**Лабораторная работа №5**

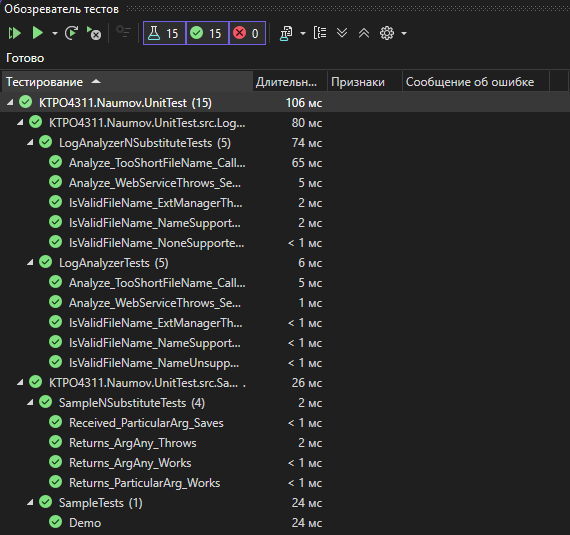
**Тестирование событий**

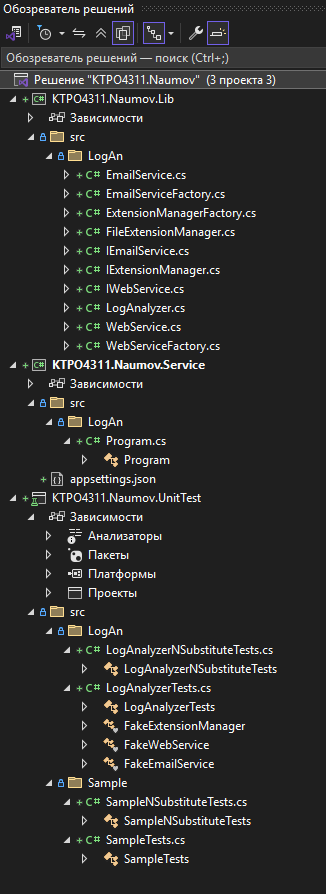
**Цель работы:**

Приобретение практических навыков использования делегатов (delegate) и событий (event); тестирования операций, связанных с событием; практика использования тестового каркаса NUnit, практика использования изолирующего каркаса NSubstitute.

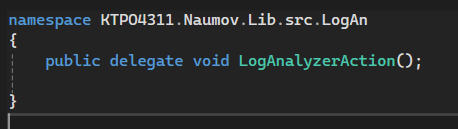
**Порядок выполнения работы**

1. **Подготовка проекта**
2. Берём решение, полученное в результате выполнения лабораторной работы № 4
3. Выполним тесты.
4. Фиксируем исходное состояние проекта.

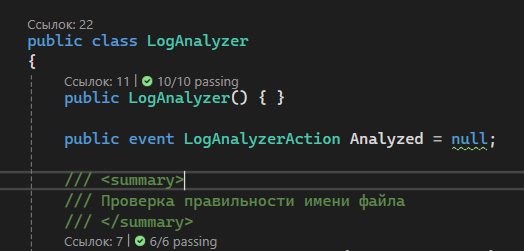




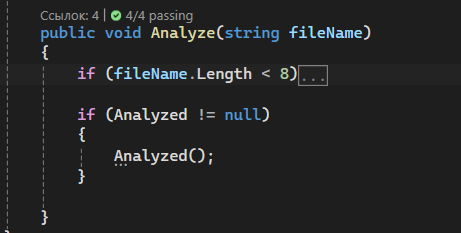
1. **Генерация события**
2. В проекте .Lib объявим делегат



