Лабораторная работа №2.

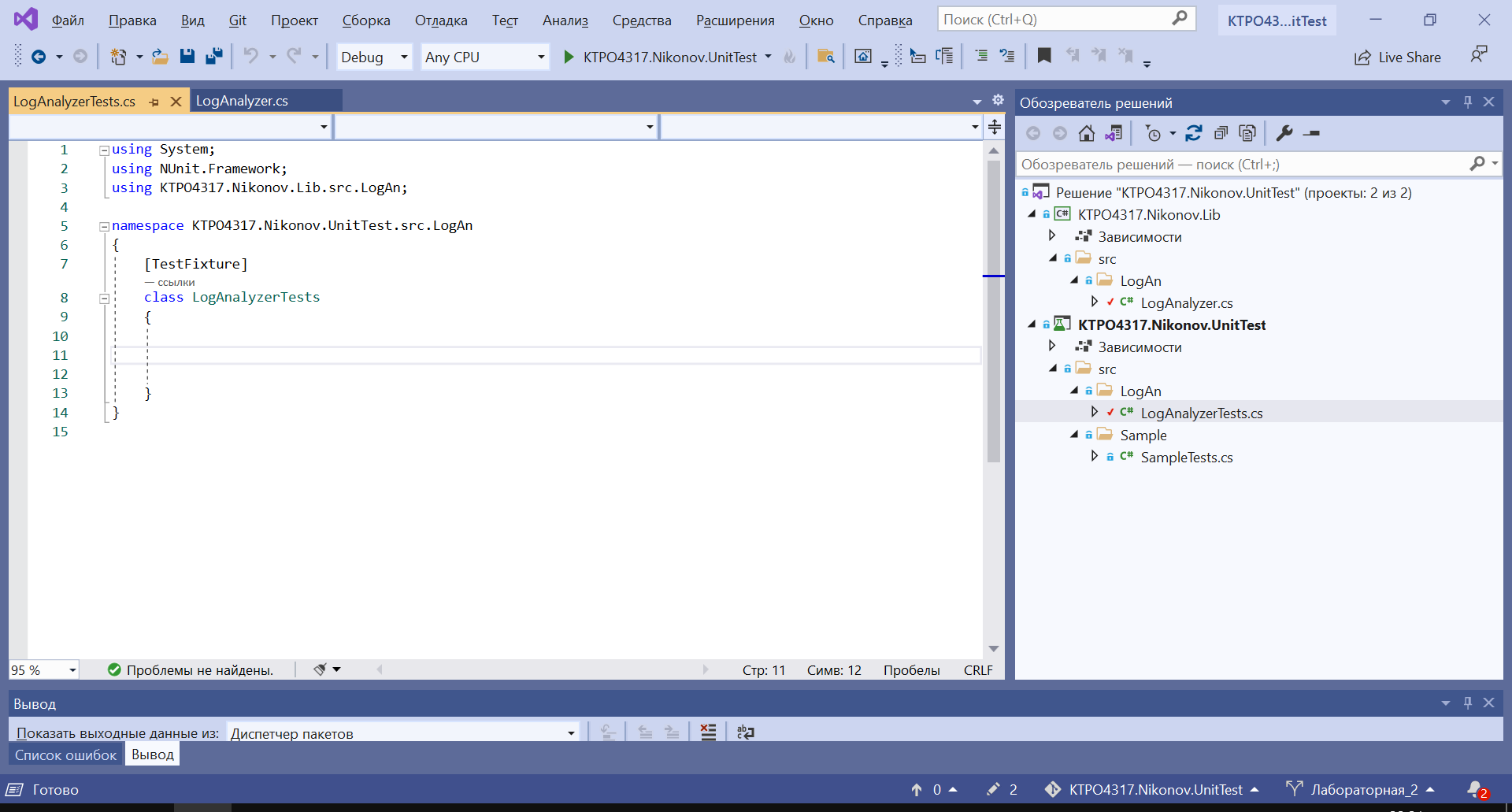
Использование заглушек для разрыва зависимостей

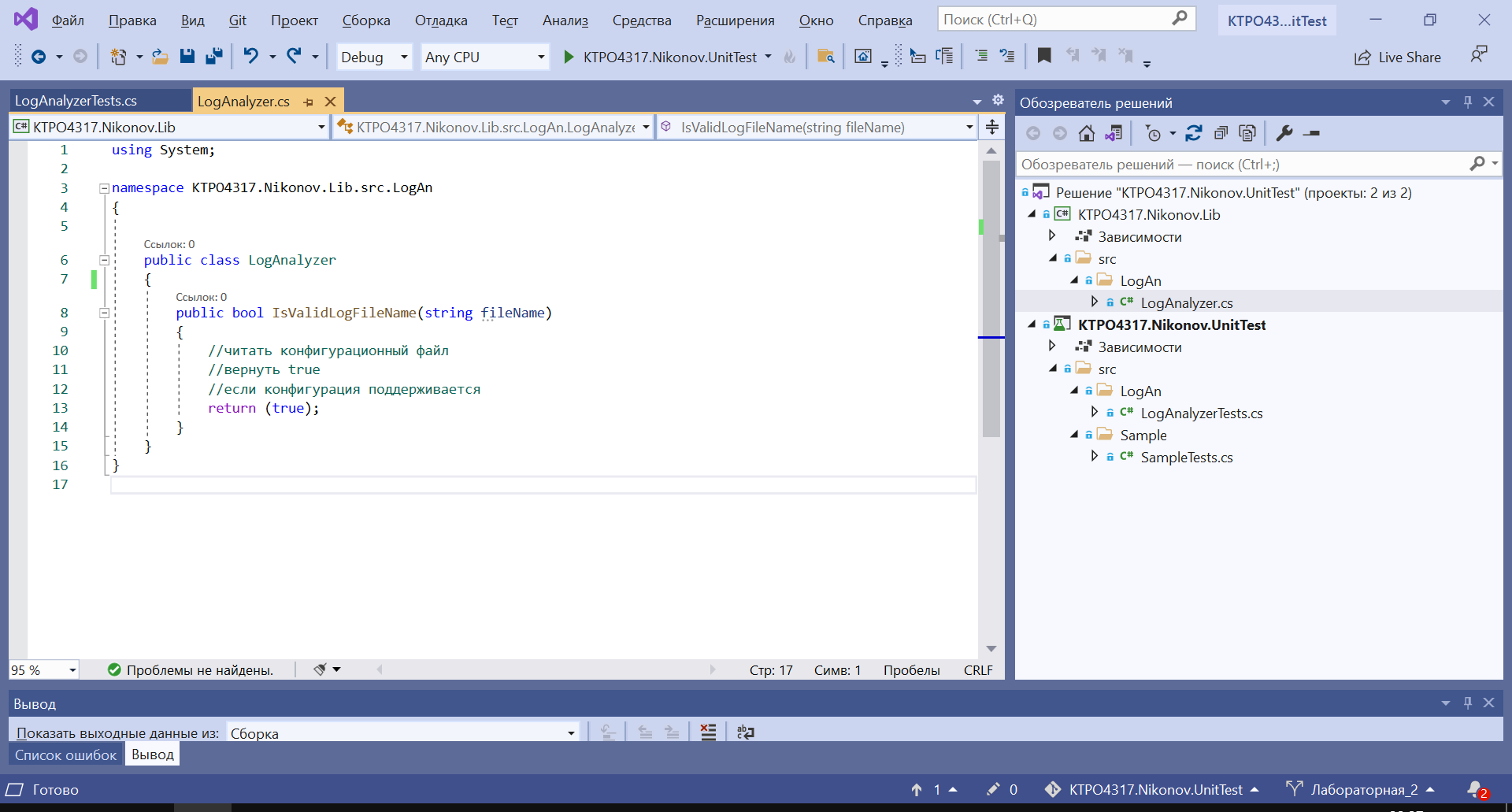
**Цель работы:**

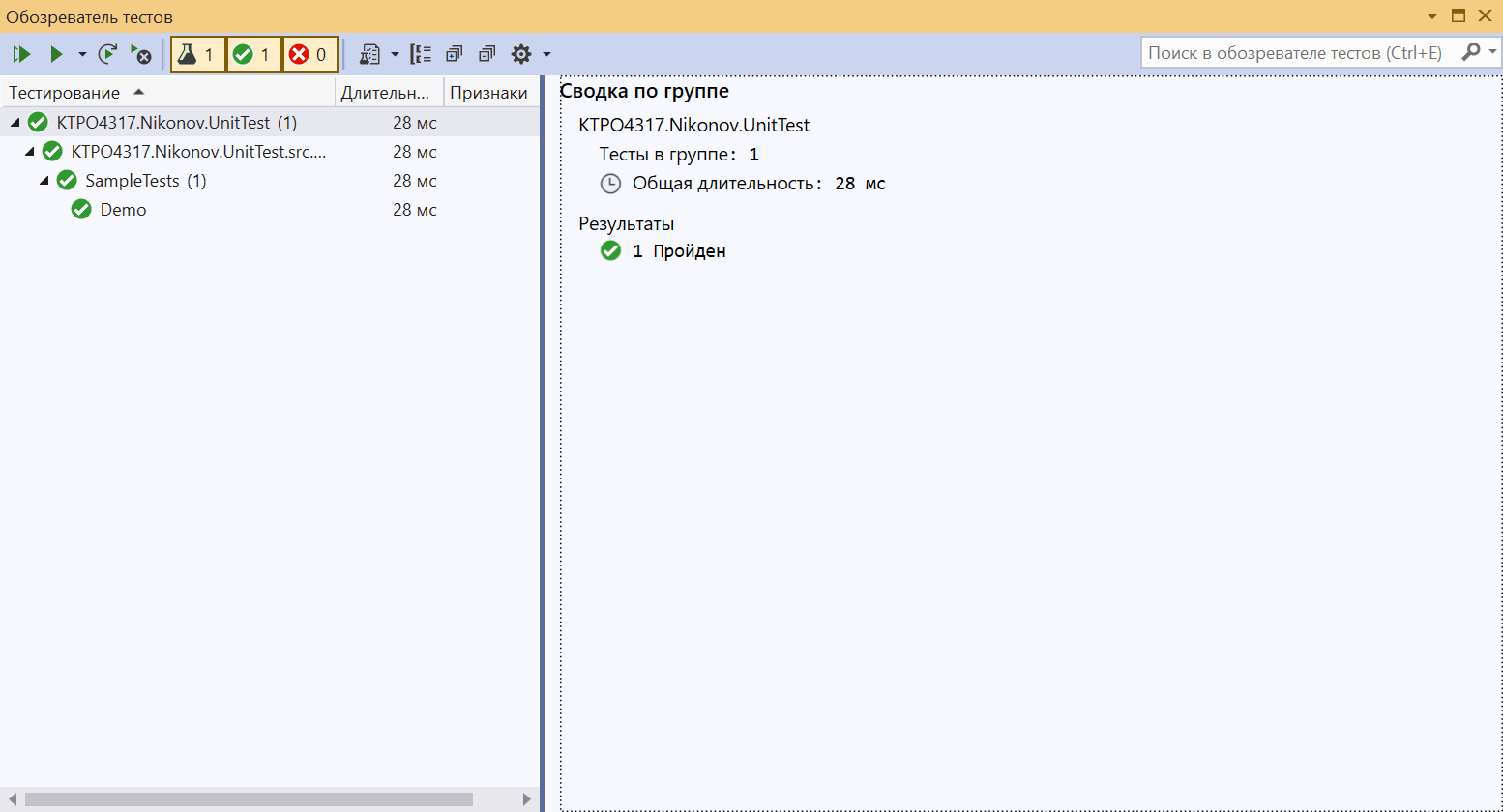
Приобретение практических навыков использования заглушен для разрыва зависимостей при автономном тестировании модулей, практика использования тестового каркаса NUnit.

