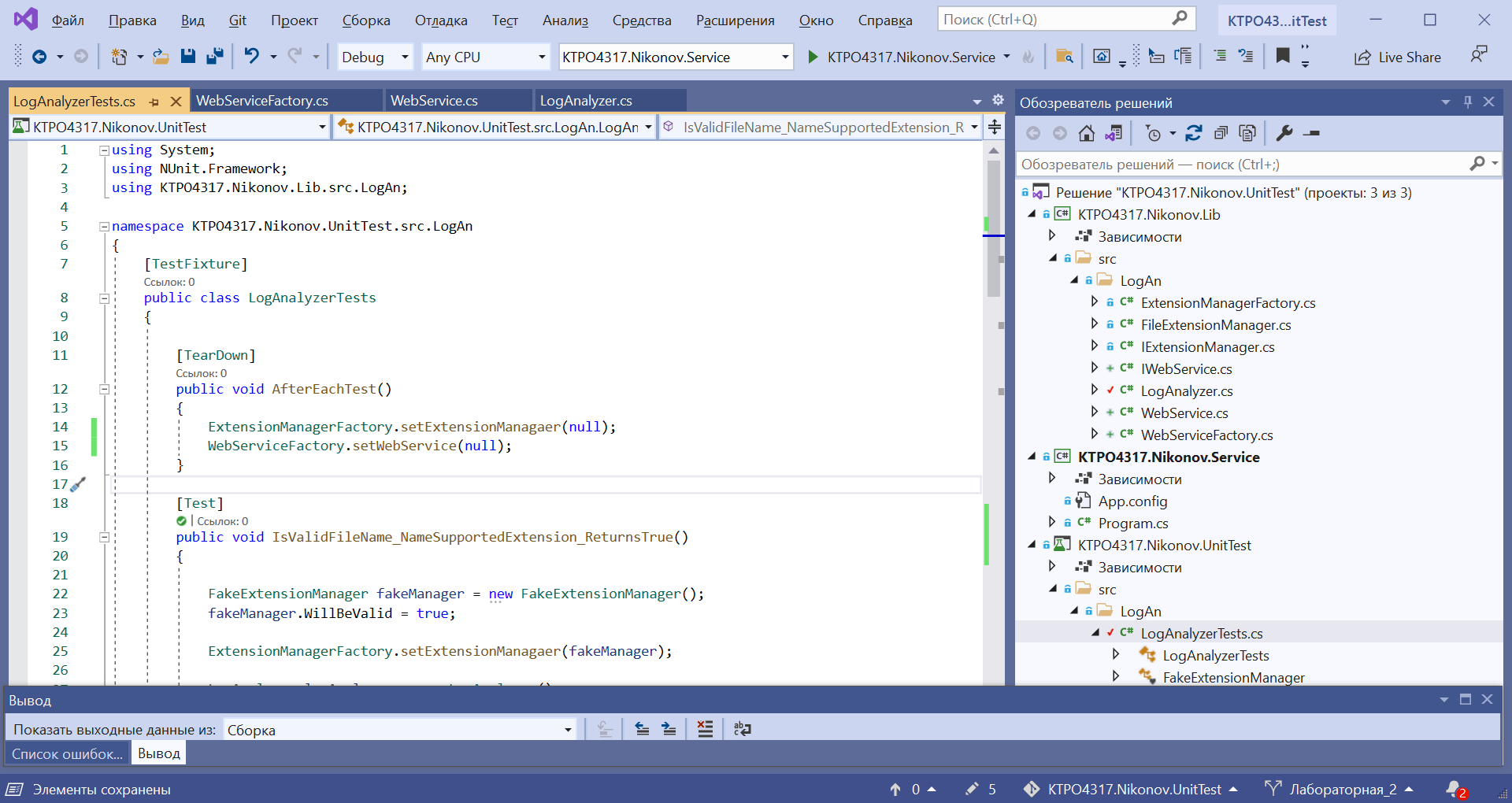






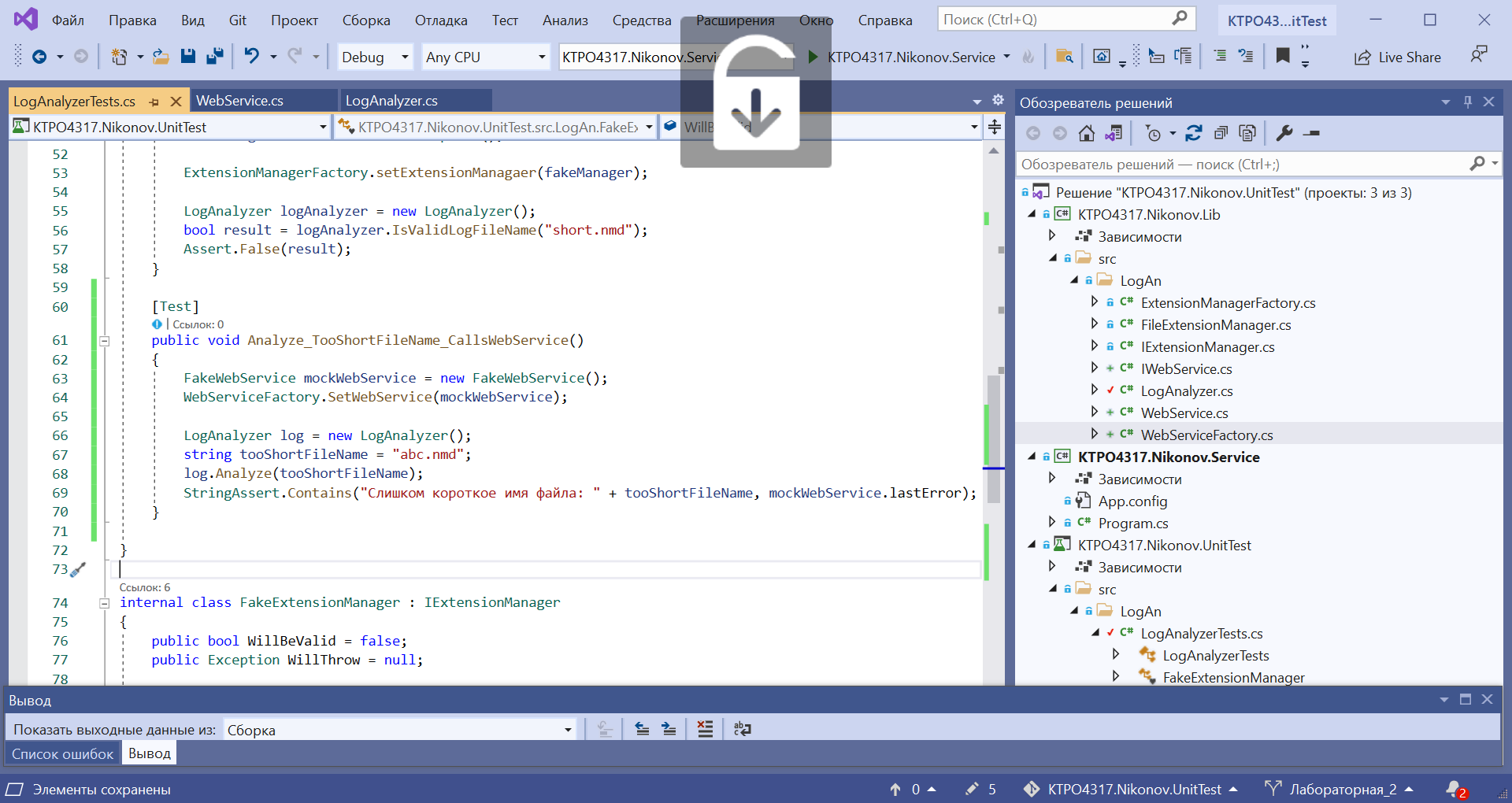
Создайте поддельный объект FakeWebService: IWebService. Этот объект выглядит как заглушка, но содержит дополнительный код, который позволяет сохранить и затем проверить состояние объекта после вызова метода, и сделать утверждение о том, что объект вызван верно. Таким образом объект можно будет использовать в качестве подставки.