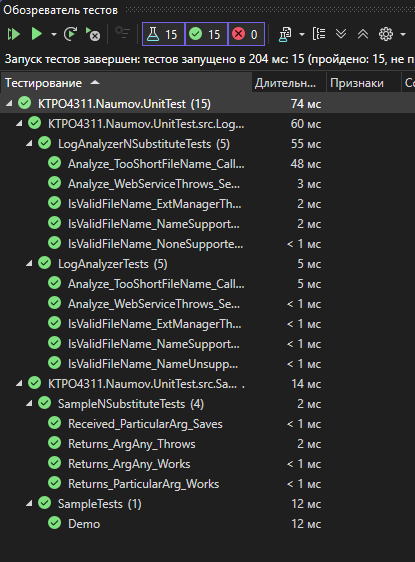
1. Объявляем событие в LogAnalyzer



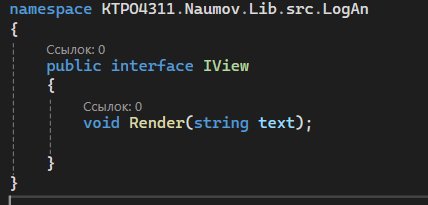
1. Добавляем вызов события в метод Analyze



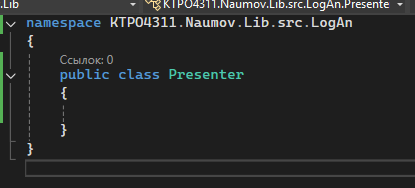
1. Выполним тесты.



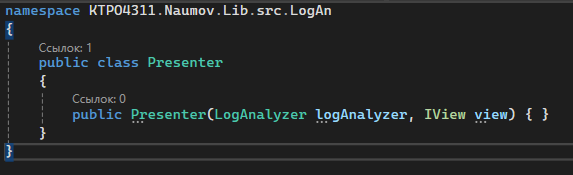
1. **Подписка объекта на события**
2. Добавим в проект .Lib интерфейс IVIew для представления. Создадим в этом интерфейсе метод Render.



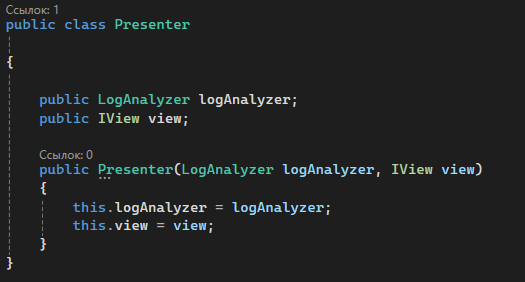
1. Создадим в этом же проекте класс Presenter.



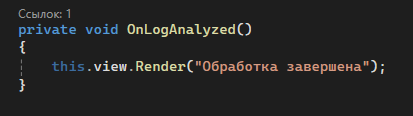
1. Реализуем в нём конструктор с 2-мя параметрами.



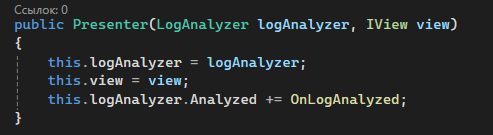
1. Добавим два поля в класс и инициализируем их в конструкторе.



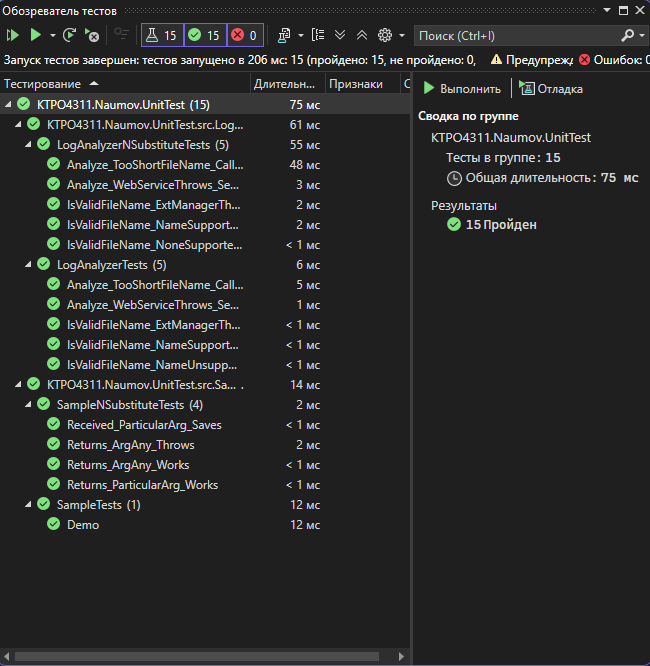
1. Создаём приватный метод OnLogAnalyzed(). Добавляем в него вызов обращение к представлению.