**Ход работы:**

**1.Подготовка проекта**

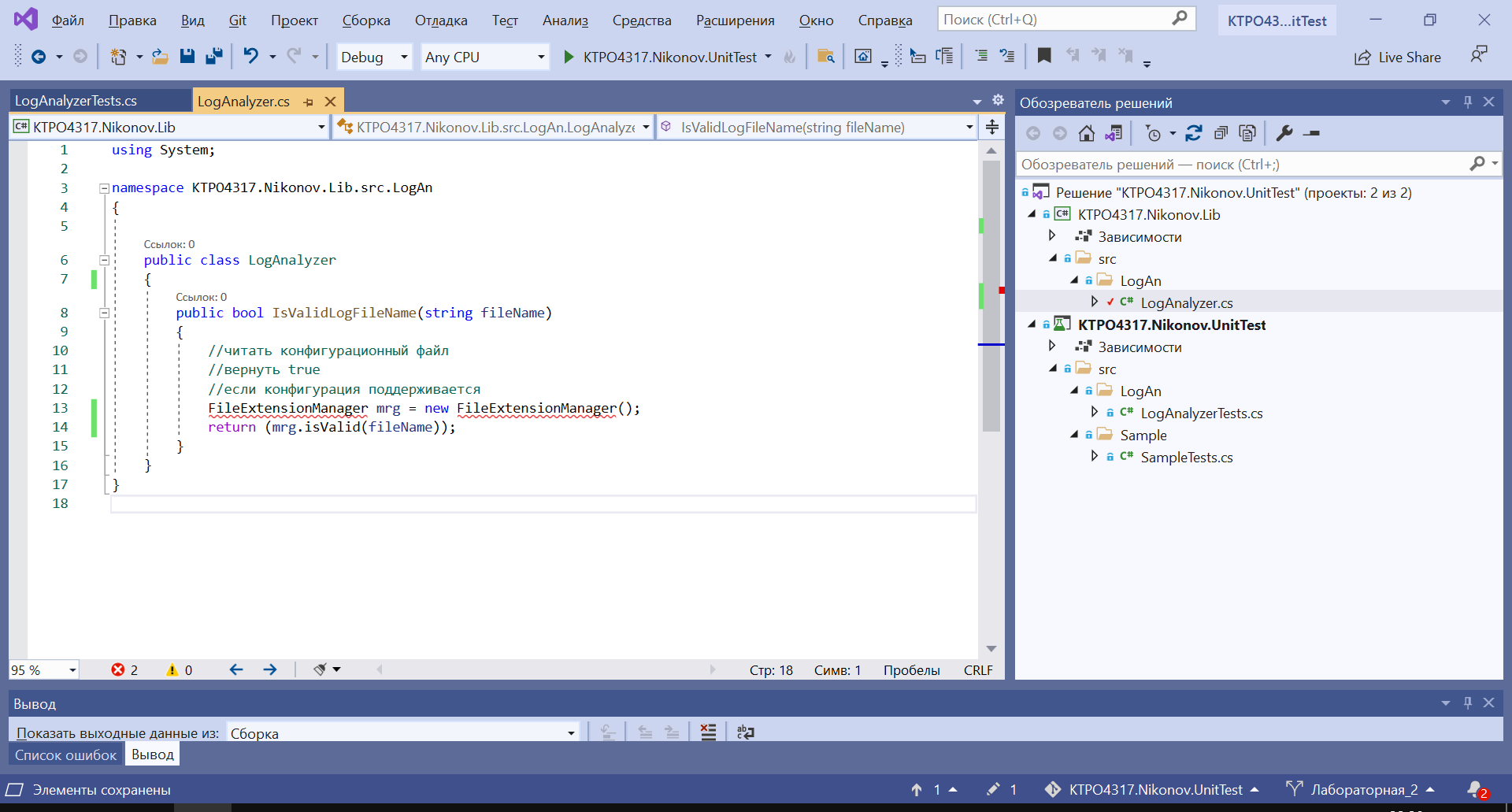






**2. Рефакторинг проекта для повышения тестопригодности: выделение в отдельный класс операций с файловой системой**

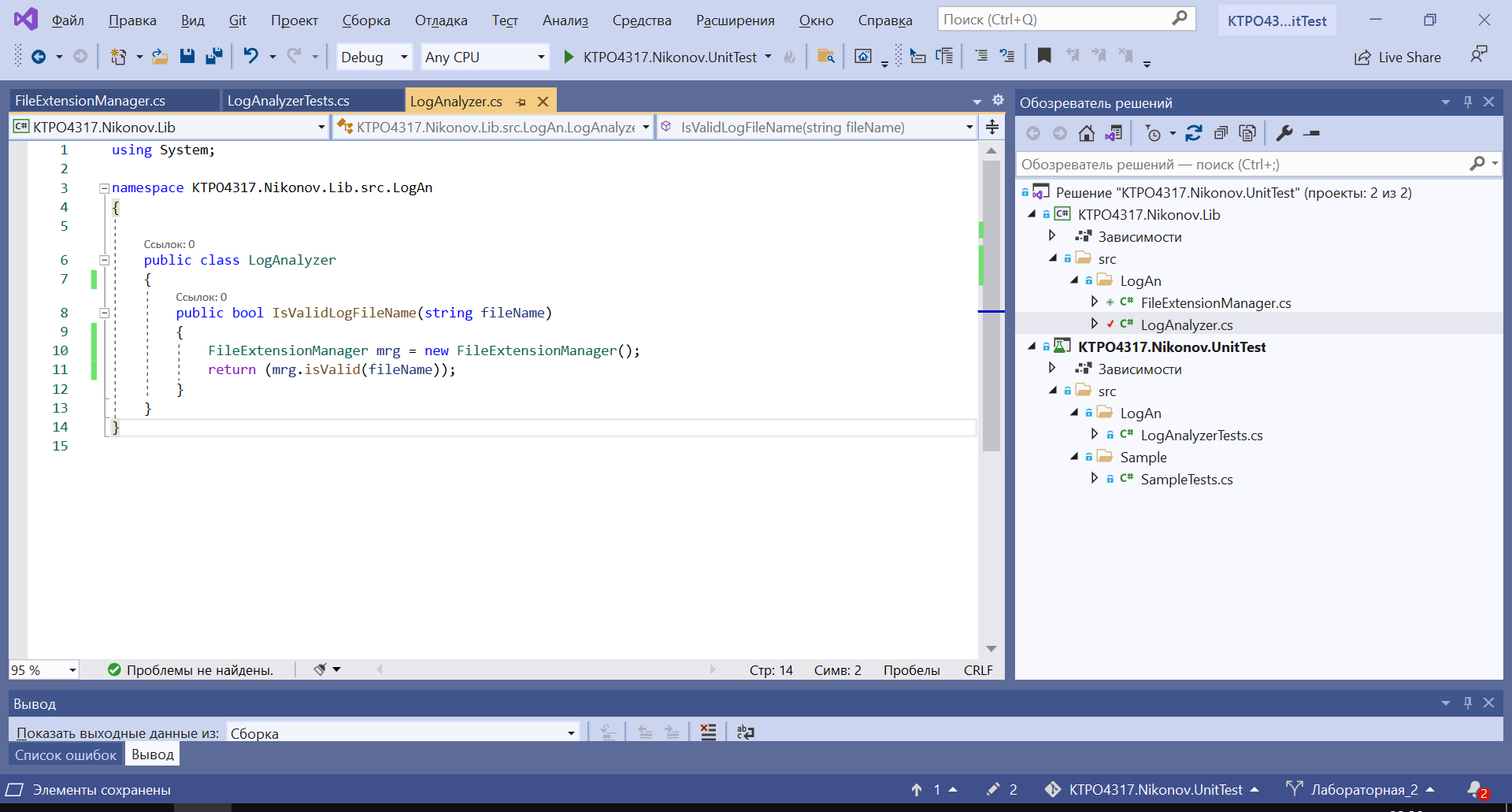
Наберите в методе LogAnalyzer.IsValidLogFileName после псевдокода создание объекта код создания FileExtensionManager и вызов метода IsValid. Поскольку данного класса и метода еще не существует при компиляции будет ошибка

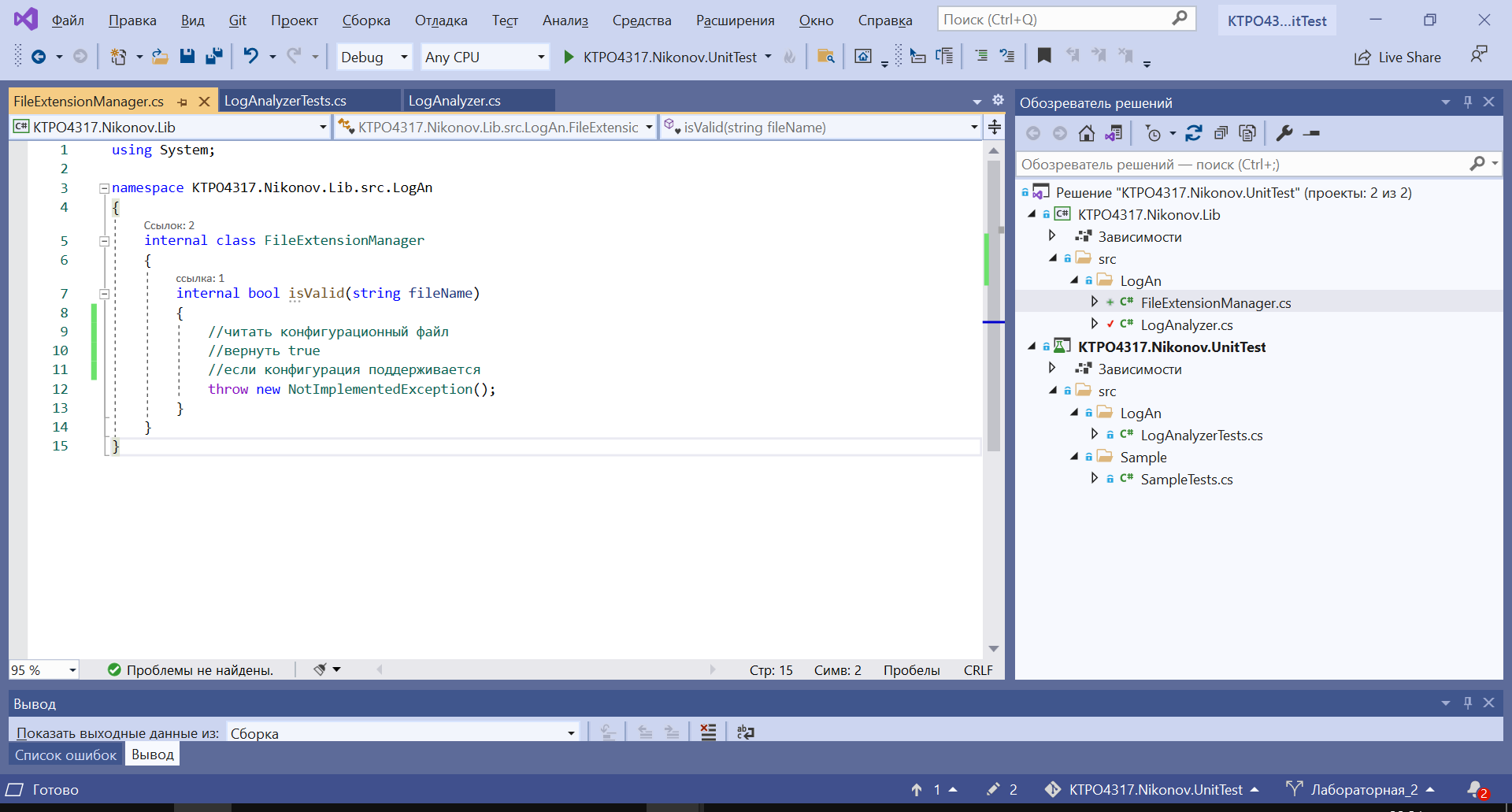


Установите курсор на FileExtensionManager и выберете: Быстрые действия и рефакторинг -> Создать class в новом файле.

Установите курсор на mgr.IsValid и выберите: Быстрые действия и рефакторинг -> Создать метод.

Перенесите в созданный метод псевдокод. Вызов исключения NotImplementedException оставьте, чтобы проект компилировался.