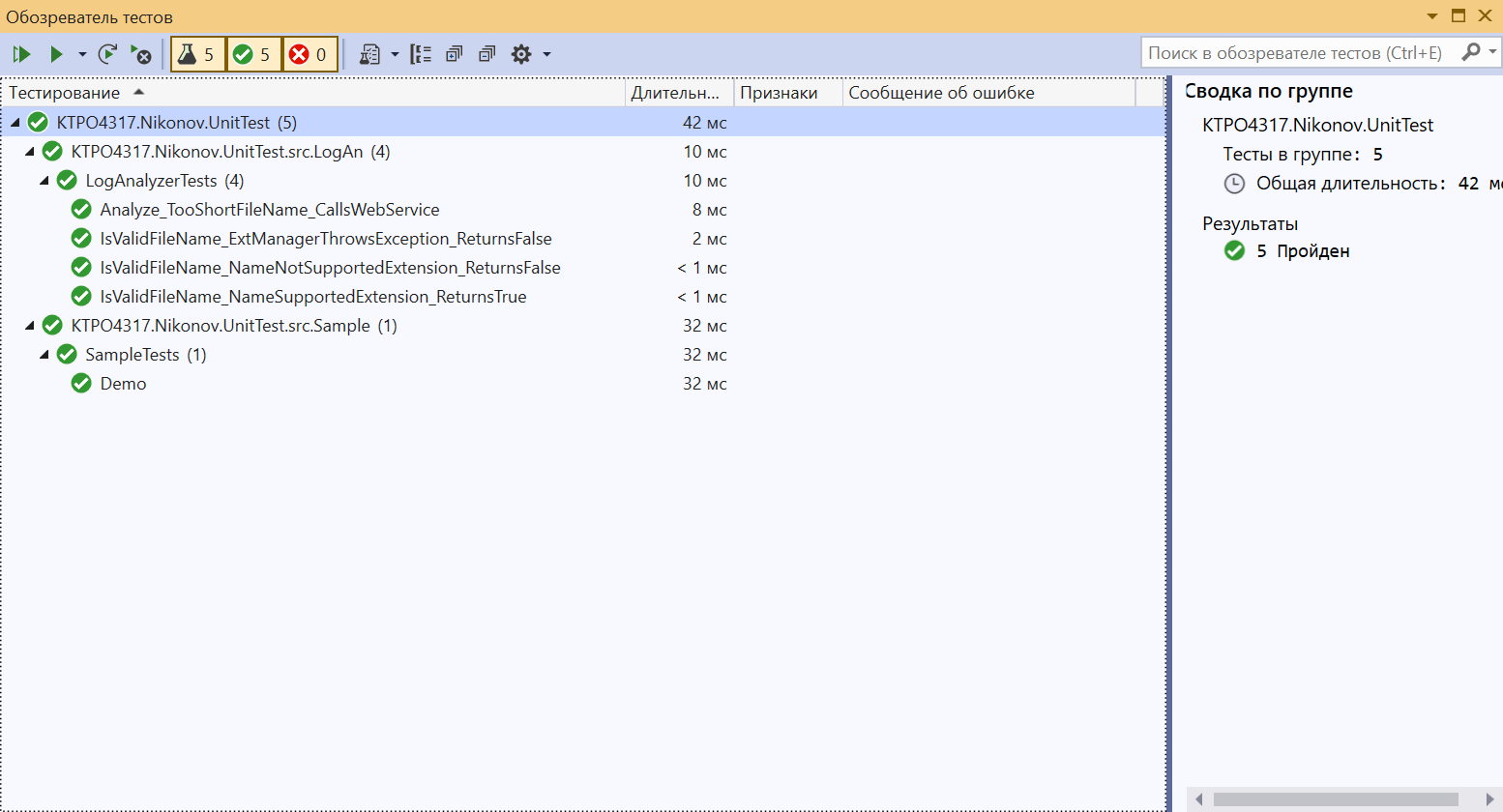


Создайте тестовый метод для сценария «Если имя слишком короткое, вызываем веб службу».

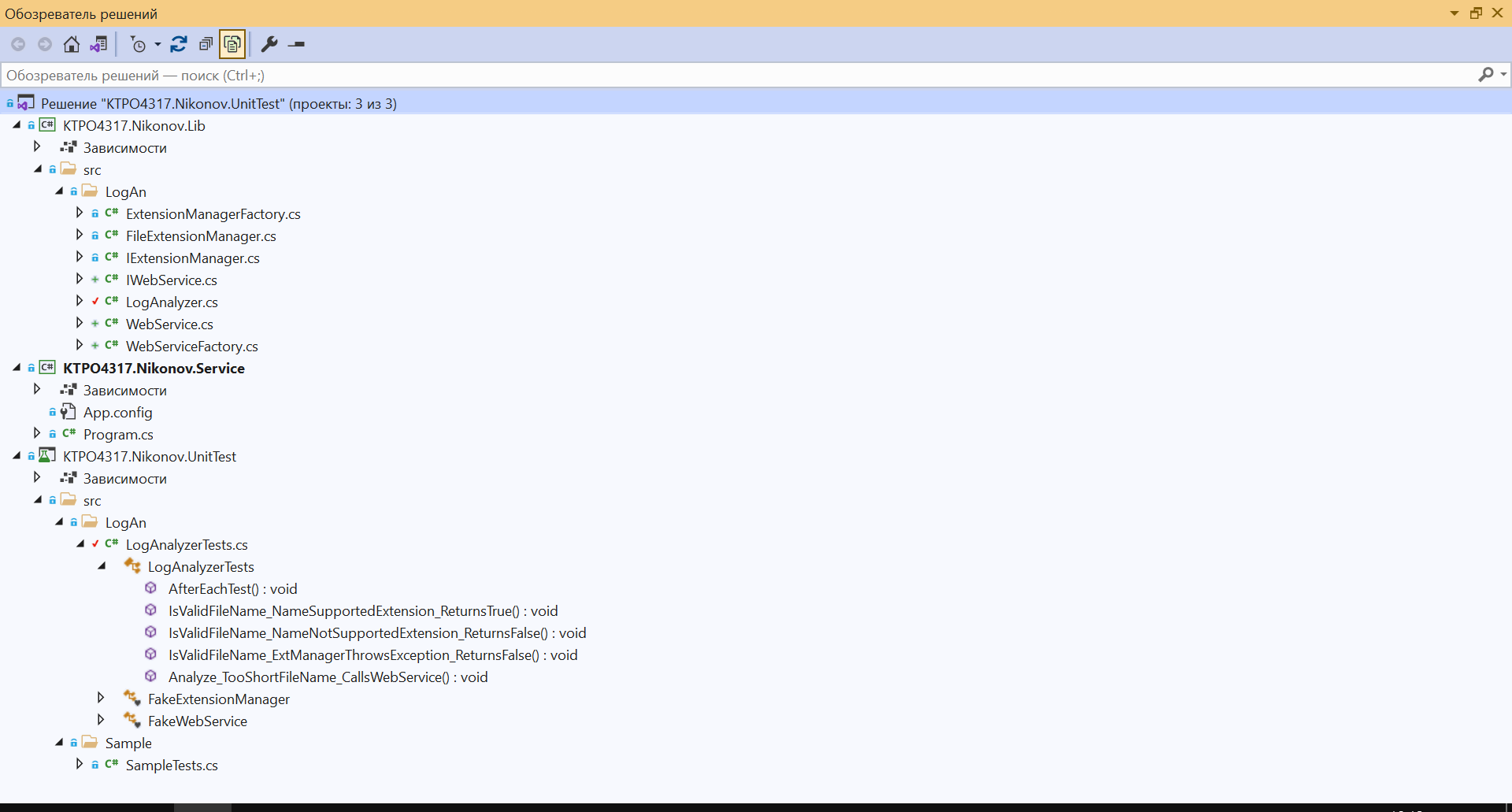
Analyze\_TooShortFileName\_CallsWebService()