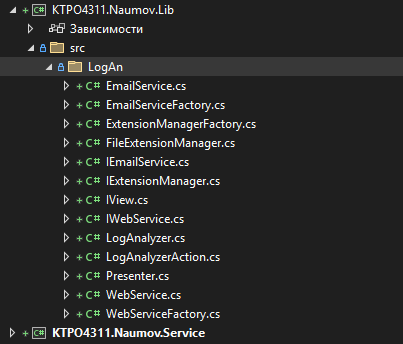


1. В конструкторе добавляем подписку на событие.

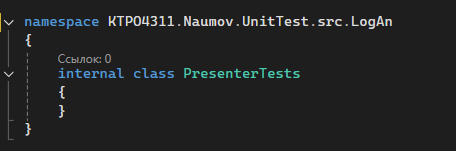


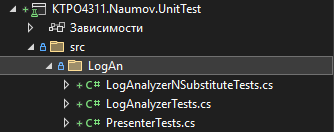
1. Выполним тесты



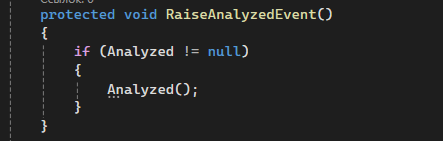


1. **Тестирование и прослушивание события. Введение зазора с помощью наследования.**
2. Создаём класс PresenterTests

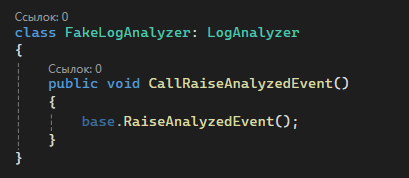




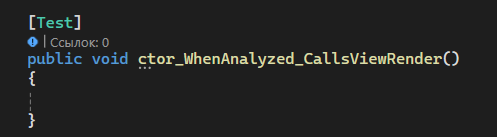
1. Создаём метод RaiseAnalyzedEvent() в LogAnalyzer.



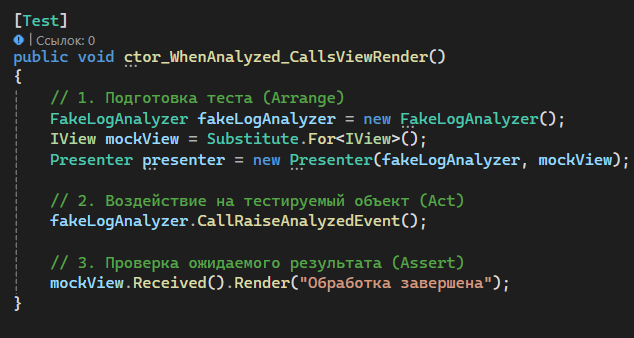
1. Создаём поддельный объект в файле PresenterTests.cs.



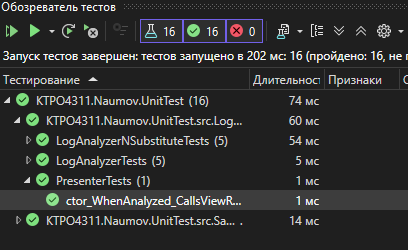
1. Создаём тестовый метод ctor\_WhenAnalyzed\_CallsViewRender()

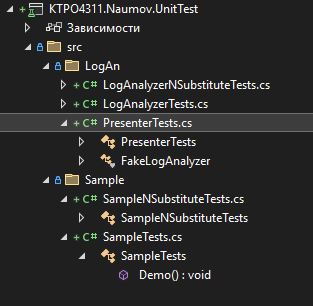


1. Реализуем тест:

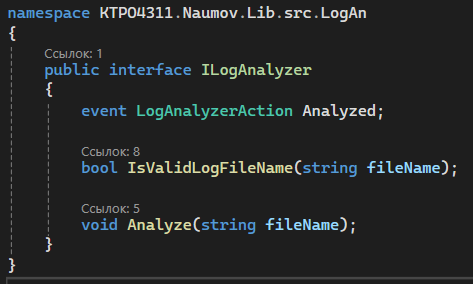


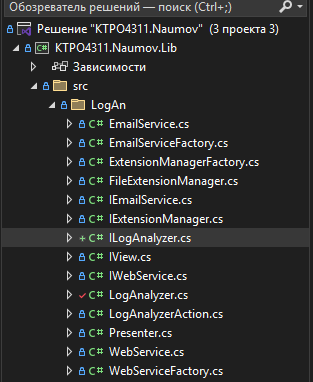
1. Выполним тесты

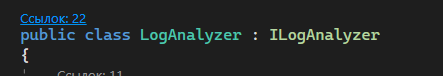




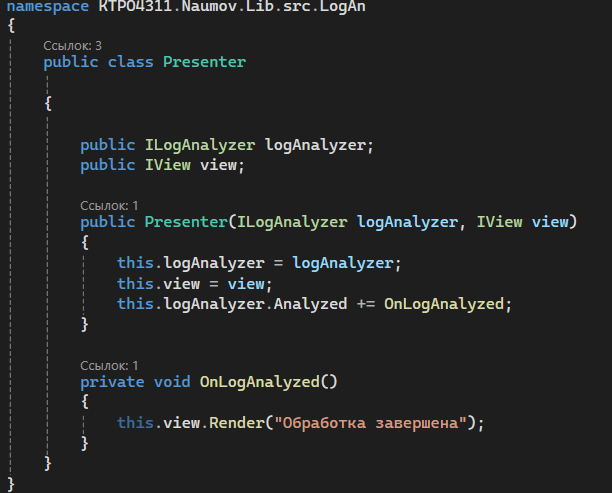
1. **Имитация события с помощью NSubstitute.**
2. Создаём интерфейс ILogAnalyzer.



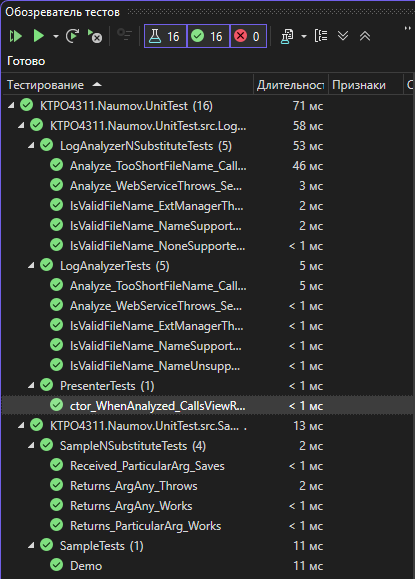




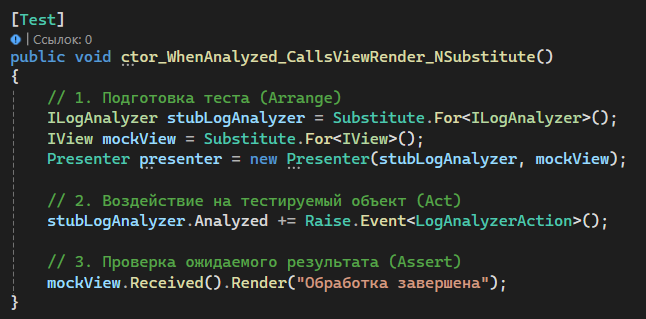
1. Заменяем использование класса LogAnalyzer на интерфейс в классе Presenter.



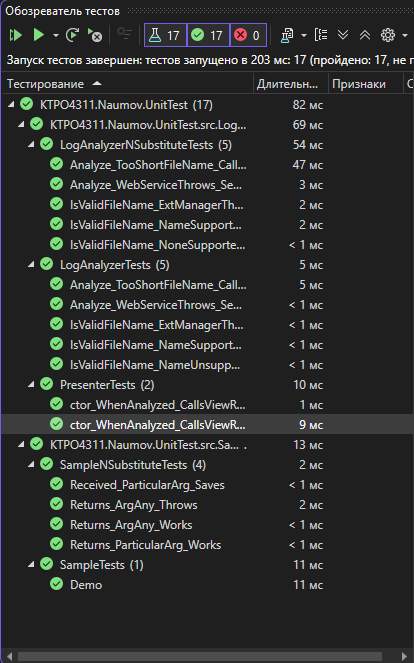
1. Выполним тесты.



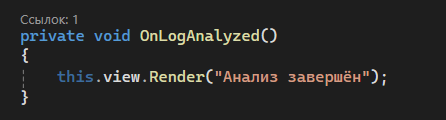
1. Создаём в PresenterTests тестовый метод ctor\_WhenAnalyzed\_CallsViewRender\_NSubstitute()

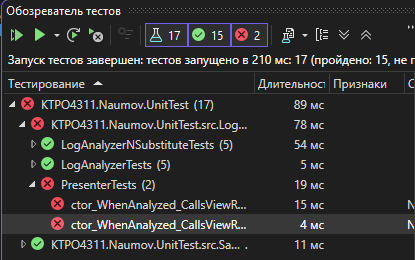


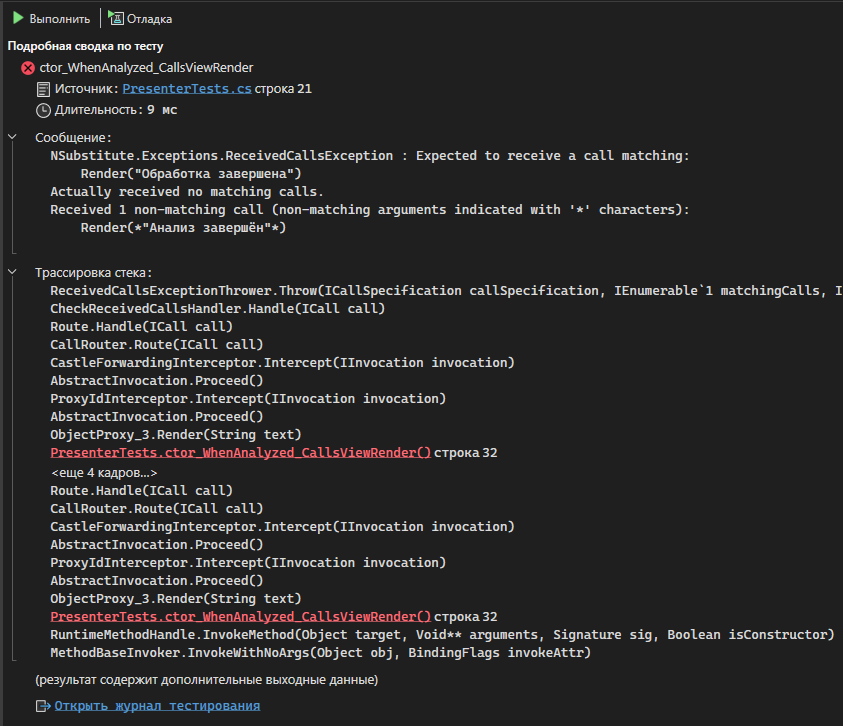
1. Выполним тесты.

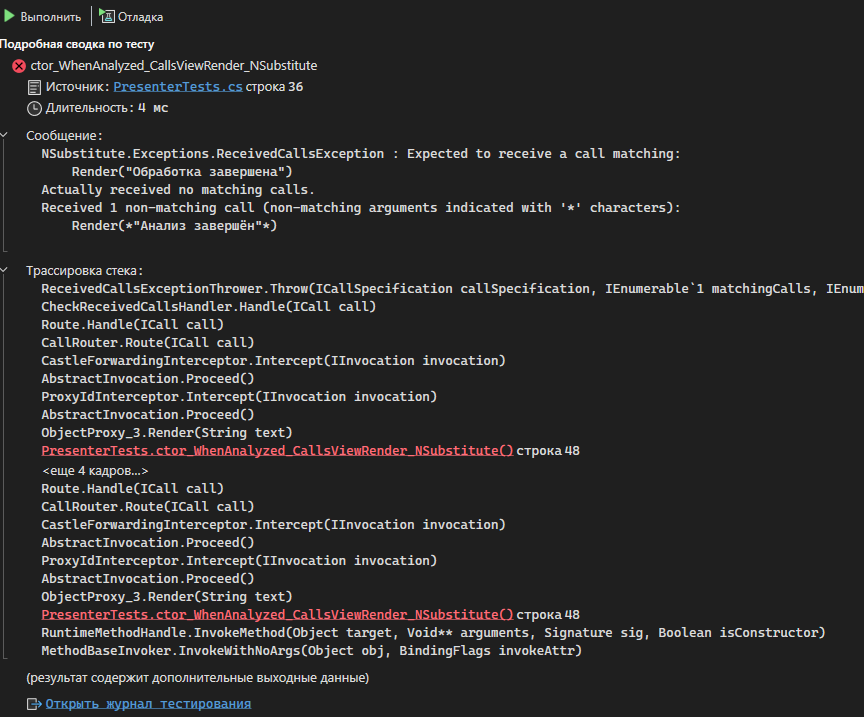


1. Проверим правильность тестовых методов, внеся дефект в тестируемый метод.
2. Изменим текст сообщения “Обработка завершена” на “Анализ завершён”.

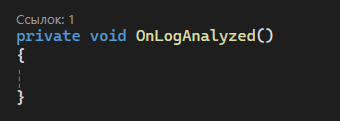


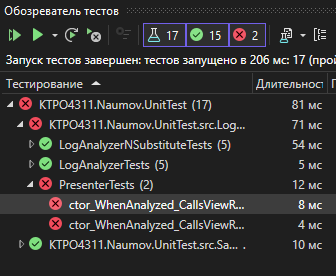


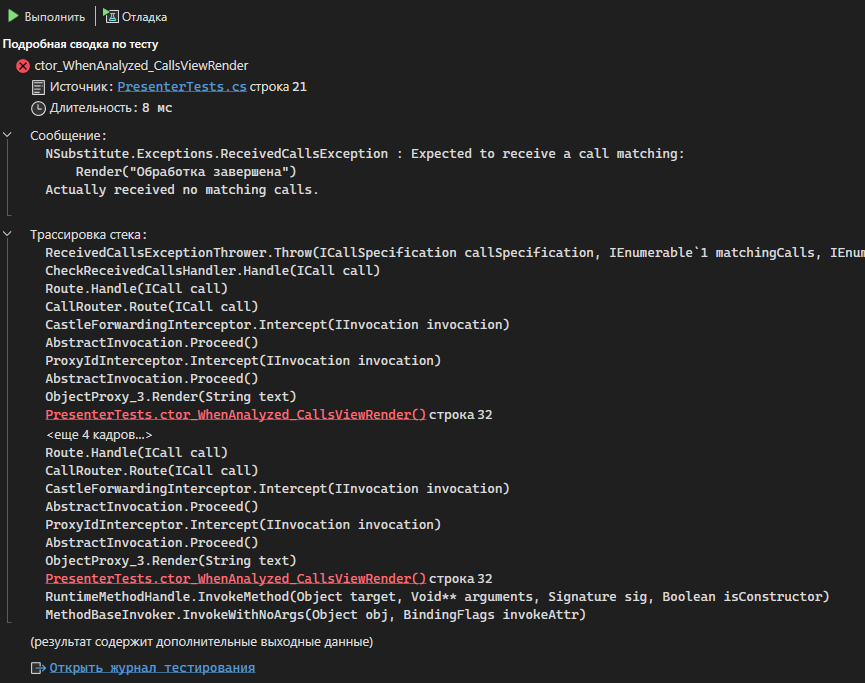


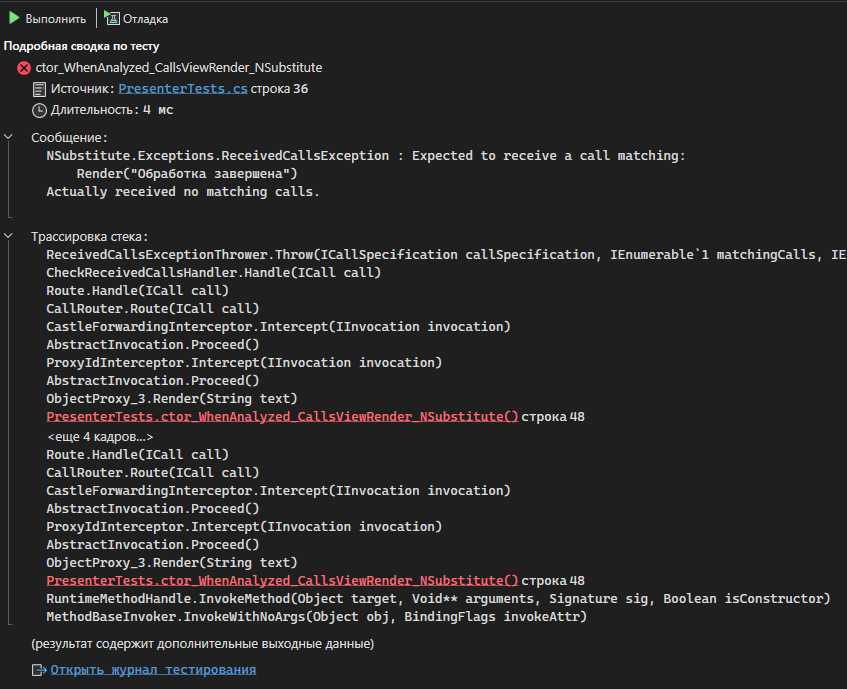


1. Уберём полностью вызов метода Render().

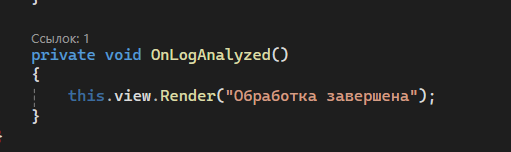


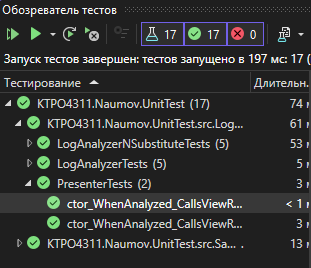


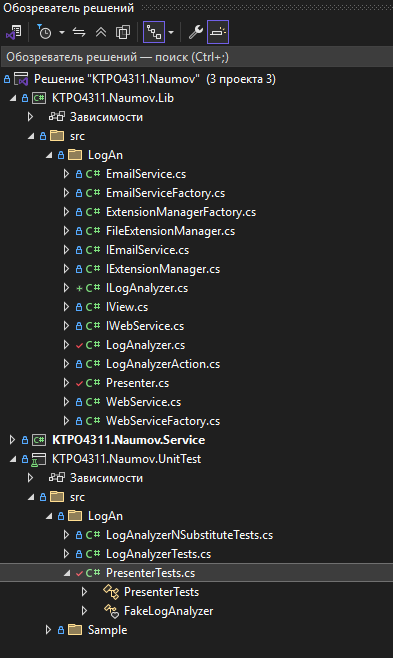




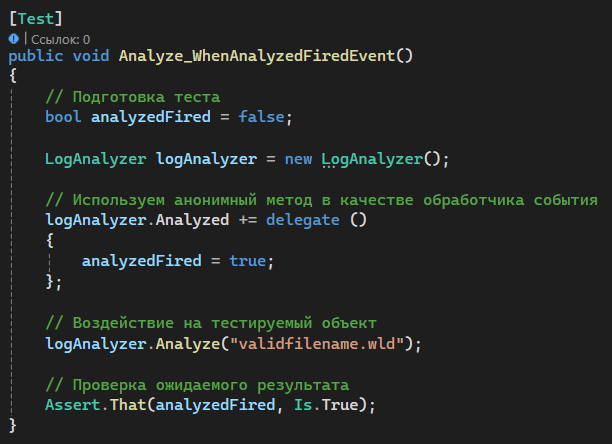
1. Восстановим правильный код и выполним тесты.



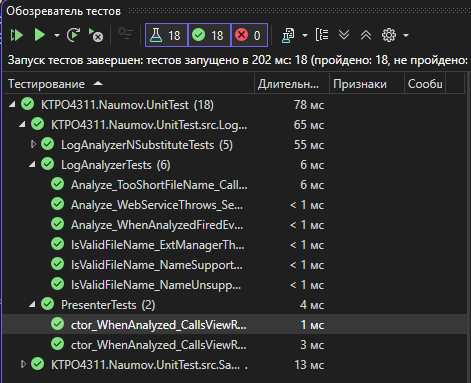




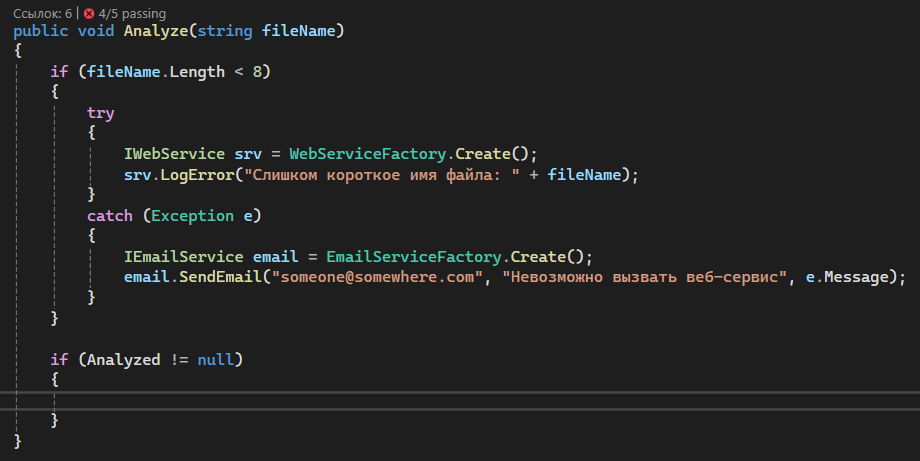
1. **Тестирование факта генерации события.**
2. Создаём тестовый метод для сценария “Если файл проанализирован, то вызывается событие” Analyze\_WhenAnalyzed\_FiredEvent в классе LogAnalyzerTests.

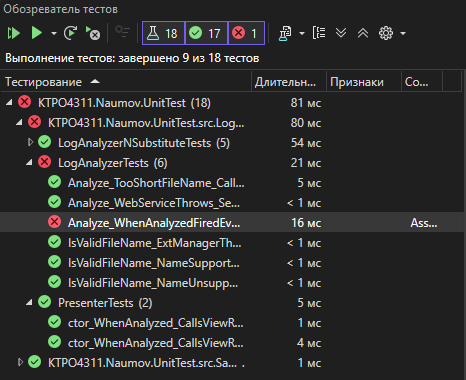


1. Выполним тесты

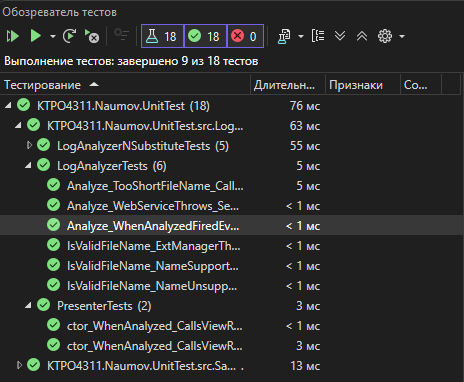


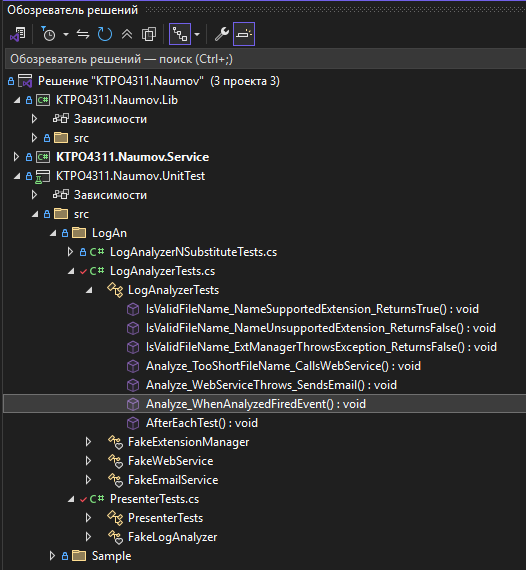
1. Проверим правильность тесты, внеся дефект в тестируемый метод.
2. Уберём вызов события.





1. Восстановим правильный код и выполним тесты.





**Выводы**

В ходе лабораторной работы №5 были приобретены практические навыки использования делегатов и событий, а также тестирования операций, связанных с событиями, с применением тестового каркаса NUnit и изолирующего каркаса NSubstitute. Цель работы заключалась в изучении и реализации механизма генерации событий в классе LogAnalyzer, подписки на события через класс Presenter, а также создании тестов для проверки корректности обработки событий. Были реализованы сценарии, включая ручную имитацию событий с использованием наследования и динамическую имитацию с помощью NSubstitute, что позволило обеспечить изолированное тестирование модулей.