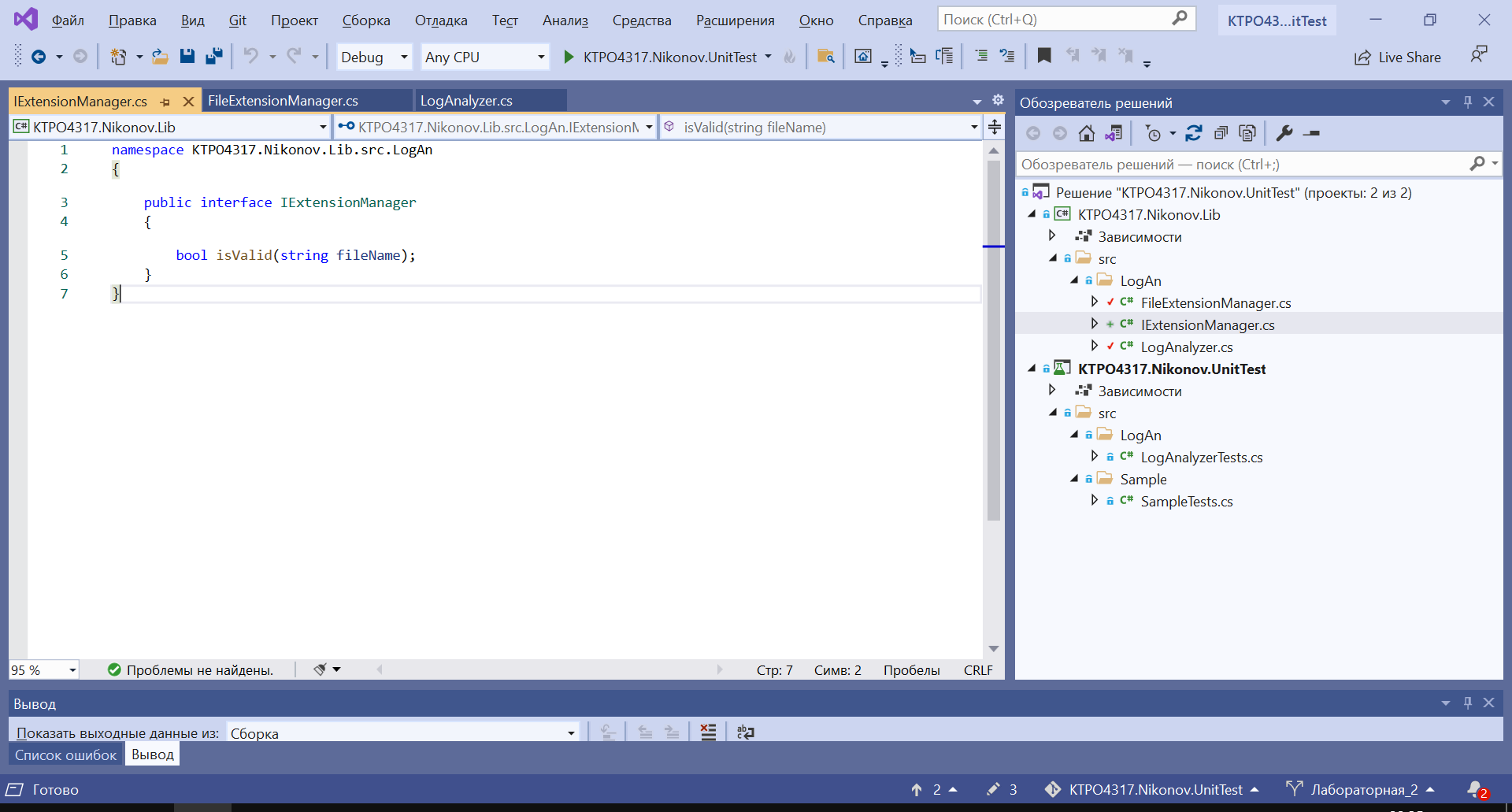


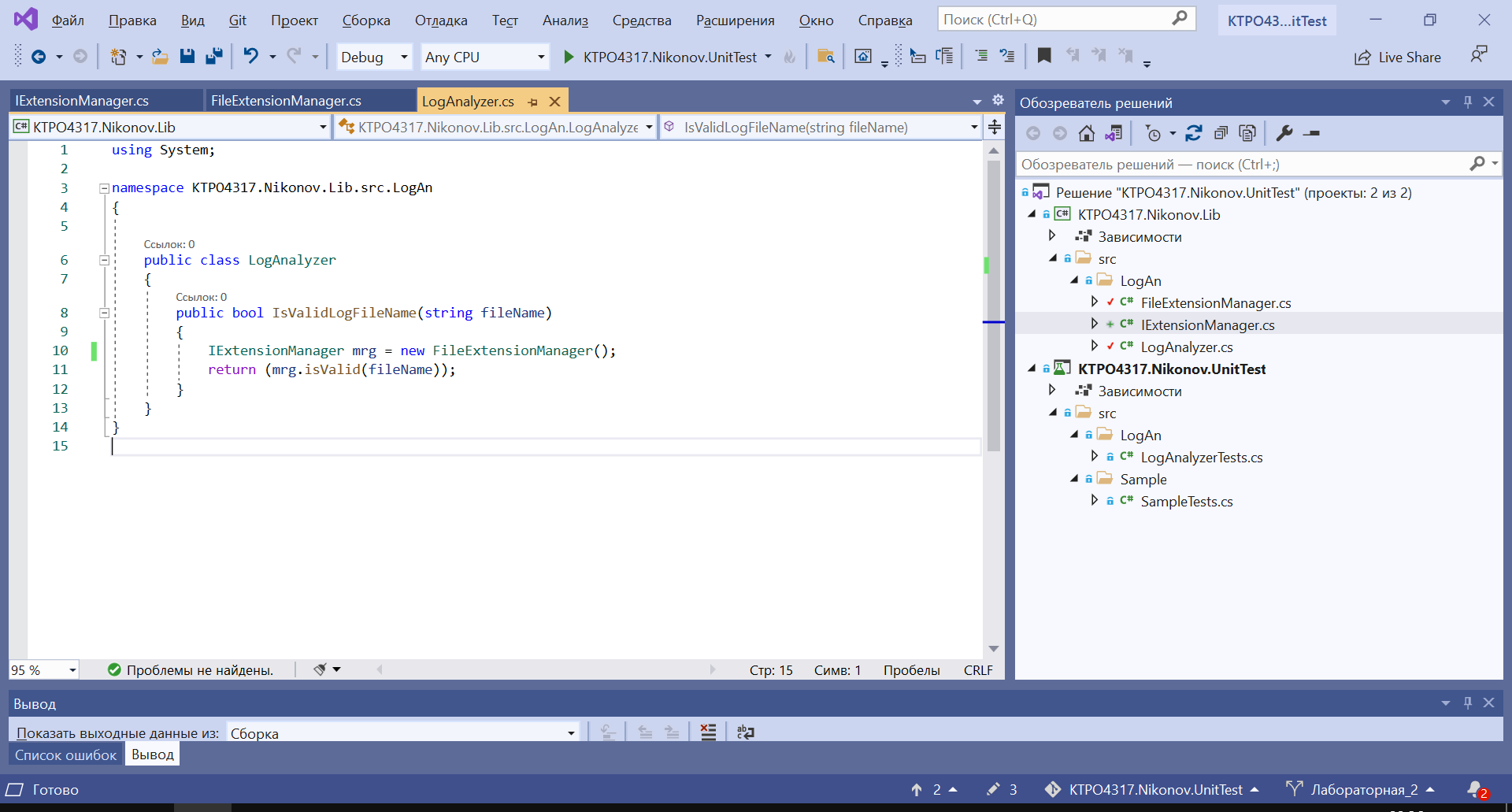
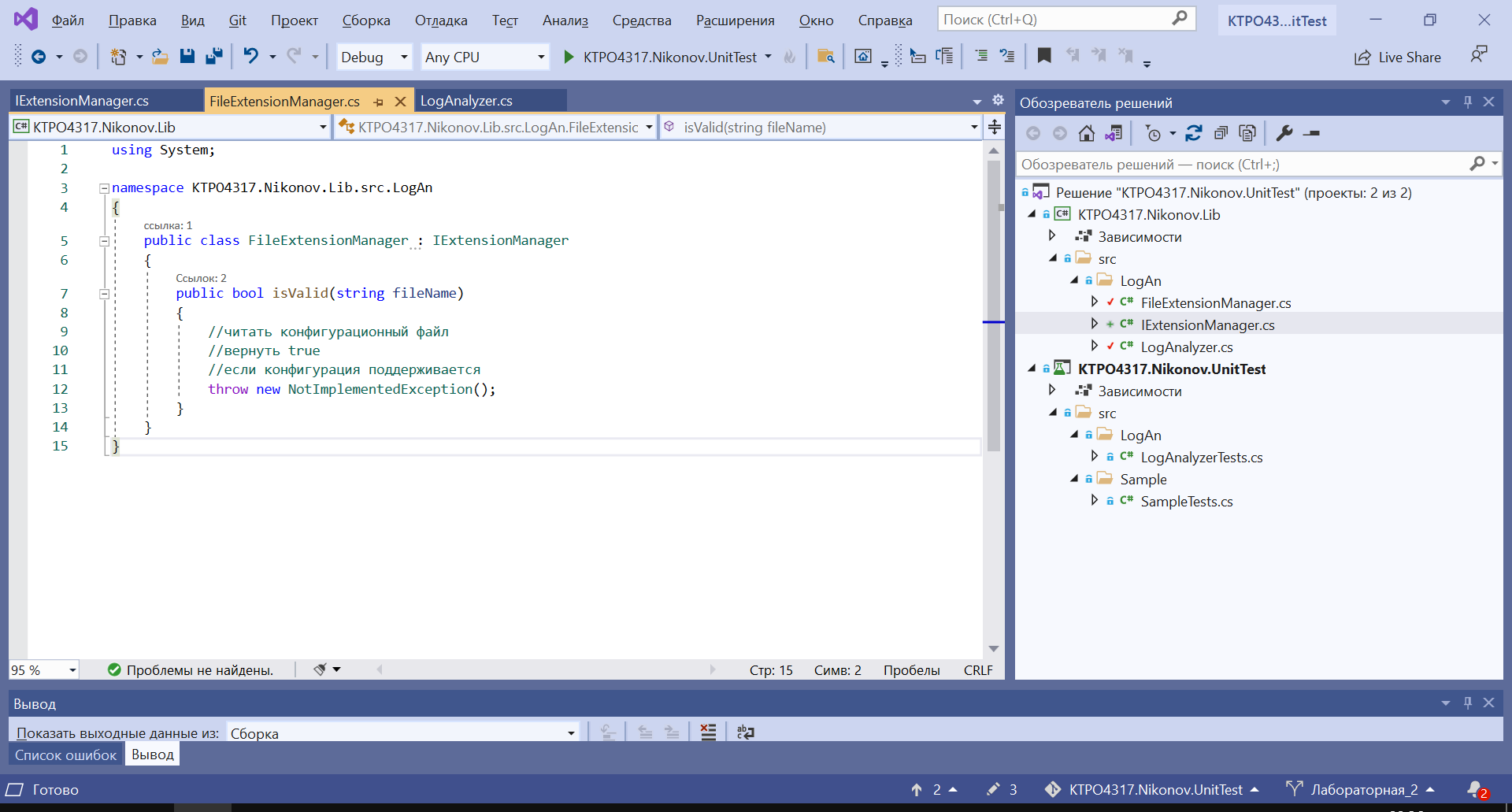


**3. Рефакторинг проекта для повышения тестопригодности: выделение интерфейса**

Замените у класса FileExtensionManager и его метода на IsValid модификатор доступа internal на public.

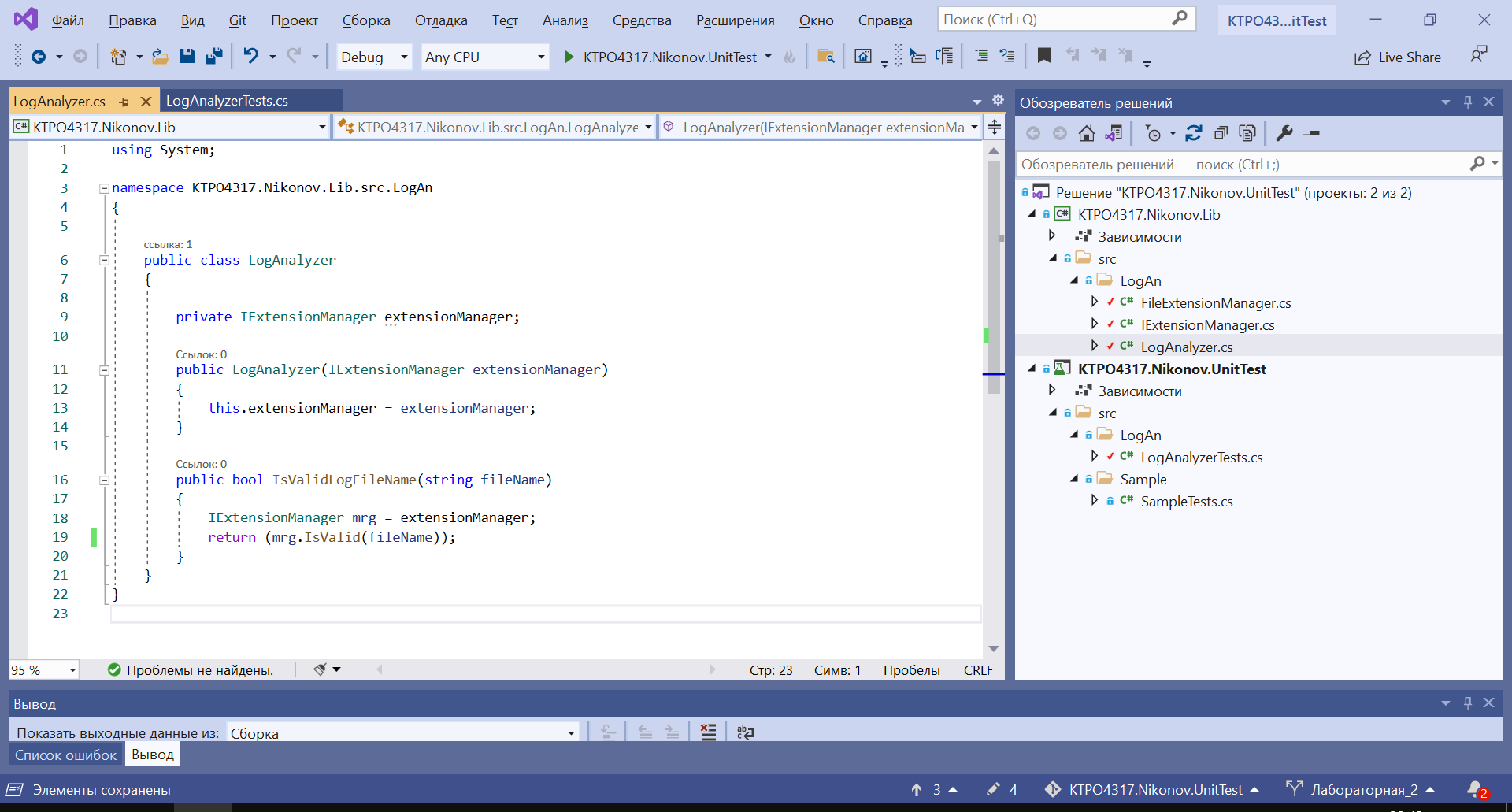
Выделите интерфейс FileExtensionManager, создать его в отдельном файле: Быстрые действия и рефакторинг->Извлечение интерфейса

Объявите переменную диспетчера расширений в тестируемом методе IsValidLogFileName как интерфейс. 

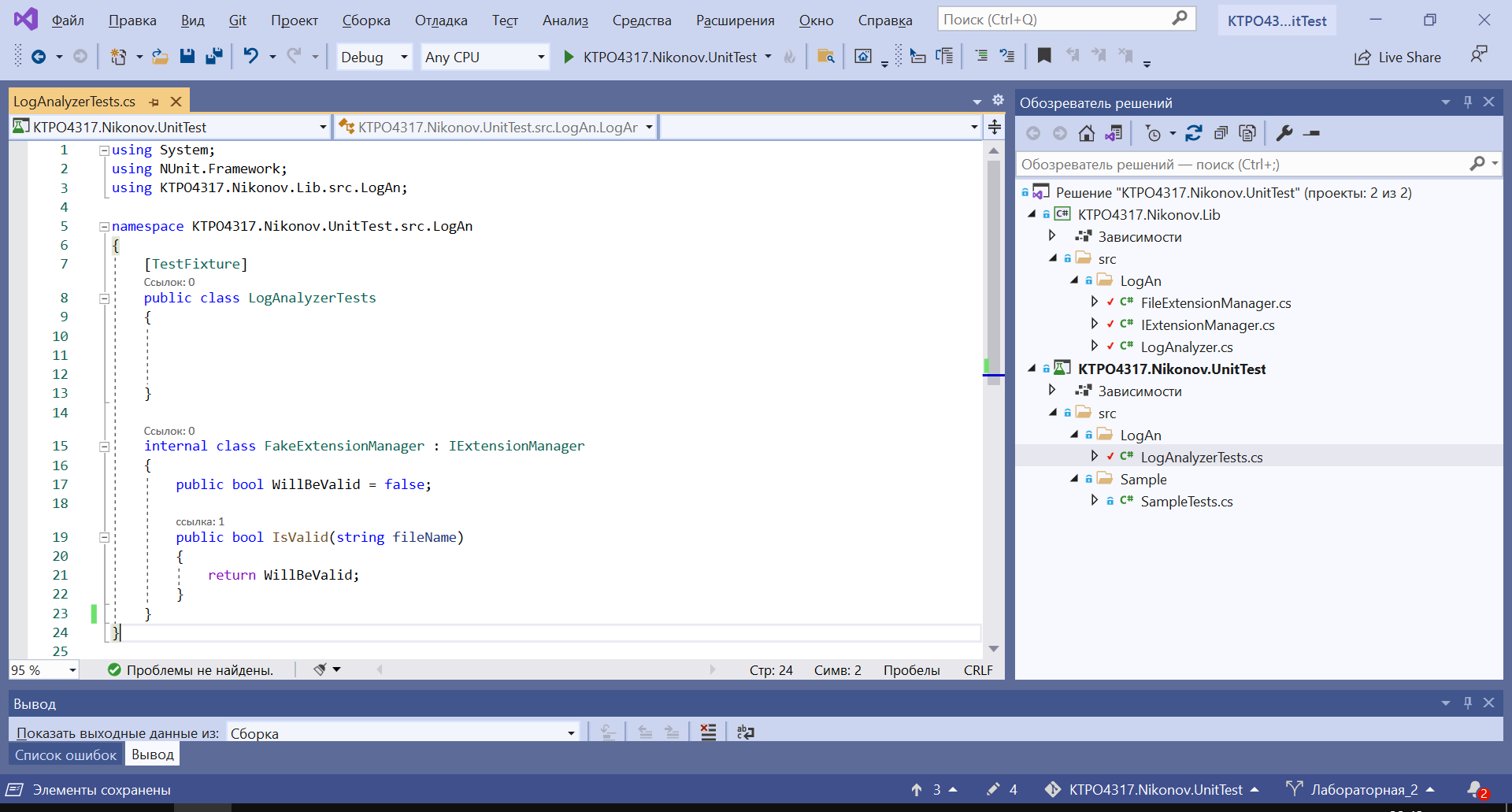


**4. Внедрение зависимости. Внедрение подделки через конструктор**

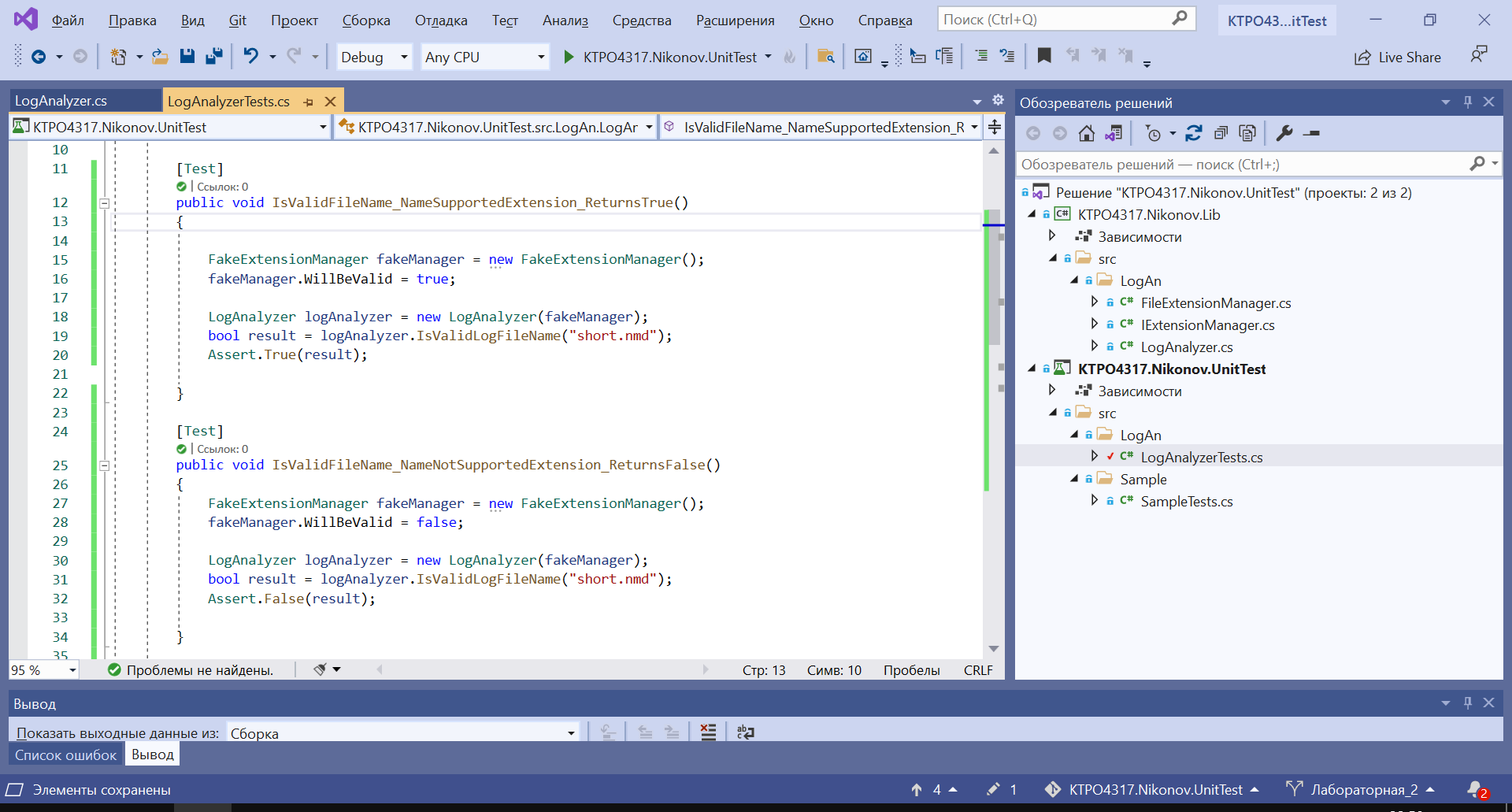
Объявите в классе LogAnalyzer поле типа IExtensionManager, конструктор с параметром тоже типа, инициализирующий это поле. В методе IsValidLogFileName замените создание нового класса на обращение к полю.



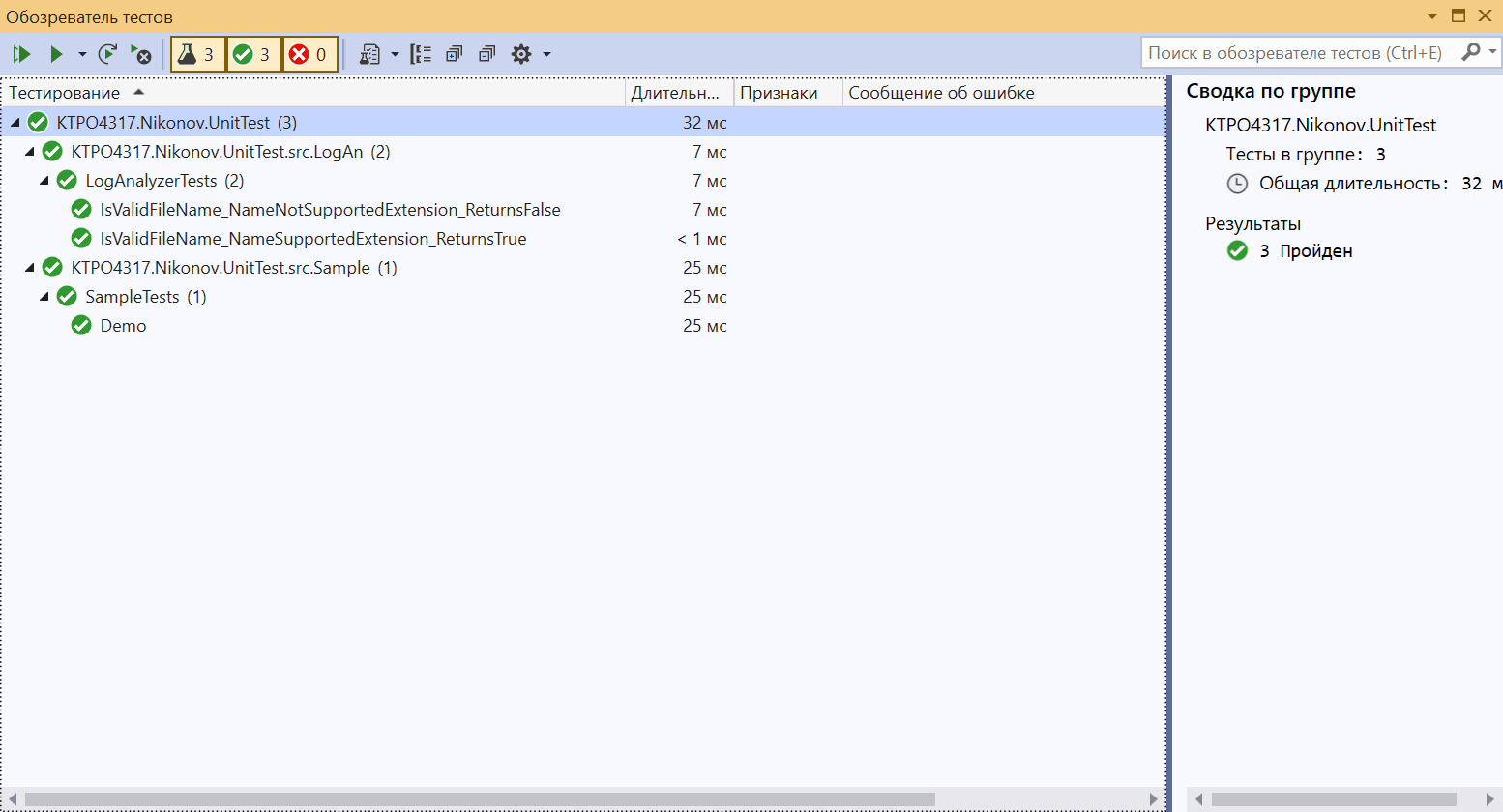
Создайте заглушку FakeExtensionManager, реализующую интерфейс IExtensionManager. Разместите исходный код поддельного класса в том же файле, что и тестовый класс.



Создайте тестовый метод IsValidFileName\_NameSupportedExtension\_ReturnsTrue для тестового сценария «Для поддерживаемых расширение метод возвращает true». Создайте и настройте заглушку для этого сценария. Используйте ее при создании экземпляра LogAnalyzer. Добавьте вызов метода IsValidLogFileName и проверку результата Добавьте самостоятельно тестовый метод для сценария «Для не поддерживаемых расширение метод возвращает false»



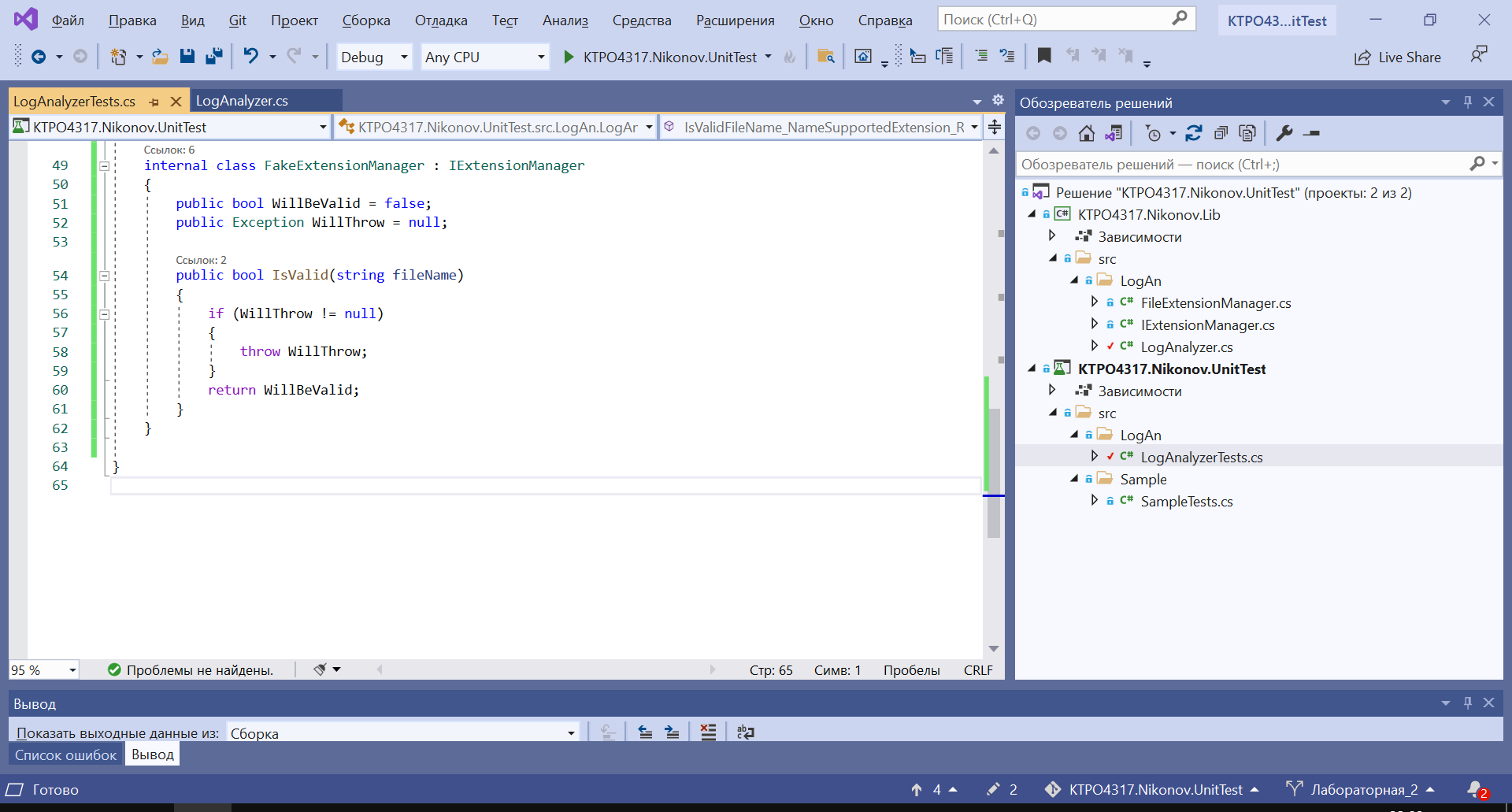
Прогоните тесты.



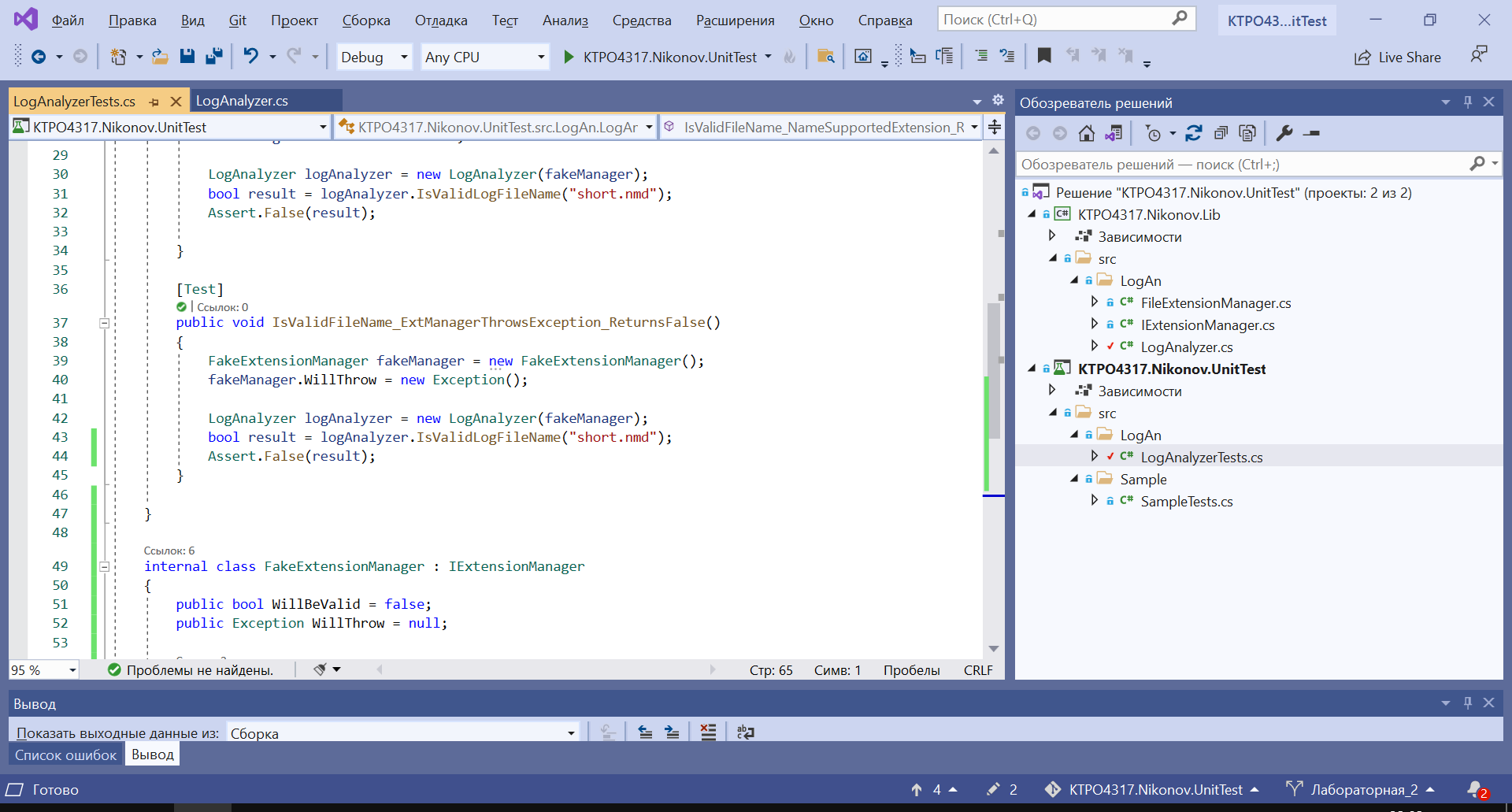
**5. Имитация возникновения исключения во внешнем классе.**

Добавьте в класс FakeExtensionManager свойство Exception WillThrow

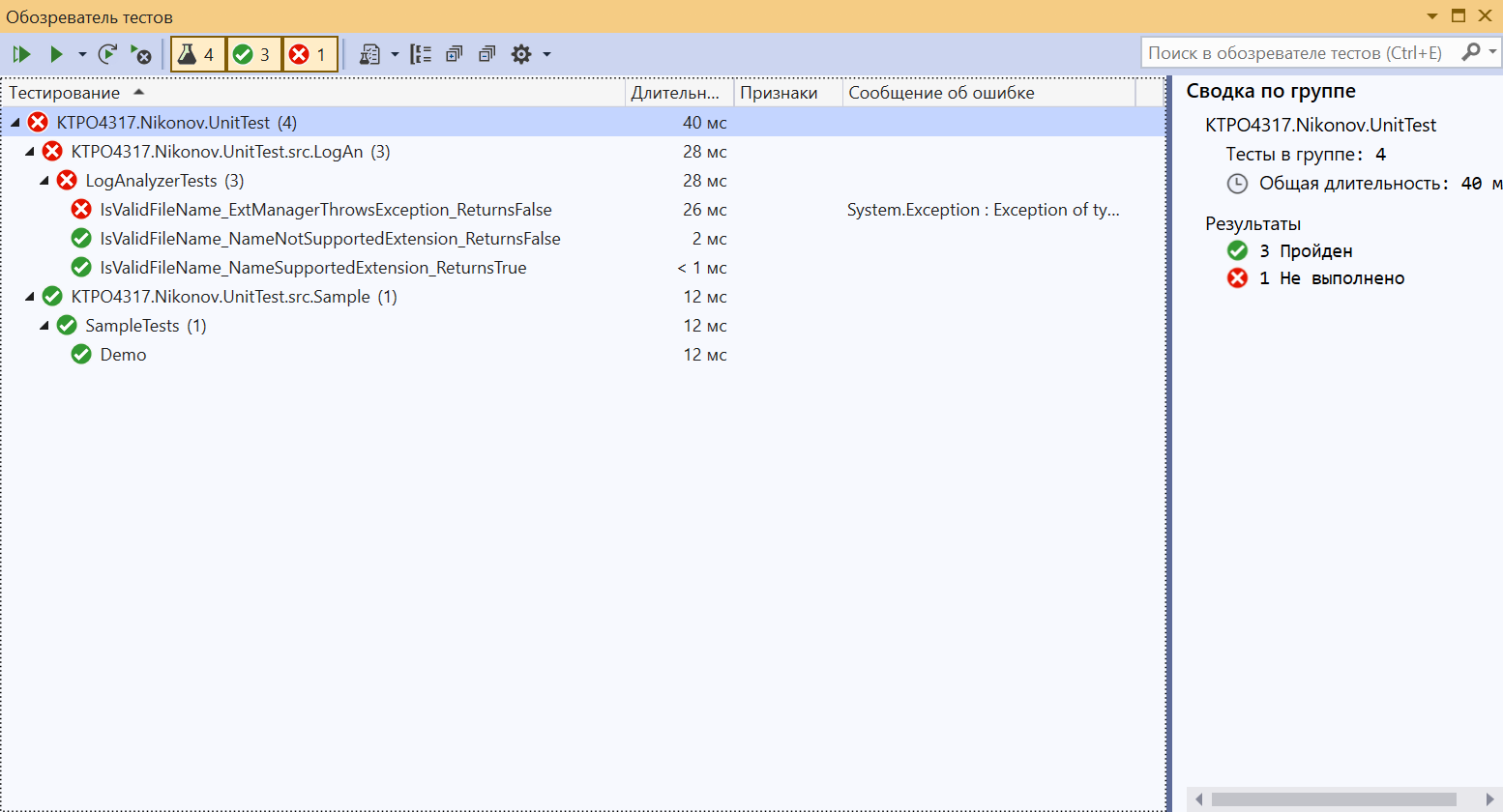
Доработайте метод IsValid заглушки таким образом, что если поле WillThrow инициализировано, то вызывается данное исключение.



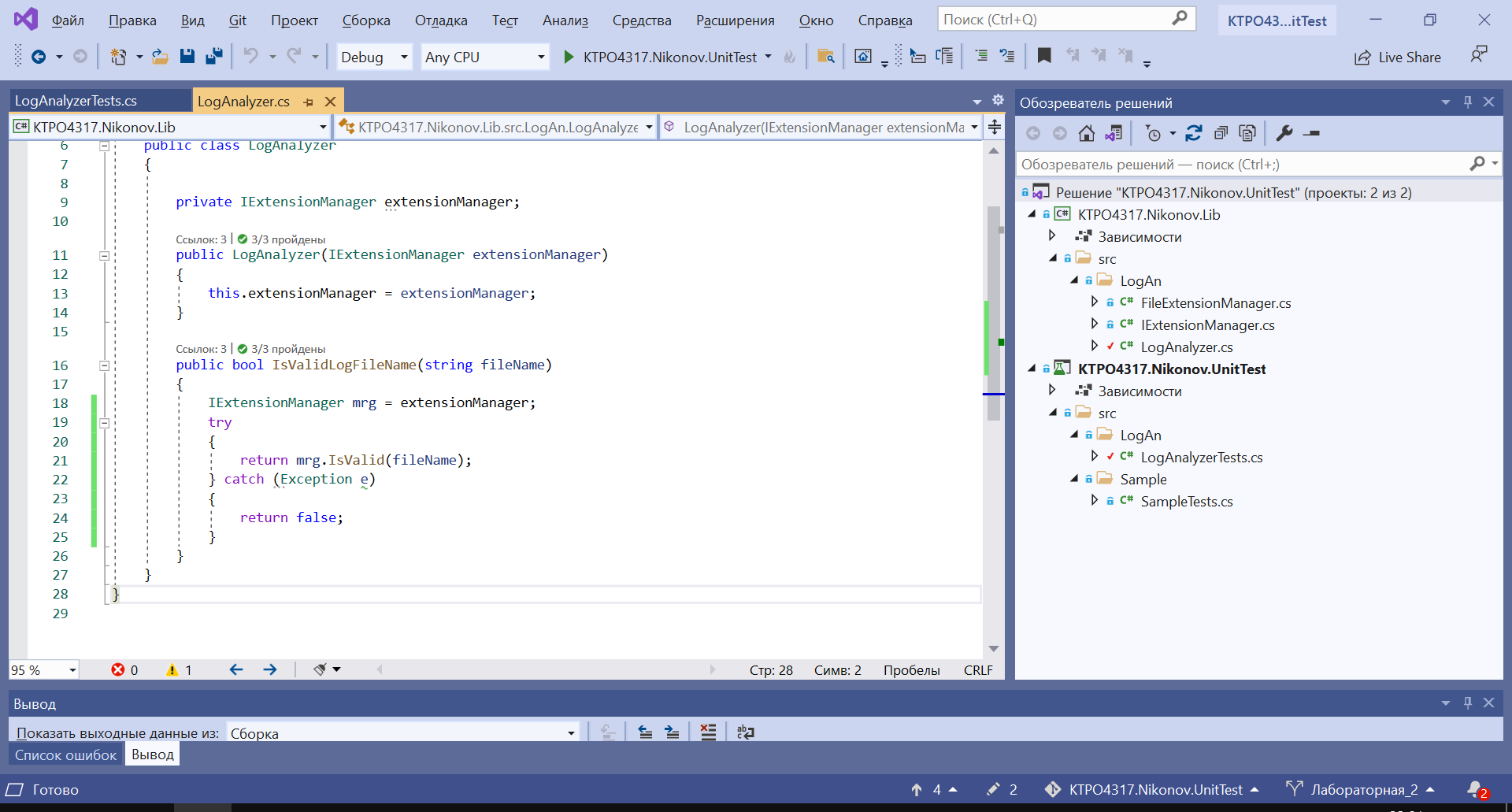
Создайте тестовый метод для сценария «Если диспетчер расширений вызвал исключение, вернуть false»: IsValidFileName\_ExtManagerThrowsException\_ReturnsFalse

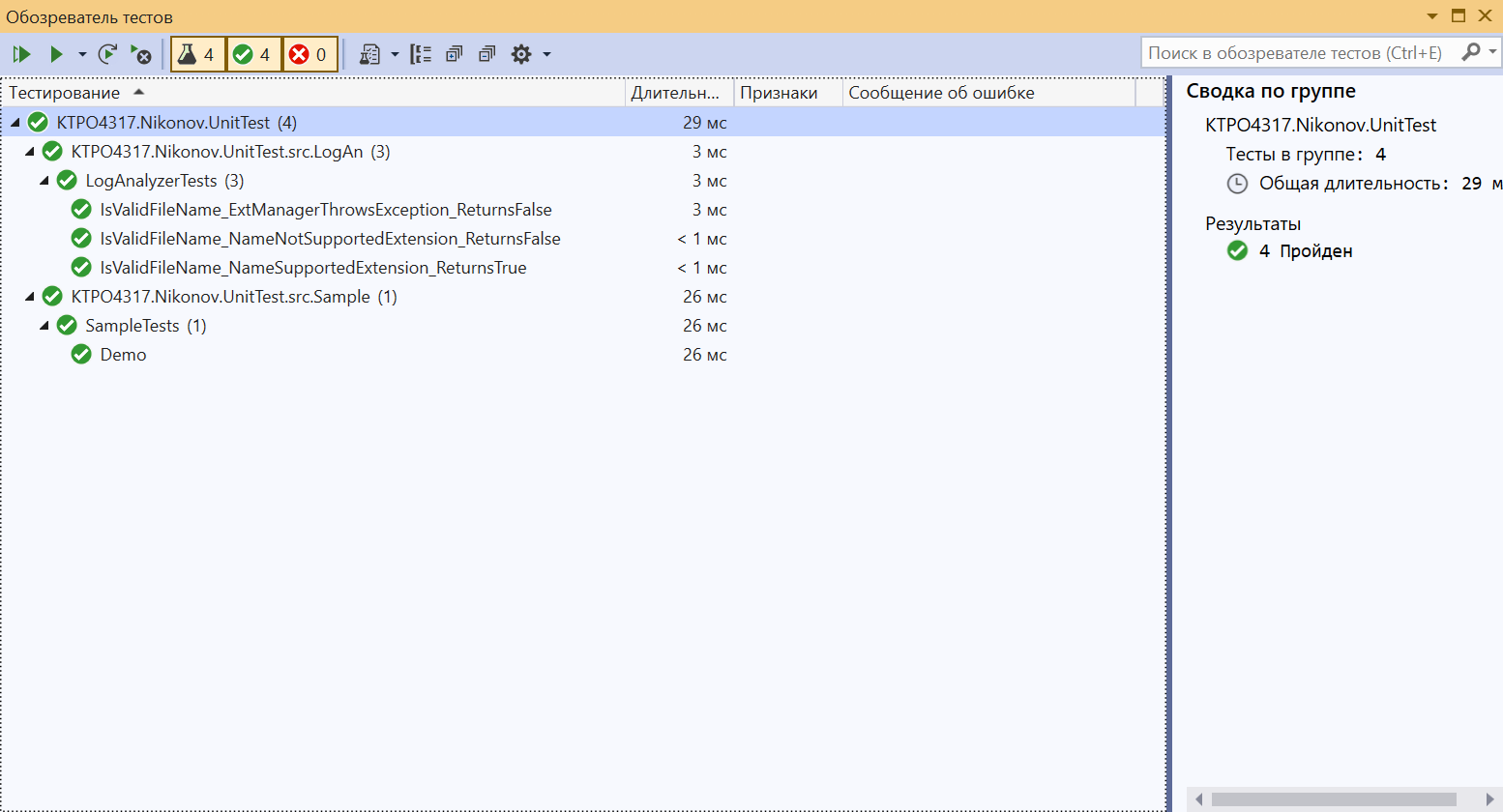


Выполните тесты.

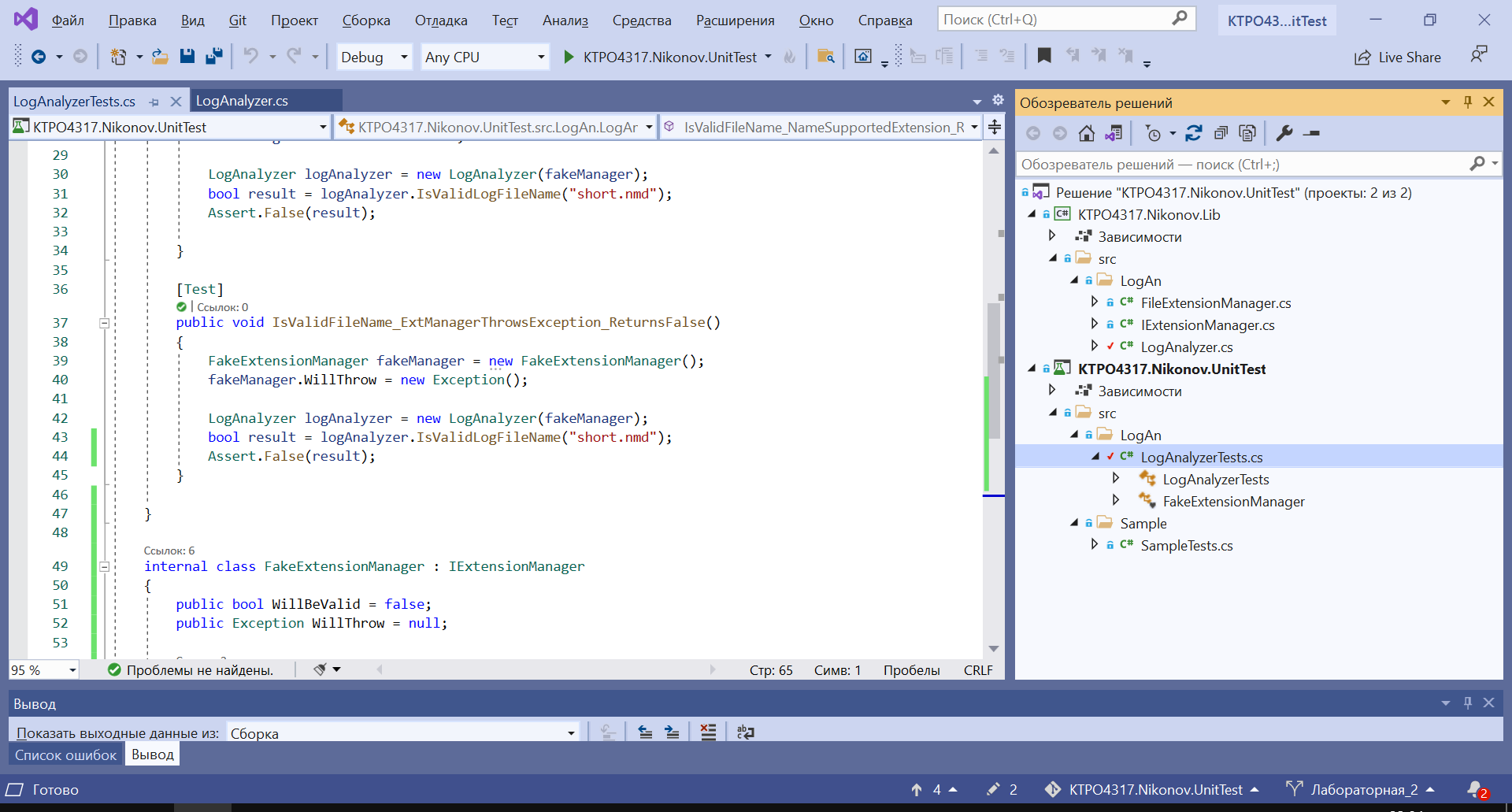


Доработайте тестируемый метод, так , чтобы новый тест стал проходить





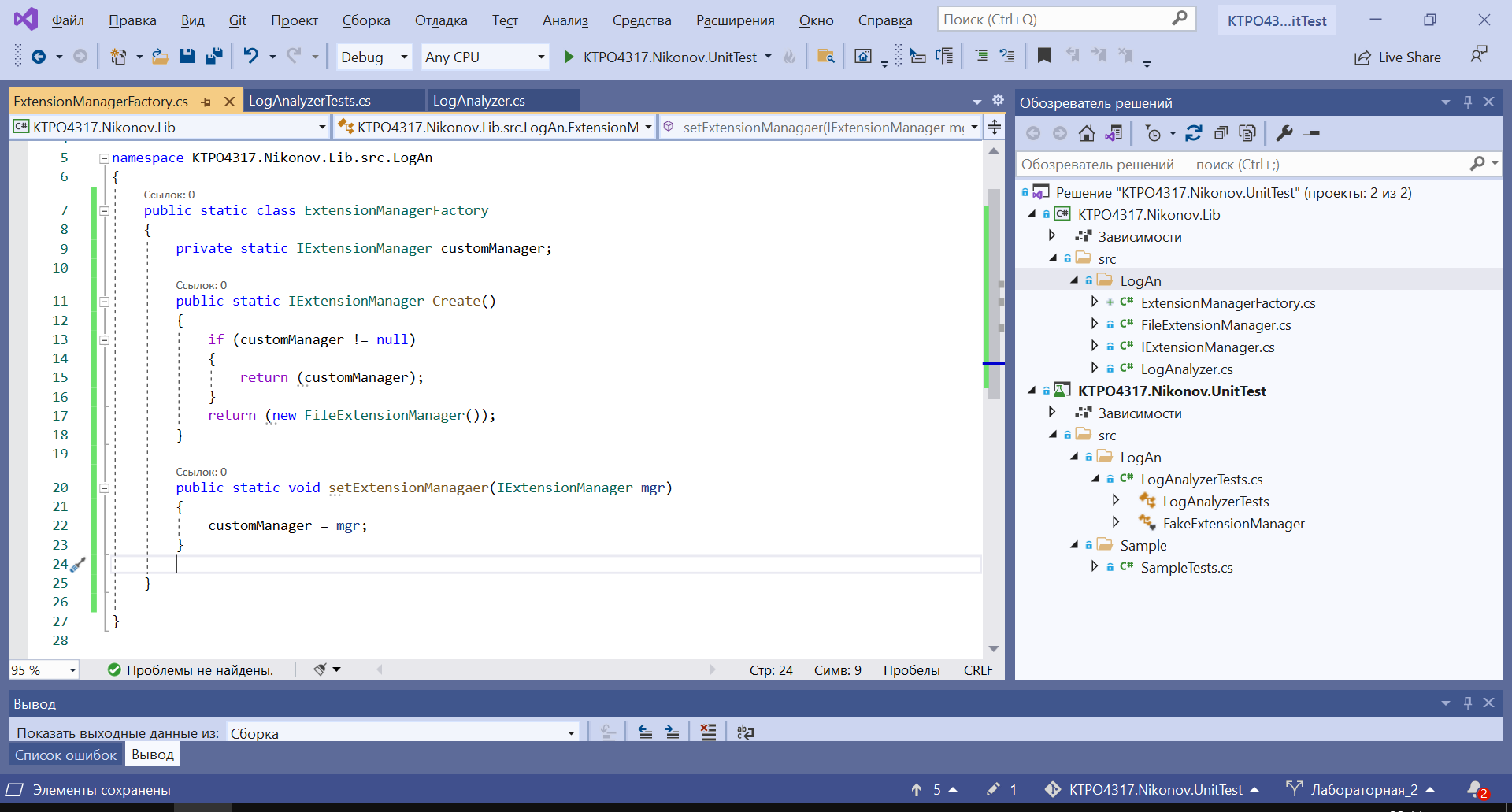
Состояние решения



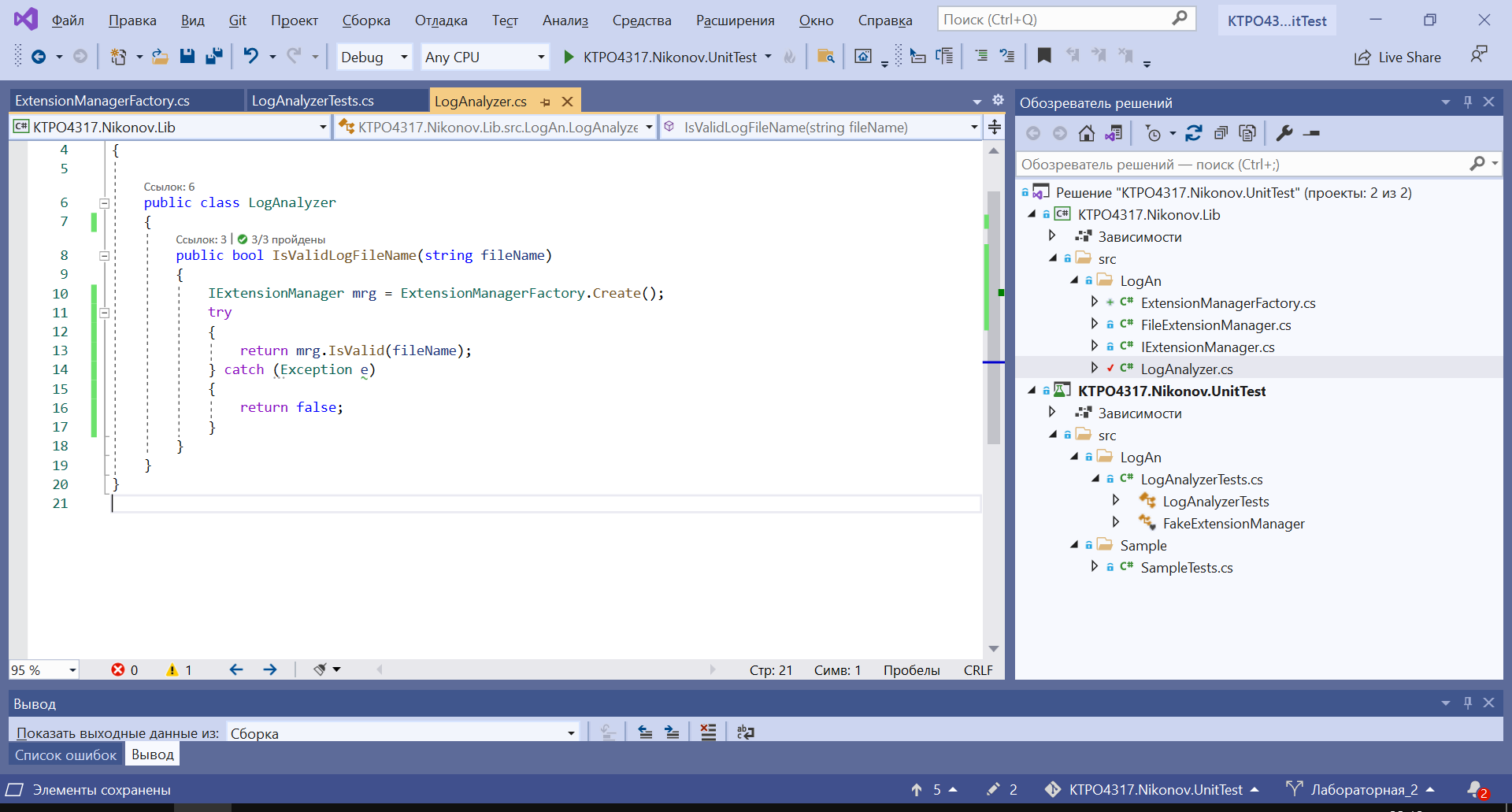
**6. Внедрение зависимости. Внедрение подделки через фабрику.**

Создайте в проекте Lib в папке LogAn фабричный класс ExtensionManagerFactory. Для упрощения воспользуемся статическим классом.

Объявите в нем метод Create отвечающий за создание объекта, и метод SetManager, позволяющий внедрить в фабрику поддельный объект.

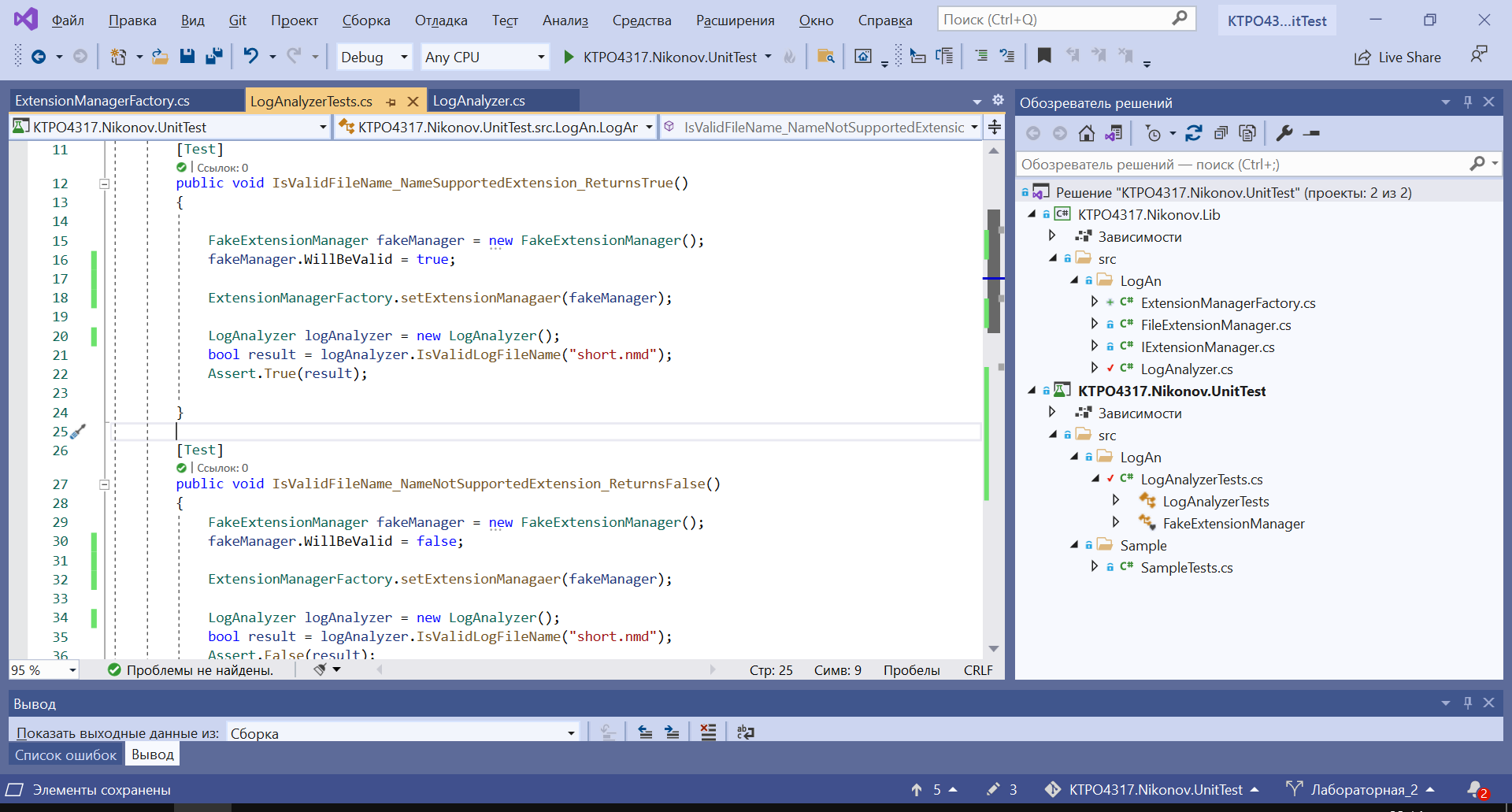


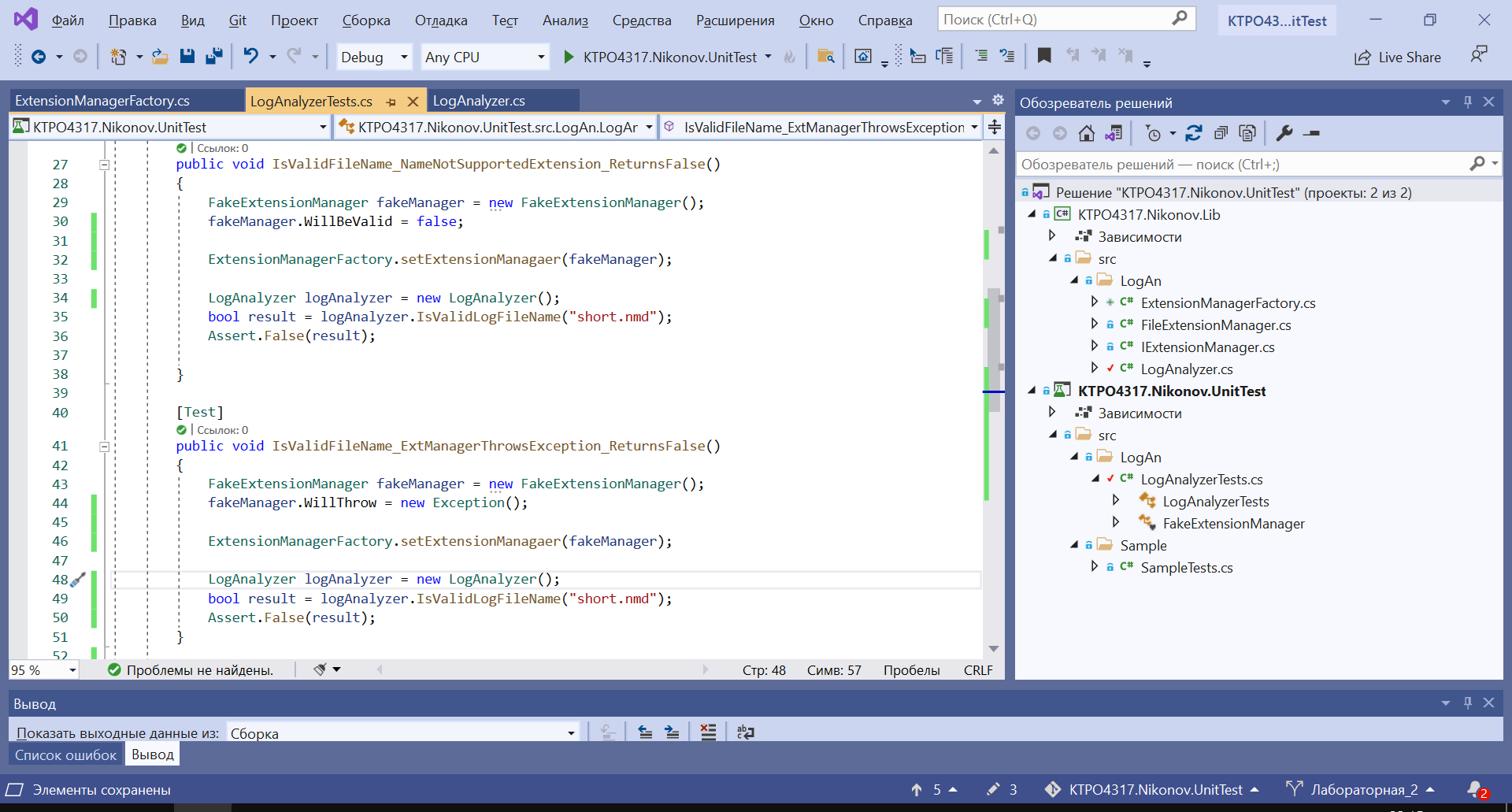
В классе LogAnalyzer удалите параметры конструктора и поле для диспетчера расширений. Диспетчер расширений получите из фабрики. Тесты перестанут компилироваться.



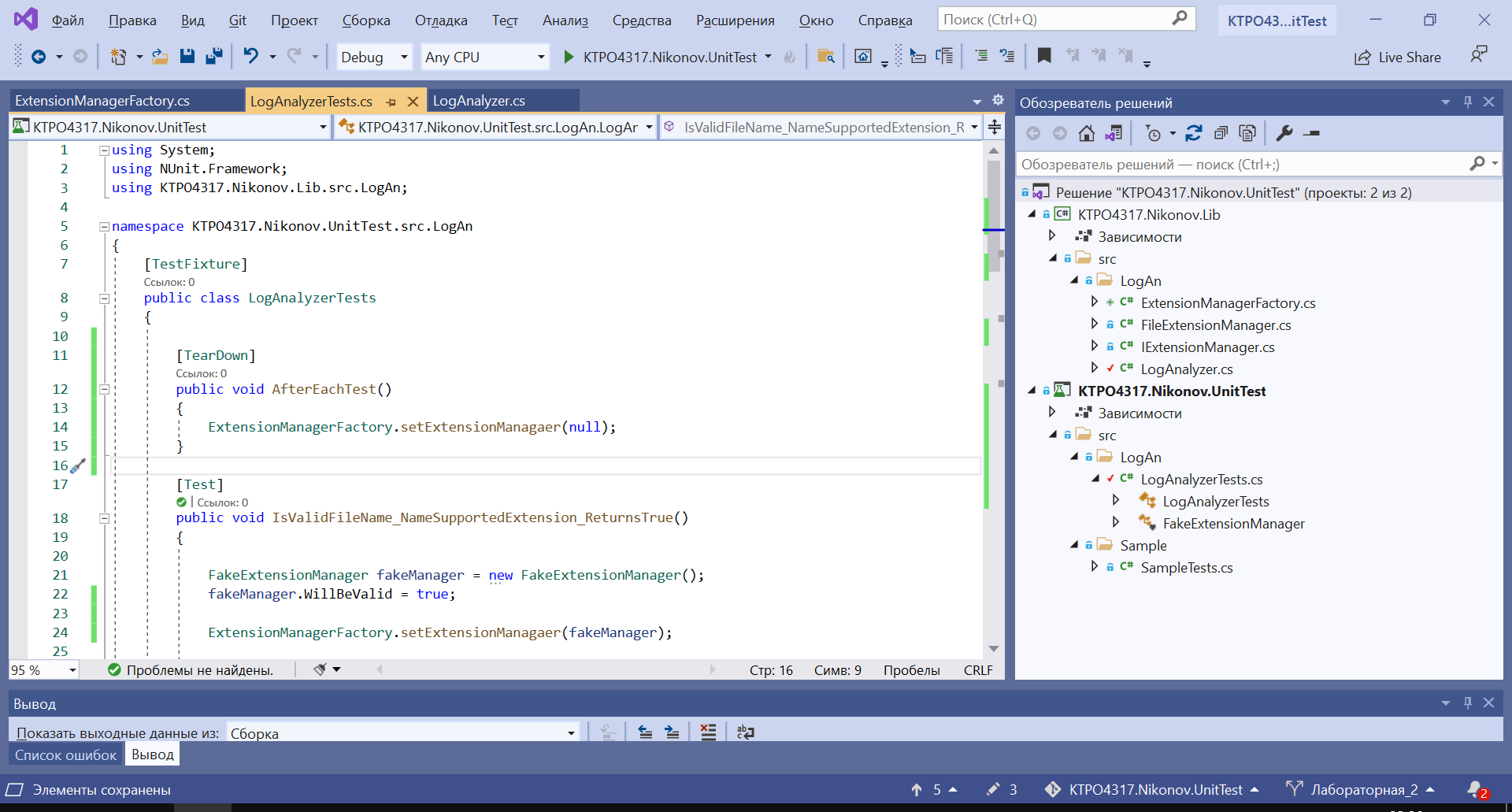


Исправьте тесты: добавьте конфигурирование фабричного класса поддельным диспетчером расширений с помощью метода SetManager.

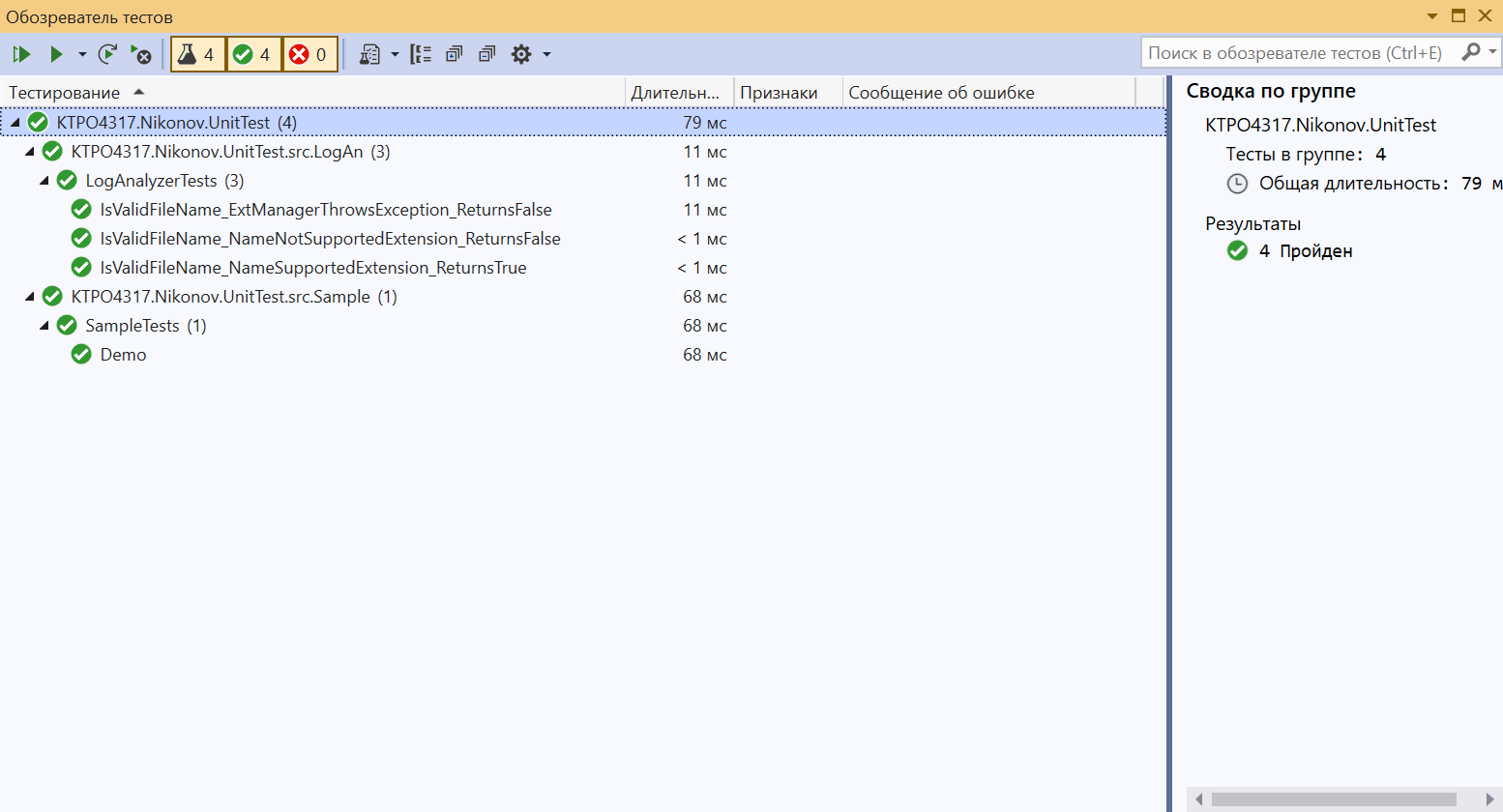




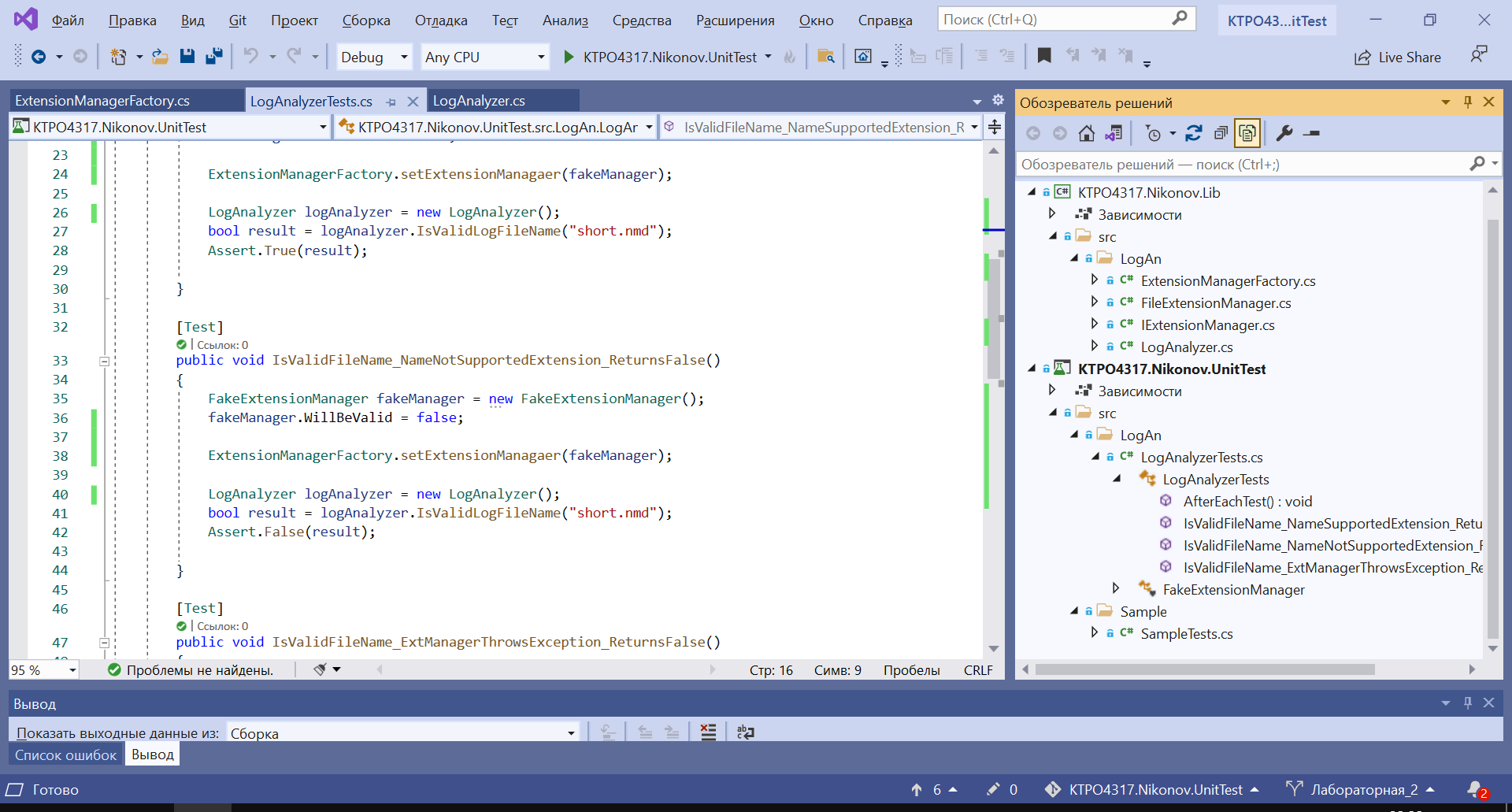
Поскольку в тесте изменяли глобальное состояние системы, необходимо выполнять возврат в исходное состояние после каждого теста. Для этого добавьте в тестовый класс метод AfterEachTest отмеченный атрибутом [TearDown]. В данном методе вызовите метод SetManager фабричного класса со значение null.



Выполните тесты



Состояние решения:



**7. Реализовать диспетчер расширений и клиентское приложение.**

Добавьте в решение новый проект консольного приложения. Имя проекта укажите KTPO4317.Nikonov.Service

Организуйте проект согласно принятым ранее соглашениям.

Добавьте в проект файл настроек. Добавьте в него информацию о правильно расширении.

Реализуйте логику класса FileExtensionManager, так, чтобы допустимое расширение читалось из файла настроек.

Добавьте в KTPO4317.Nikonov.Service вызов класса LogAnalyzer для разных имен файлов, с правильными и не правильными расширениями. Результат вызова выведете на консоль.

Протестируйте и отладьте приложение.

Запустите тесты.