Выполните тесты.

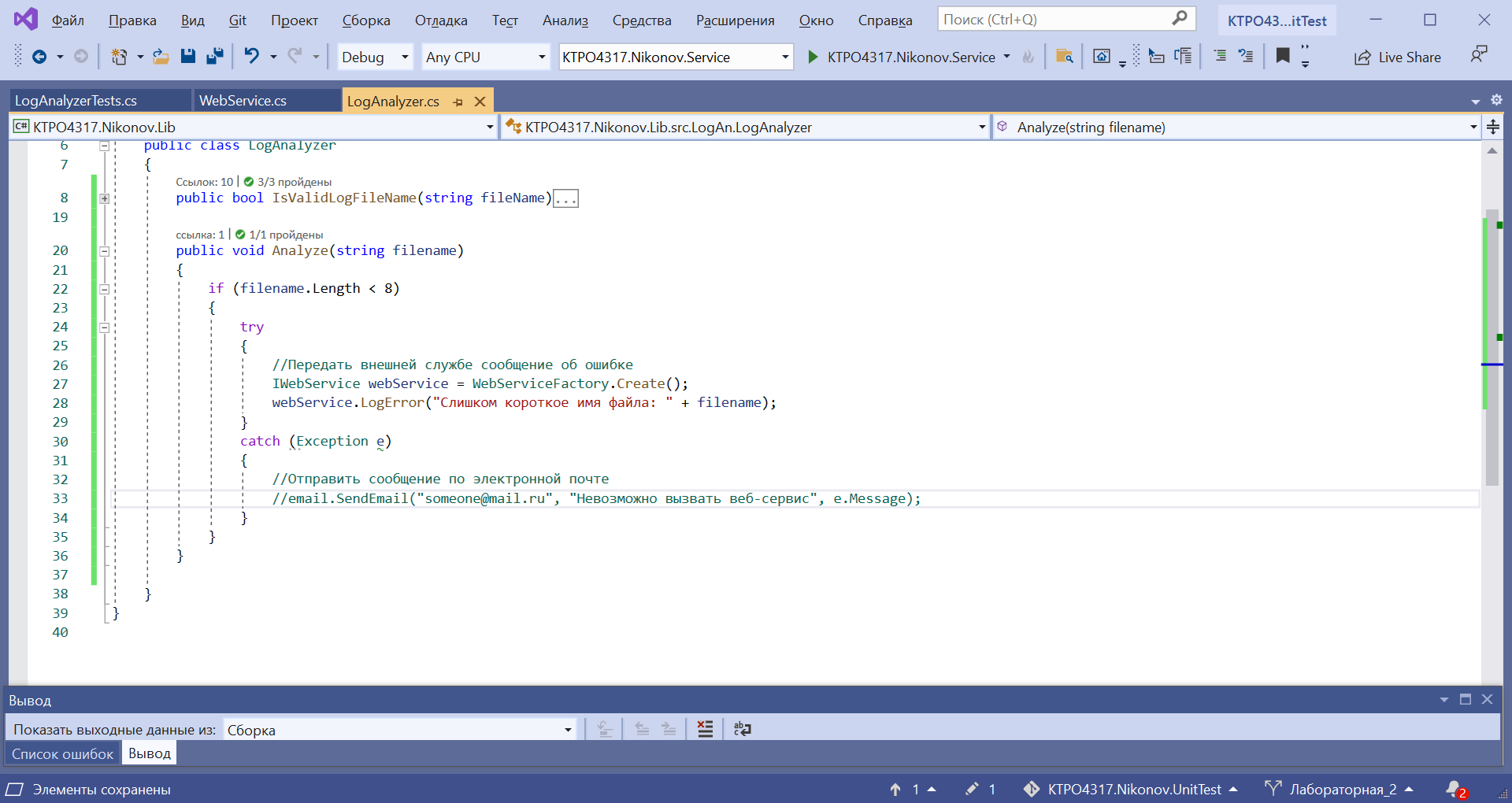


Структура проекта:

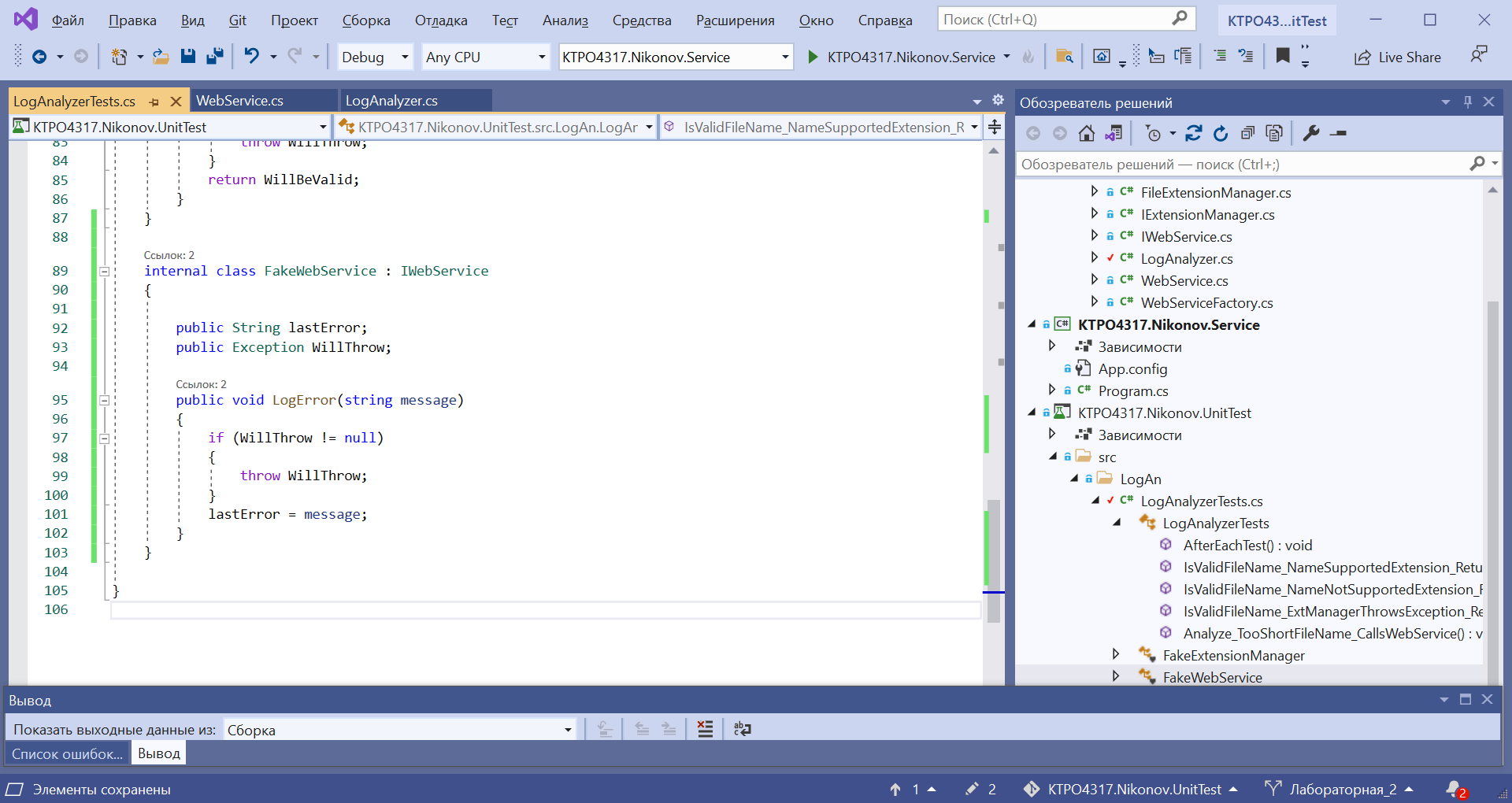


**3. Несколько поддельных объектов. Совместное использование поставки и заглушки**

Пусть теперь класс LogAnalyzer должен обратиться к веб-службе, и если служба вернет ошибку, то записать эту ошибку в другую зависимость, отправив сообщение по электронной почте. Логика выглядит следующим образом



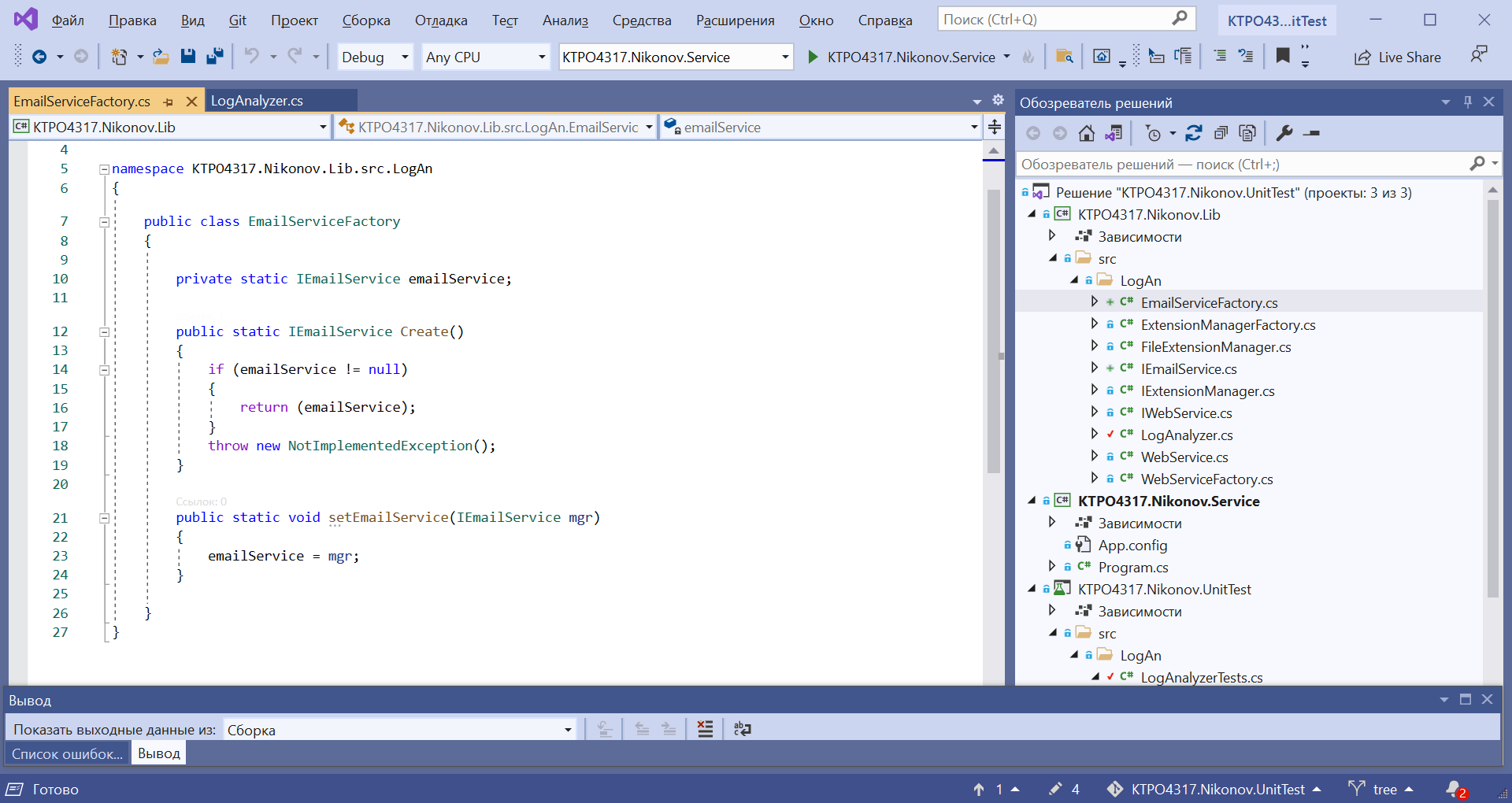
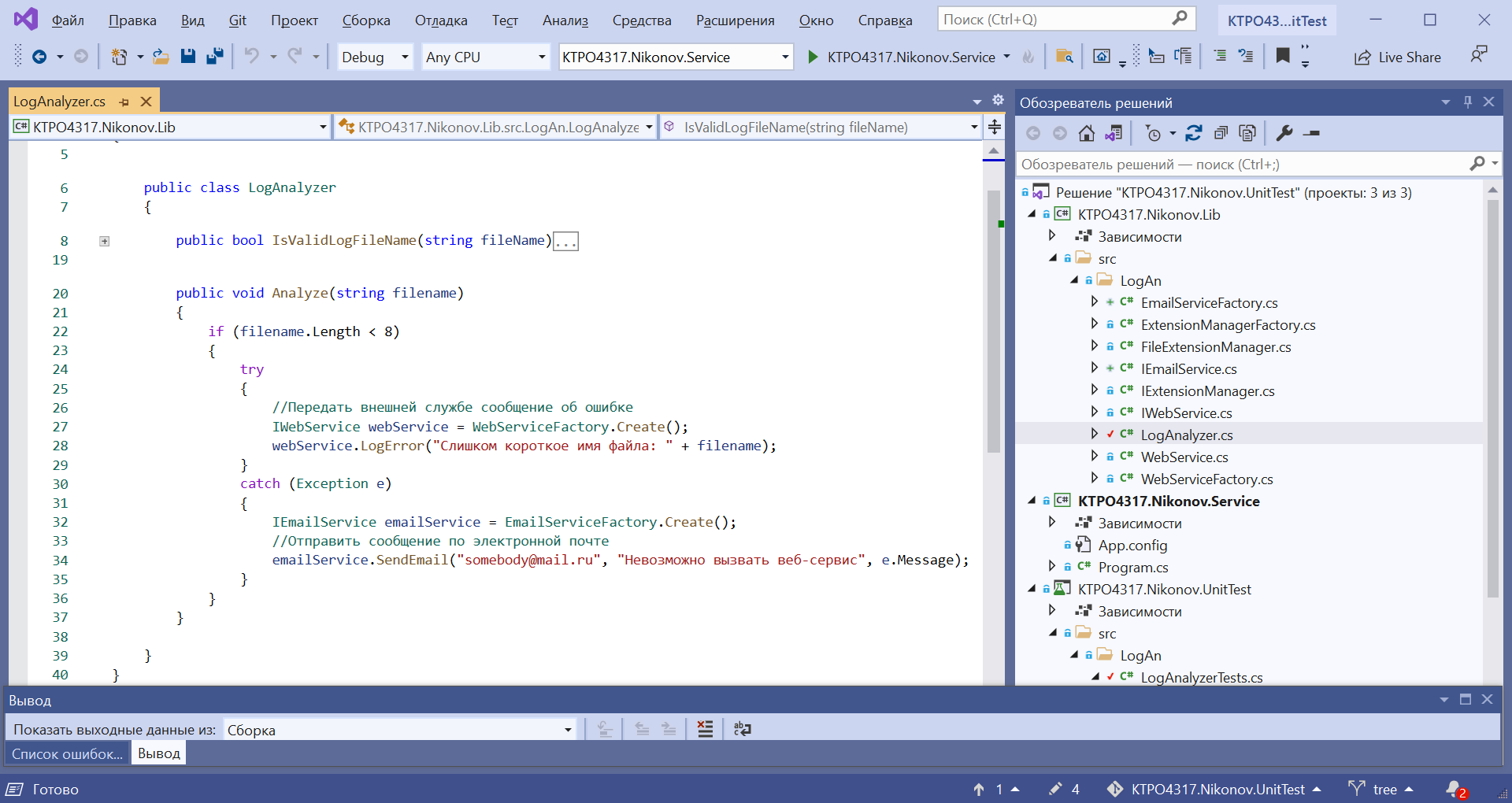
Для заглушки веб-службы воспользуемся интерфейсом и поддельным объектом, созданным в предыдущем упражнении. Доработайте класс FakeWebService, так, чтобы можно было управлять результатом вызова, а и именно имитировать вызов исключения, так как это было сделано в лабораторной работе №2.



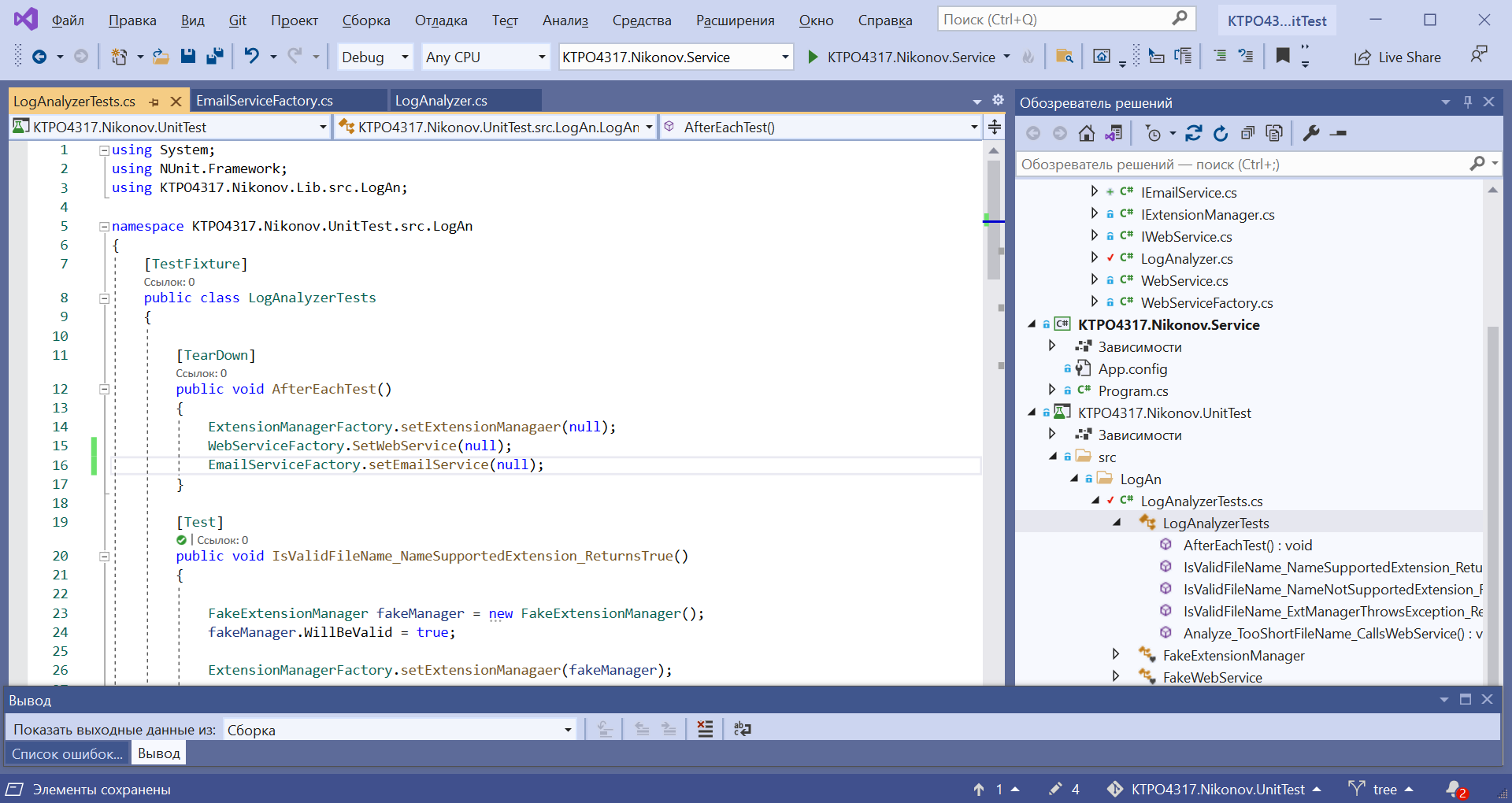
Выделите интерфейс IEmailService с одним методом void SendEmail(string to, string subject, string body), где to – адрес, subject – тема, body – сообщение. Это интерфейс будем использовать как для создания подставки



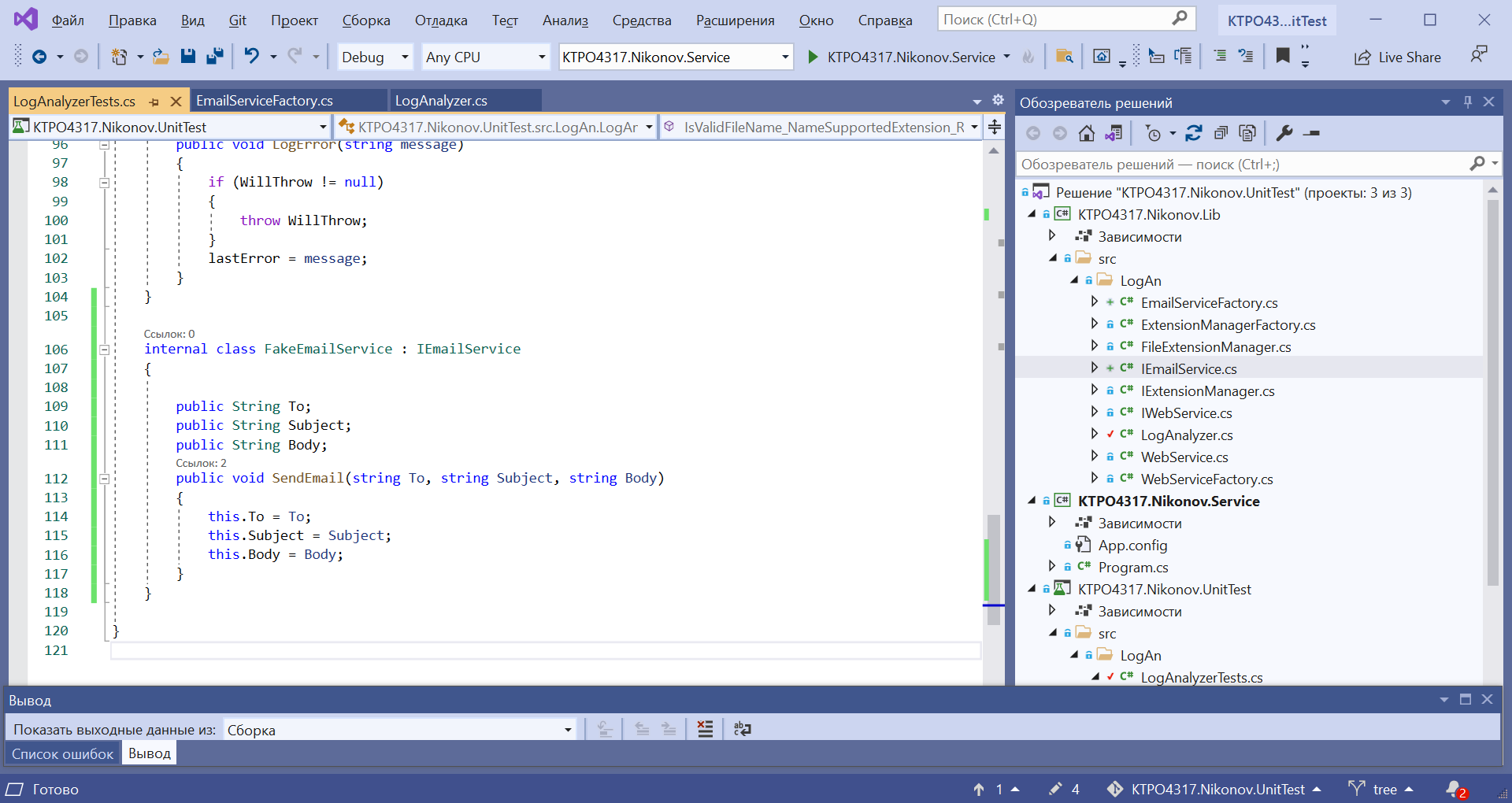
Замените псевдокод на обращение к веб-службе через интерфейс IEmailService. Внедрите зависимость используя рассмотренный в лабораторной работе №2 метод внедрения зависимости через фабрику.



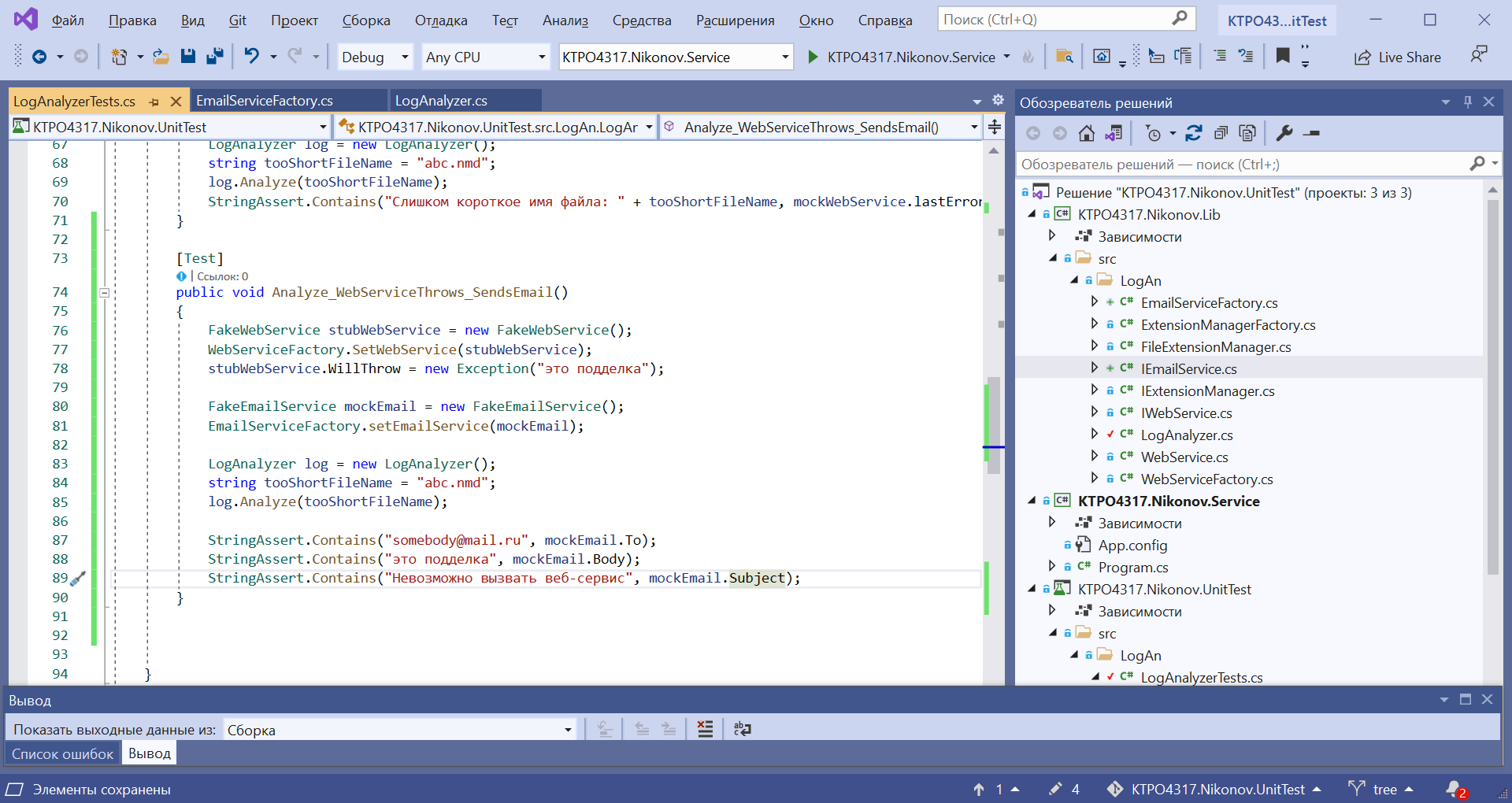
Воспользуйтесь методом [TearDown] для восстановления исходного состояния фабрики.



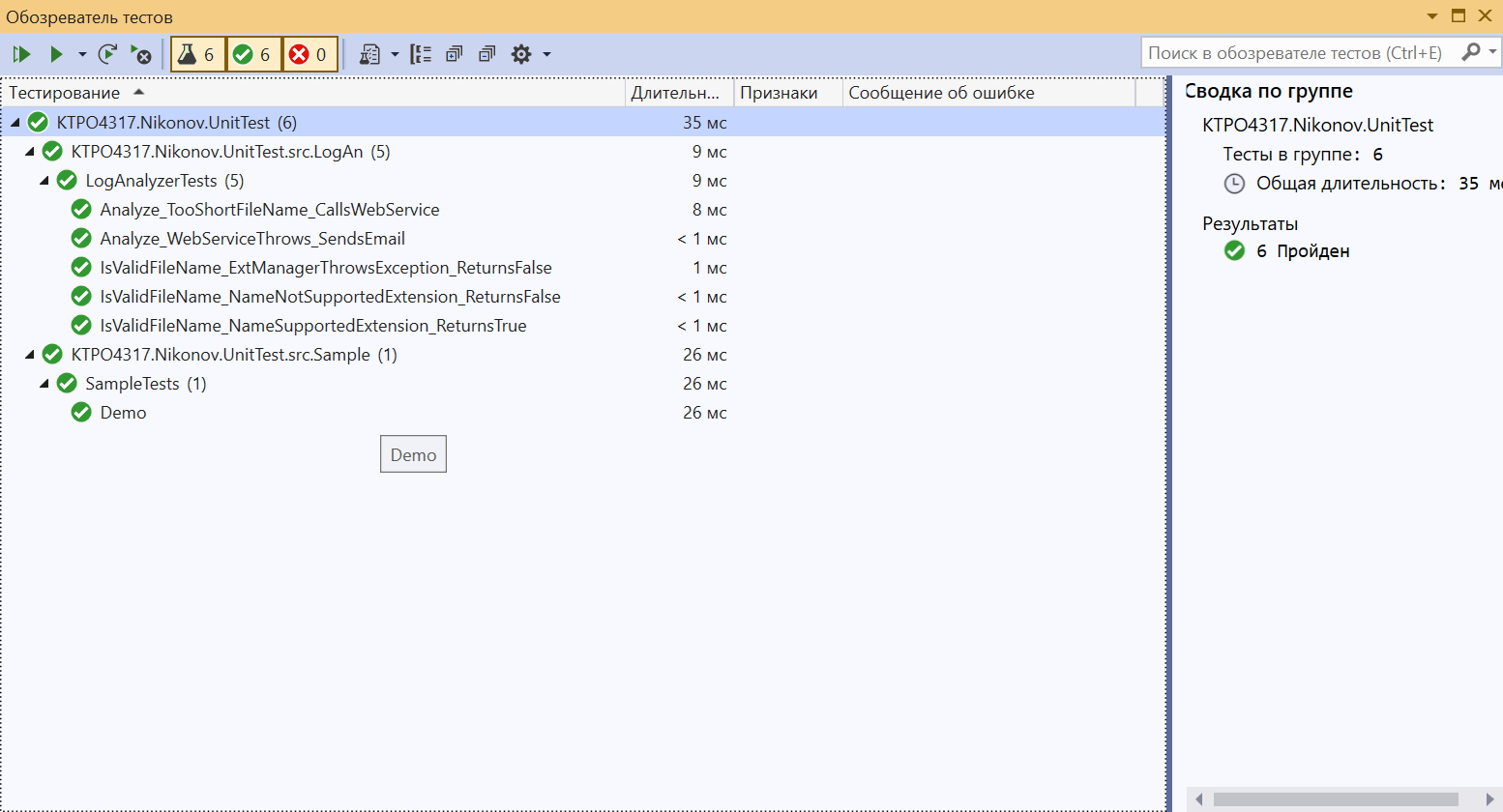
Создайте поддельный объект FakeEmailService: IEmailService. Добавьте код, который позволить делать утверждения относительного этого объекта, после вызова метода SendMail. Необходимо сохранить состояние всех параметров метода, так как утверждения тоже должно высказываться относительно правильности всех трех значений.



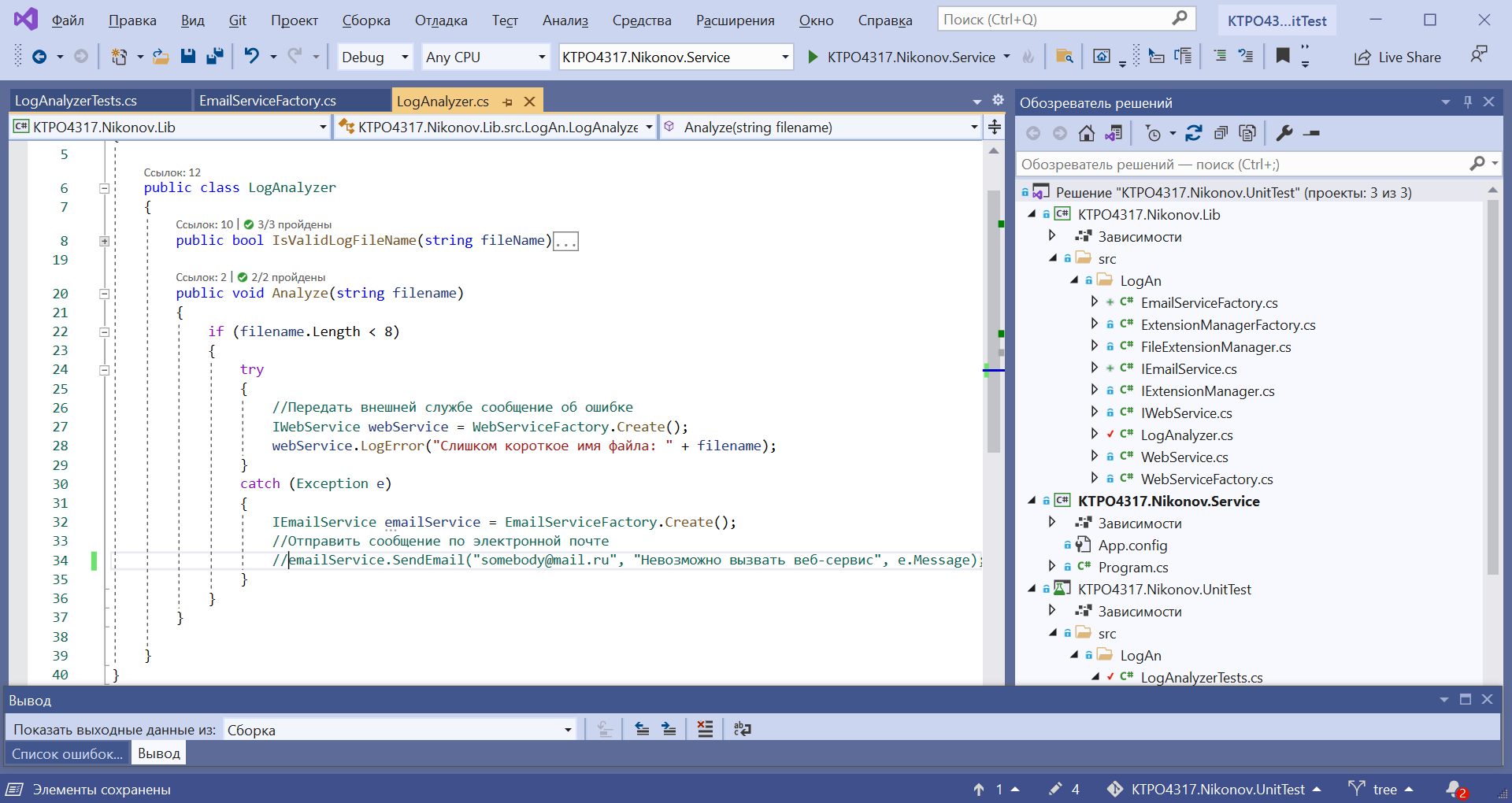
Создайте тестовый метод для сценария «Если веб-служба вызывает исключение, отправляем почтовое сообщение». Analyze\_WebServiceThrows\_SendsEmail()

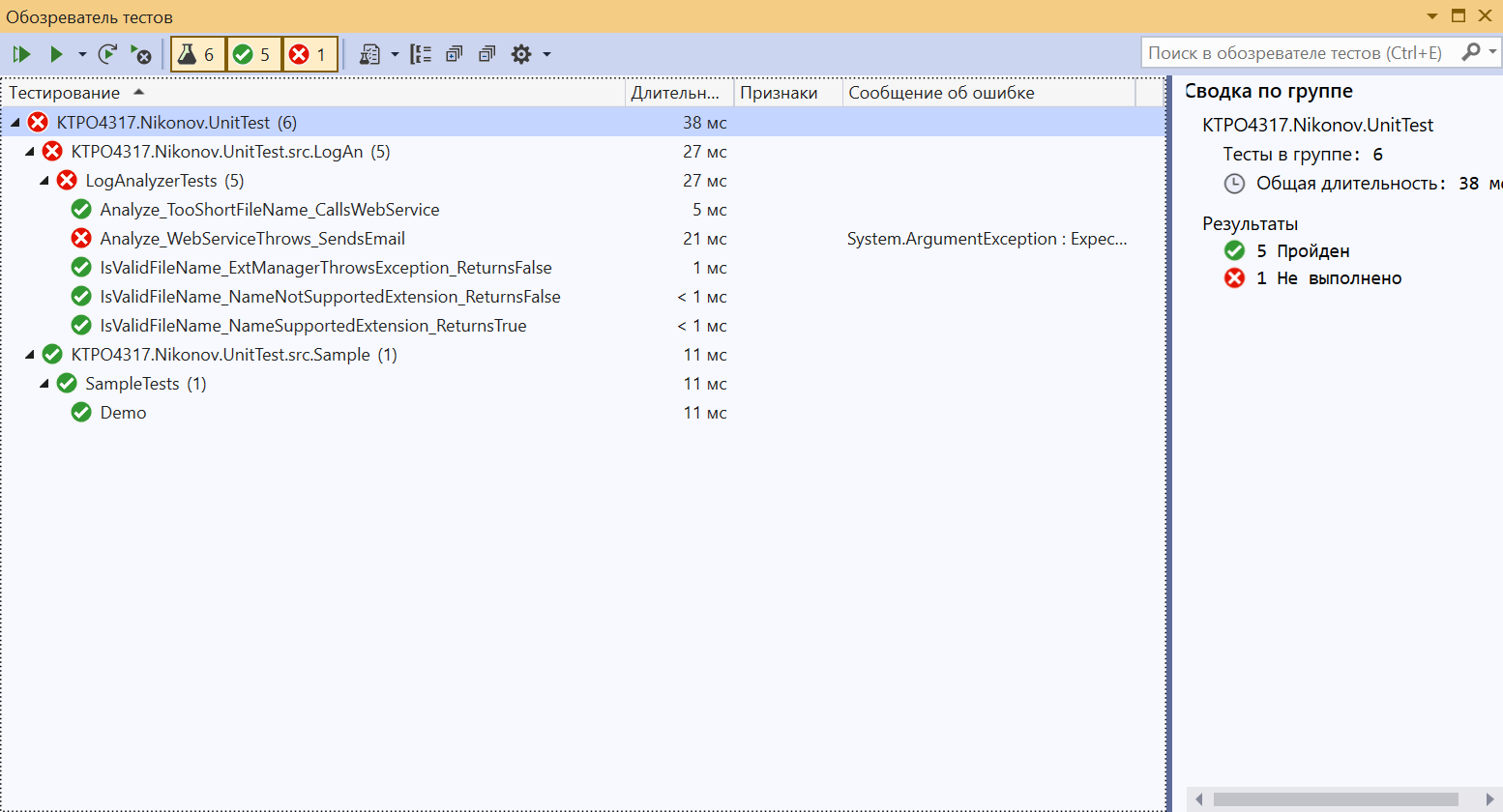


Выполните тесты.



Проверьте правильность тестового метода. Для этого внесите в тестируемый метод дефект, ошибку которую должен обнаружить тест. Выполните тест и зафиксируйте результат. Восстановите правильный код.





Структура решения:

