|  |  |
| --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное АВТОНОМНОЕ образовательное учреждение высшего образования  «Национальный исследовательский ядерный университет «МИФИ» | МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное АВТОНОМНОЕ образовательное учреждение высшего образования  «Национальный исследовательский ядерный университет «МИФИ» |
| **Димитровградский инженерно-технологический институт –**  филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский ядерный университет «МИФИ»  **(ДИТИ НИЯУ МИФИ)** | **Димитровградский инженерно–технологический институт –**  филиал федерального государственного автономного образовательного учреждения высшего  профессионального образования «Национальный исследовательский ядерный университет «МИФИ»  **(ДИТИ НИЯУ МИФИ)** |

Факультет Информационных технологий и социально-экономических наук

Кафедра Информационных технологий

Направление подготовки 09.05.01 Применение и эксплуатация автоматизированных систем специального назначения

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ**

«Практический анализ комплекса программ для проверки расписания»

Студент

Группы АС-41 Т.И. Иванова

(подпись)

Руководитель

Старший преподаватель В.Л. Русаков

(подпись)

Заведующий кафедрой

доцент, к.п.н. О.А. Ракова

(подпись)

Димитровград 2023

|  |  |
| --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное АВТОНОМНОЕ образовательное учреждение высшего образования  «Национальный исследовательский ядерный университет «МИФИ»  **Димитровградский инженерно-технологический институт –**  филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский ядерный университет «МИФИ»  **(ДИТИ НИЯУ МИФИ)** | МИНИСТЕРСТВО ОБРАЗОВАНИЯ И  НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное АВТОНОМНОЕ образовательное учреждение высшего образования  «Национальный исследовательский ядерный университет «МИФИ» |
|  | **Димитровградский инженерно–технологический институт –**  филиал федерального государственного автономного образовательного учреждения высшего  профессионального образования «Национальный исследовательский ядерный университет «МИФИ»  **(ДИТИ НИЯУ МИФИ)** |

**ФАКУЛЬТЕТ** Информационных технологий и социально-экономических наук

**КАФЕДРА** Информационных технологий

**НАПРАВЛЕНИЕ ПОДГОТОВКИ** 09.05.01 **ГРУППА** АС-41

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

1.Фамилия, имя, отчество Иванова Татьяна Игоревна

2.Тема проекта «Практический анализ комплекса программ для проверки расписания»

3. Срок сдачи студентом готового проекта «02» июня 2023 г.

4. Руководитель курсового проекта Русаков Владимир Леонидович

Дата выдачи задания « 07» февраля 2023 г.

Руководитель курсового проекта

(подпись руководителя)

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата и подпись студента)

1. Исходные данные к проекту:

требования к уровню разработки программного продукта:

*а) современный пользовательский интерфейс;*

*б) язык программирования С#*

1. Содержание работы:

Введение;

*а) теоретическая часть:*

* + - *Microsoft Visual Studio.*
    - *Комплекс программных продуктов для проверки расписания;*

*б) практическая часть:*

* + - *Анализ рассмотренного комплекса программных продуктов ;*
    - *Разработка программного продукта;*
    - *Тестирование программного продукта.*

1. Основная литература по теме проекта:
2. Язык C# и платформа .NET : [Электронный ресурс] // Metanit. Режим доступа: https://metanit.com/sharp/tutorial/1.1.php (Дата обращения: 25.02.2023 г.);
3. Архитектура .NET [Электронный ресурс] // Microsoft . Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/ (Дата обращения: 25.02.2023 г.);
4. Руководство по WPF : [Электронный ресурс] // Metanit. Режим доступа: https://metanit.com/sharp/wpf/ (Дата обращения: 25.02.2023 г.);
5. Отчетный материал проекта:

*а) пояснительная записка;*

*б) исходники приложения и исполняемый файл;*

*в) компакт-диск с материалами работы.*

РЕФЕРАТ

Иванова Т.И. «Практический анализ комплекса программ для проверки расписания»: Курсовой проект, ДИТИ НИЯУ МИФИ №09.05.01.87.2023 – Димитровград, 2023. – стр. 38, рис. 26, библ. назв. 7, приложений 1.

Ключевые слова: ПРОГРАММНЫЙ ПРОДУКТ, С#, WPF, проверка расписания, комплекс программ.

Объект исследования – комплекс программ.

Цель проекта – разработать программный продукт на основе анализа ошибок ранее созданного комплекса программ.

Выводы: во время разработки программного продукта был проведен анализ ранее разработанного программного комплекса на предмет недоработок и ошибок, были изучены теоретические и практические аспекты Visual Studio, С#, WPF. Проведено проектирование, разработка плана и тестирование конечного продукта. Результатом данной работы стал программный продукт, позволяющий проверить корректность учебного расписания занятий в ВУЗе.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc135304160)

[1 Теоретические аспекты разработки программного продукта проверки расписания 6](#_Toc135304161)

[1.1 Комплекс программных продуктов для проверки расписания 6](#_Toc135304162)

[1.2 Microsoft Visual Studio и язык программирования C# 7](#_Toc135304163)

[1.3 Система управления версиями. 9](#_Toc135304164)

[2 Практические аспекты разработки программного продукта проверки расписания 10](#_Toc135304165)

[2.1 Анализ рассмотренного комплекса программных продуктов 10](#_Toc135304166)

[2.2 Разработка программного продукта 17](#_Toc135304167)

[2.3 Тестирование программного продукта 30](#_Toc135304168)

[2.4 Перспективы развития 33](#_Toc135304169)

[ЗАКЛЮЧЕНИЕ 35](#_Toc135304170)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 36](#_Toc135304171)

[Приложение А Unit Тест Модуля поиска в расписании занятий у разных преподавателей, проводимых одновременно в одной аудитории 37](#_Toc135304172)

# ВВЕДЕНИЕ

Составление расписания в вузах – крайне важная работа, т.к. именно качественно составленное расписание позволяет студентам полноценно вливаться в процесс обучения, а преподавателям – работать в комфортных условиях. Расписание, составленное вручную, часто имеет большое количество ошибок, которые нарушают процесс обучения и усвоения материала студентами, создают неудобства для работы преподавателей.

С целью предотвращения негативных последствий составления расписания студенты 3 курса ДИТИ НИЯУ МИФИ подготовили комплекс программ. Однако, в связи с отсутствием опыта коллективной разработки проекта и коммерческой разработки в целом, в представленном комплексе имеется ряд недоработок и ошибок, не позволяющих полноценно использовать его для решения поставленной задачи.

Целью данной курсовой работы является разработка программного продукта на основе анализа ошибок и недоработок ранее созданного комплекса программ.

Для достижения цели были поставлены следующие задачи:

* Изучить литературу по теме работы;
* Изучить комплекс программ;
* Провести анализ программного комплекса на наличие недоработок, неточностей и ошибок;
* Разработать программный продукт, устраняющий описанные проблемы;
* Провести тестирование разработанного программного продукта.

# Теоретические аспекты разработки программного продукта проверки расписания

## Комплекс программных продуктов для проверки расписания

Программное обеспечение, предоставленное для изучения, представляет из себя набор из 11 консольных приложений и одного командного файла .bat. Консольные приложения написаны на языке программирования C#, 9 из них используют целевую платформу .Net 6.0, 2 – .Net Core 3.1.

Для работы с файлами формата .xlsx и .xls была использована встроенная в Microsoft Office 2016 библиотека Microsoft Office 16.0 Object Library.

Программный комплекс на данный момент представлен в виде 11 проектов Visual Studio, он не имеет релизной версии. Тестирование программного комплекса будет производиться методом белого ящика, т.к. для изучения был представлен исходный код.

В данном комплексе отсутствуют модули с тестами, что связано с ручным тестированием написанного кода и недостатком внимания к этому процессу.

В комплекс программ входят следующие модули.

Модуль поиска в расписании занятий у разных преподавателей, проводимых одновременно в одной аудитории, чей код представлен в Приложении A (здесь и далее в этом разделе буквенная идентификация приложений относится к приложениям к отчёту по рассматриваемому комплексу программ).

Модуль поиска несоответствий аудиторного фонда типу занятий, чей код представлен в Приложении Б. За ошибку считается проведение лабораторного занятия в аудиториях, не оснащённых компьютерами. В случае проведения практических занятий в аудиториях, не оснащённых компьютерами, выдаётся предупреждение.

Модуль поиска недостатка в количестве посадочных мест для проведения занятий, чей код представлен в Приложении В.

Модуль считывания номеров всех аудиторий и типов занятий из файла расписания, чей код представлен в Приложении Г.

Модуль считывания ФИО преподавателей и наименований дисциплин из файла расписания, чей код представлен в Приложении Д.

Модуль вывода сортированного по убыванию списка повторения дисциплин, аудиторий и ФИО преподавателей, чей код представлен в Приложении Е. Коды поиска и сортировки приведены в приложениях Г и Д.

Модуль поиска схожих наименований дисциплин, ФИО преподавателей и аудиторий, чей код представлен в Приложении Ж.

Модуль обработки полученных данных в приложении Ж, чей код представлен в Приложении З. Программный продукт отображает схожие наименования в парах и предлагает выбрать правильный вариант написания.

Модуль подсчёта общего количества названий дисциплин, ФИО преподавателей, номеров аудиторий и наименований типов занятий, чей код представлен в Приложении И.

Модуль считывания файла предупреждений, созданных в приложении Б и вывода списка практических занятий кафедры ИТ, чей код представлен в Приложении Й.

Модуль вывода перечня аудиторий, соответствующих заданному списку аудиторий, чей код представлен в Приложении К.

Модуль последовательного запуска приложений из программного комплекса, чей код представлен в Приложении Л. Представляет собой командный файл. Для запуска приложений релизная версия всех приложений должна находиться в папке с командным файлом.

## Microsoft Visual Studio и язык программирования C#

Интегрированная среда разработки (IDE) – это многофункциональная программа, которая поддерживает многие аспекты разработки программного обеспечения. Интегрированная среда разработки Visual Studio – это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. Помимо стандартного редактора и отладчика, которые есть в большинстве сред IDE, Visual Studio включает в себя компиляторы, средства автозавершения кода, графические конструкторы и многие другие функции для улучшения процесса разработки [1].

Функциональная структура среды включает в себя:

* редактор исходного кода;
* отладчик кода;
* редактор экранных форм;
* веб-редактор;
* дизайнер классов;
* дизайнер схем баз данных.

Visual Studio также позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне [1].

C# – это язык программирования, предназначенный для разработки разнообразных приложений, предназначенных для выполнения в среде .NET Framework. Язык C# прост, типобезопасен и объектно-ориентирован [2].

C# в плане объектно-ориентированности много перенял у Java и С++. Например, C# поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию.

Программы C# выполняются в .NET, виртуальной системе выполнения, вызывающей общеязыковую среду выполнения (CLR) и набор библиотек классов. Среда CLR – это реализация общеязыковой инфраструктуры языка (CLI), являющейся международным стандартом. CLI является основой для создания сред выполнения и разработки, в которых языки и библиотеки прозрачно работают друг с другом [3].

Исходный код, написанный на языке C#, компилируется в промежуточный язык (IL), который соответствует спецификациям CLI. Код на языке IL и ресурсы, в том числе растровые изображения и строки, сохраняются в сборке, обычно с расширением *.dll*. Сборка содержит манифест с информацией о типах, версии, языке и региональных параметрах для этой сборки.

При выполнении программы C# сборка загружается в среду CLR. Среда CLR выполняет JIT-компиляцию из кода на языке IL в инструкции машинного языка. Среда CLR также выполняет другие операции, например, автоматическую сборку мусора, обработку исключений и управление ресурсами. Код, выполняемый в среде CLR, иногда называется управляемым кодом. «Неуправляемый код» преобразуется в машинный язык, предназначенный для конкретной платформы.

## Система управления версиями.

Git – распределённая система управления версиями. Эта система спроектирована как набор программ, специально разработанных с учётом их использования в сценариях. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы. Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки.

Ряд сервисов предоставляют хостинг для git-репозиториев. Для данного проекта был выбран хостинг GitHub, так как GitHub – крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

В качестве визуального клиента используется Sourcetree, который предоставляет доступ к репозиториям Git и Mercurial на базе Windows и Mac и визуализирует процесс разработки без доступа к командной строке.

Выводы: Для реализации проекта было решено использовать язык программирования C# на целевой платформе .Net 6.0. Контроль версий будет осуществляться с помощью Git. Репозиторий проекта располагается на GitHub

# Практические аспекты разработки программного продукта проверки расписания

## Анализ рассмотренного комплекса программных продуктов

Модульное тестирование в комплексе программ отсутствует, так что для исследования будут написаны Unit тесты для некоторых проектов, а также будет произведено тестирование методом белого ящика.

Модульное тестирование выявило следующие недостатки программных продуктов.

Для модуля поиска в расписании занятий у разных преподавателей, проводимых одновременно в одной аудитории были написаны unit тесты для метода удаления лишних пробелов (приложение А).

По результатам проверки корректности работы функции лишь треть тестов завершилась успешно (4 из 12 тестов), 7 тестов показали, что функция не всегда выводит ожидаемый результат, а в одном тесте вовсе выпало необработанное исключение (рис. 2.1).

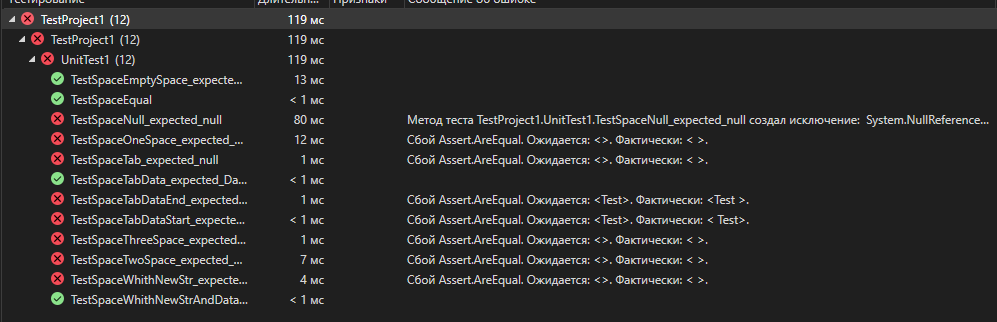


Рисунок 2.1 – Результаты Unit тестирования функции удаления пробелов.

При тестировании методом белого ящика было произведено минимальное редактирование кода для возможности запуска приложения. Это связано с наличием абсолютной ссылки на файл с расписанием. Из-за такого способа задания пути, происходит появление необработанного исключения Runtime.InteropServices.COMException (рис. 2.2).

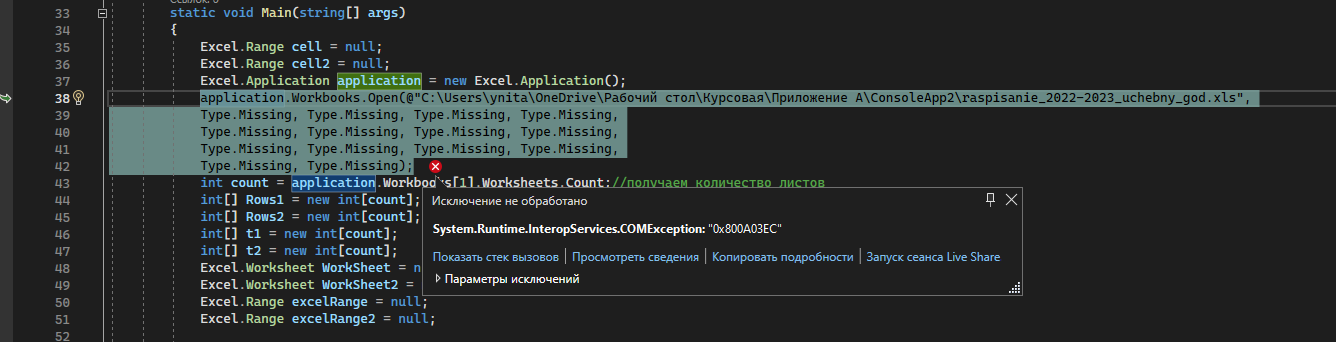


Рисунок 2.2 – Появление исключения Runtime.InteropServices.COMException.

Это делает работу с данным приложением практически неработоспособным на различных машинах. Кроме того, использование Microsoft Office 16.0 Object Library делает невозможным работу приложения на машинах, на которых установлена любая другая версия Microsoft Office, кроме Microsoft Office 2016 (рис. 2.3).

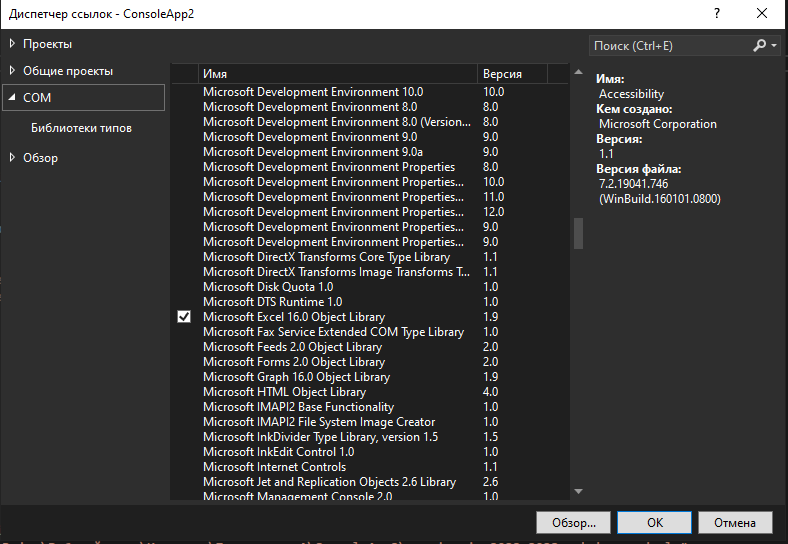


Рисунок 2.3 – Библиотека Microsoft Office 16.0 Object Library в зависимостях проекта.

Полная работа приложения требует в среднем 1 минуту и 40 секунд, занимая при этом около 48 мб памяти на системе с 8 Гб оперативной памяти DDR3, Процессор AMD FX4300, SSD Kingston A400 (рис. 2.4).

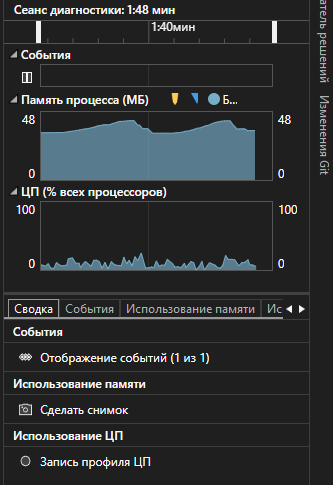


Рисунок 2.4 – Сеанс диагностики и память процесса приложения.

Это является серьёзным недостатком, так как на обработку 6 страниц документа в формате MS Excel уходит слишком много времени и ресурсов. Кроме того, по завершению выполнения приложение уходит в вечный цикл (2.5).

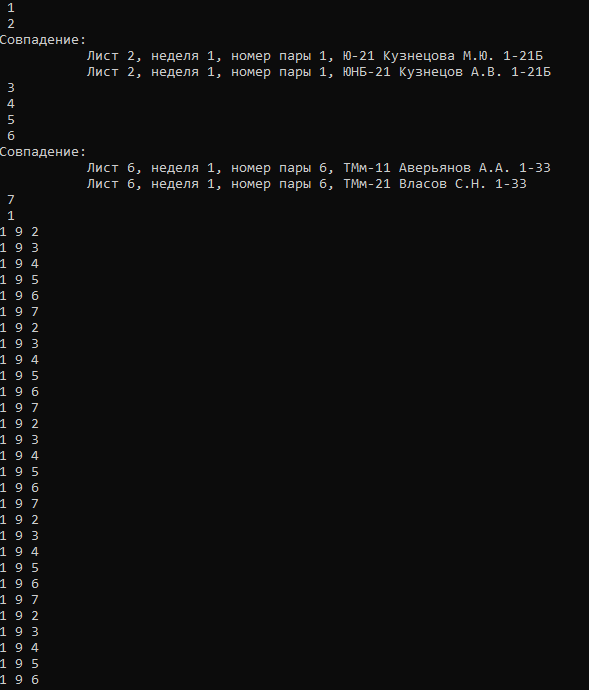


Рисунок 2.5 – Завершение работы модуля переходом в состояние вечного цикла.

Это не позволяет завершить работу приложения корректно. Кроме того, по результатам работы программы не происходит закрытие потока работы с xls-файлом, что создаёт утечку памяти, связанную с открытием нескольких экземпляров приложения Excel (рис.2.6).



Рисунок 2.6 – Утечка памяти в виде открытия в фоновом режиме нескольких экземпляров приложения Microsoft Excel.

Такая утечка приводит к засорению памяти и к невозможности дальнейшей работы с файлом расписания (рис. 2.7).

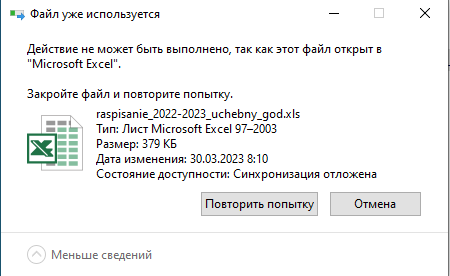


Рисунок 2.7 – Отсутствие возможности дальнейшей работы с файлом расписания.

Для модуля поиска несоответствий аудиторного фонда типу занятий было использовано то же unit тестирование функции удаления лишних пробелов, которое ранее было представлено для предыдущего модуля. Тестирование дало аналогичные результаты: треть тестов завершилось успешно (4 из 12 тестов), 7 тестов показали, что функция не всегда выводит ожидаемый результат, а в одном тесте вовсе выпало необработанное исключение.

Это приложение, как и прочие, имеет в исходном коде абсолютные ссылки, что приводит к его полной неработоспособности на других машинах.

Приложение имеет те же недостатки, что и предыдущий модуль, так как его код написан на основе кода из предыдущего модуля.

Для модуля поиска недостатка в количестве посадочных мест для проведения занятий произведено ручное тестирование без написания unit тестов.

В результате было выявлено, что приложение не обрабатывает возможность отсутствия файла по указанному абсолютному пути, что приводит к появлению необработанного исключения Runtime.InteropServices.COMException и невозможности дальнейшей работы программы. Кроме того при отсутствии по указанному абсолютному пути текстового файла, в который будет записываться итог работы программы, происходит появление необработанного исключения IO.DirectoryNotFoundExeption (рис. 2.8).

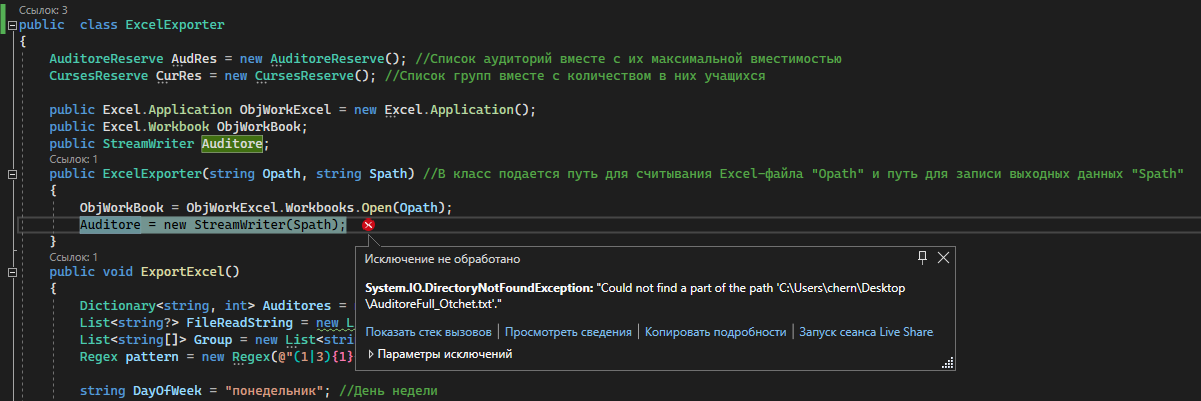


Рисунок 2.8 – Появление исключения IO.DirectoryNotFoundExeption.

Кроме того, при работе программы происходит запись слишком большого числа (оно должно обозначать номер недели) в переменную типа byte, что вызывает необработанное исключение OverflowExeption (рис. 2.9).

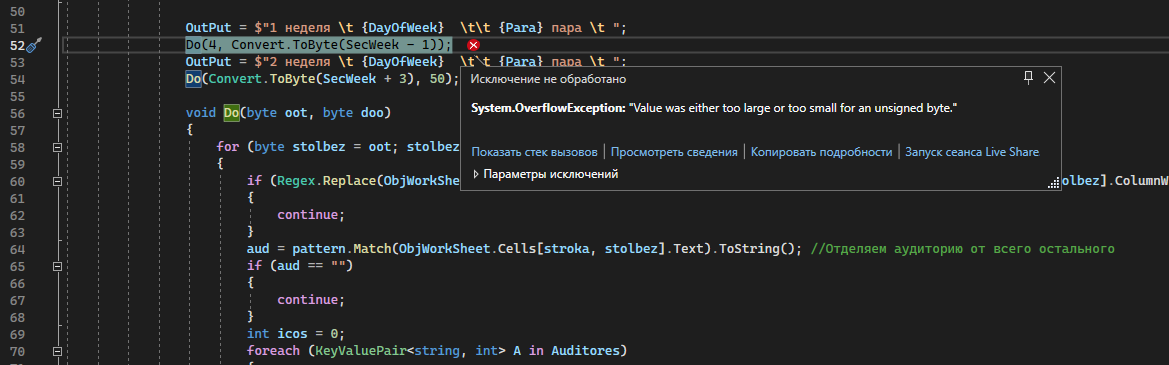


Рисунок 2.9 – Выбрасывание исключения OverflowExeption.

Выпадение любого из перечисленных необработанных исключений приводит к полному прерыванию работы приложения и невозможности дальнейшего его использования.

Дальнейшее тестирование заканчивалось появлением того или иного необработанного исключения, что привело к невозможности полноценного исследования программы и завершения его работы корректным способом.

Постоянные «аборты» программы приводили к утечкам памяти, так как при подобном завершении программы не было реализовано закрытие потока работы с xls-файлом (рис. 2.10).

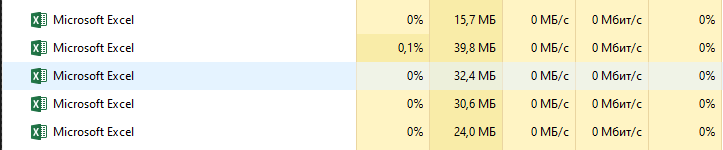


Рисунок 2.10 – Утечка памяти при работе модуля.

Для модуля считывания номеров всех аудиторий и типов занятий из файла расписания было проведено ручное тестирование. По результатам данного тестирования можно сделать вывод, что как и во всех ранее рассмотренных модулях в данном приложении имеются абсолютные пути к файлам, что делает его неработоспособным на других машинах в связи с появлением необработанных исключений. Кроме того, при передаче в программу пустого файла типа xls происходит появление необработанного исключения, прерывающего работу программы (рис. 2.11).

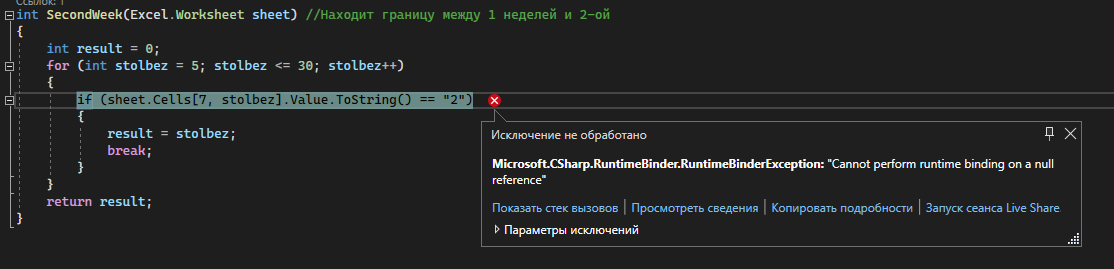


Рисунок 2.11 – Выбрасывание исключения RuntimeBinder.RuntimeBinderExeption.

Появление этого исключения так же приводит к утечке памяти и создания нескольких экземпляров фоновых процессов программы excel.exe.

Для модуля считывания ФИО преподавателей и наименований дисциплин из файла расписания проводилось ручное тестирование.

Большим преимуществом данного решения, по сравнению с другими ранее рассмотренными модулями, является отсутствие абсолютных путей и реализация считывания текущего пути для запускаемой программы. Это делает данный модуль работоспособным на других машинах, однако запись в исходном коде названия считываемого файла делает работу неудобной.

При некорректном указании пути или отсутствии указанного файла расписания в папке с проектом происходит появление необработанного исключения Runtime.InteropServices.COMException. Кроме того, приложение реализовано только частично. Вывод списка предмета не осуществляется, что не соответствует заданию (рис. 2.12).

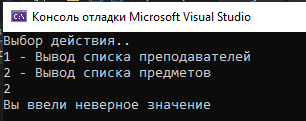


Рисунок 2.12 – Ошибочная работа приложения.

Кроме того, даже после корректного завершения работы программы не происходит закрытие потока работы с excel-файлом, что приводит к утечке памяти.

Для приложений Е, Ж, З, И, Й и К характерна ошибка многих ранее рассмотренных модулей – наличие абсолютных путей, которые вызывают появление необработанного исключения и прекращение работы программы, а также – отсутствие проверки существования указанного файла.

Код модуля последовательного запуска приложений из программного комплекса представляет из себя последовательный вызов всех ранее описанных модулей.

@echo off

echo Start %time%

echo Start Приложение А\ConsoleApp2\bin\Debug\netcoreapp3.1\ConsoleApp2.exe

echo Start Приложение Б\ConsoleApp2\bin\Debug\netcoreapp3.1\ConsoleApp2.exe

echo Start Приложение В\bin\Debug\net6.0\ConsoleApp1.exe

echo Start Приложение Г\bin\Debug\net6.0\ConsoleApp1.exe

echo Start Приложение Д\bin\Debug\net6.0\ConsoleApp1.exe

echo Start Приложение Е\bin\Debug\net6.0\ConsoleApp1.exe

echo Start Приложение Ж\bin\Debug\net6.0\ConsoleApp2.exe

echo Start Приложение З\bin\Debug\net6.0\ConsoleApp1.exe

echo Start Приложение И\bin\Debug\net6.0\ConsoleApp2.exe

echo Start Приложение Й\bin\Debug\net6.0\ConsoleApp1.exe

echo Start Приложение К\bin\Debug\net6.0\ConsoleApp1.exe

echo Stop %time%

Проверка показала, что данный код не работоспособен, так как для многих модулей необходимо дождаться завершения ранее запущенных программ. Важно отметить, что все приложения никак не взаимосвязаны, т.е. даже при условии, что одна программа завершит свою работу и выдаст результат, другая не сможет воспользоваться этим результатом, т.к. пути в 11 из 12 проектов заданы в виде абсолютных ссылок.

По ранее перечисленным причинам невозможно провести интеграционное тестирование, т.к. модули не связаны в один проект и представляют из себя набор автономных проектов.

По результатам модульного тестирования можно сделать вывод, что программный комплекс не способен успешно пройти функциональное тестирование, так как программы лишь частично выполняют заданную работу при наличии большого количества условий. Это приводит к неработоспособности проекта, а следовательно – к неудовлетворению пользовательских потребностей.

В проведении системного и приёмочного тестирования нет необходимости, так как представленный проект неудовлетворительно прошёл проверку на более нижних ступенях пирамиды тестирования.

## Разработка программного продукта

При разработке программного продукта были учтены недочеты рассматриваемого комплекса. В первую очередь было решено отказаться от реализации программного продукта в виде консольного приложения в пользу WPF приложения. Это сделано с целью увеличения круга пользователей, т.к. многие пользователи не имеют опыта работы с консольными приложениями [4].

Вторым важным изменением является введение возможности выбора любого файла формата .xlsx, поскольку это увеличивает удобство пользовательского интерфейса. Кроме того, в представленном программном комплексе пути к файлам были прописаны непосредственно в коде программы, что кроме неудобства увеличивает вероятность отказа.

В анализируемом программном комплексе списки групп, количество студентов в группах, список аудиторий и т.д. представлены в виде данных, внесенных в код программы. Это приводит к тому, что данный программный продукт устаревает в течение года и требует доработки и ежегодной поддержки. Для устранения данного недостатка данные было решено хранить вне программного кода в текстовых файлах программного продукта. Это позволяет пользователю в любое время отредактировать списки так, как необходимо для составления расписания. Для хранения данных были выбраны текстовые файлы, а не база данных, так как по заданию данный программный продукт должен работать «из коробки», без установки большого количества дополнительного ПО. Для работы программного продукта необходимо установить только SDK 6.0.311 и выше, который распространяется на бесплатной основе компанией Microsoft и который может скачать с официального сайта даже неопытный пользователь. SDK необходим исключительно для разработки и сборки программного продукта. Для релизной версии он не требуется, однако, так как программный продукт написан на .Net 6, для ОС Windows 8.1 и старше, возможно, придётся обновить универсальную среду выполнения C (UCRT). Для Windows 10 и Windows 11 эти действия не нужны. Для выполнения обновления необходимо либо зайти в центр обновления Windows и установить все доступные обновления, либо перейти на официальный сайт Microsoft и скачать файл установки обновления оттуда. Установка такого обновления не представляет проблем даже для начинающих пользователей. Так же необходимо установить .NET Runtime 6.0.16, установка которого аналогична установке SDK (NET Runtime 6.0.16 является частью SDK).

При запуске приложения Timetable.exe открывается окно, предлагающее выбрать файл расписания, либо отредактировать текстовые файлы, содержащие группы, ФИО преподавателей, аудитории, типы предметов и наименования предметов (рис.2.13).

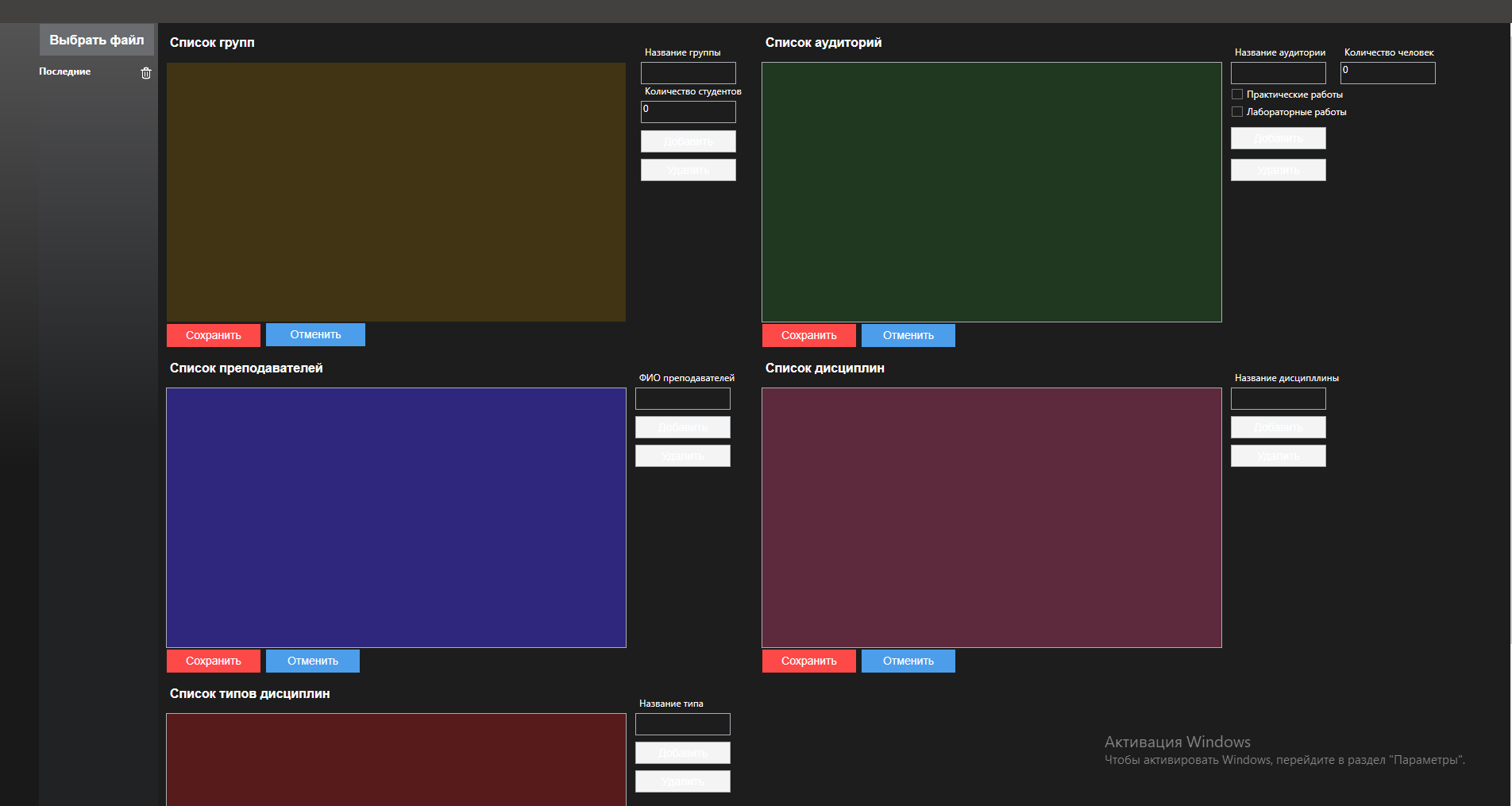


Рисунок 2.13 – Главное окно приложения.

Для работоспособности проекта в корневой папке необходимо наличие файлов:

* EPPlus.dll, EPPlus.Interfaces.dll,
* EPPlus.System.Drawing.dll,
* Microsoft.Extensions.Configuration.Abstractions.dll,
* Microsoft.Extensions.Configuration.dll,
* Microsoft.Extensions.Configuration.FileExtensions.dll,
* Microsoft.Extensions.Configuration.Json.dll,
* Microsoft.Extensions.FileProviders.Abstractions.dll,
* Microsoft.Extensions.FileProviders.Physical.dll,
* Microsoft.Extensions.FileSystemGlobbing.dll,
* Microsoft.Extensions.Primitives.dll,
* Microsoft.IO.RecyclableMemoryStream.dll,
* Newtonsoft.Json.dll,
* System.Security.Cryptography.Pkcs.dll.

Данные файлы являются используемыми библиотеками и поставляются вместе с релизной версией проекта.

Важно упомянуть, что при первом запуске какие-то из упоминаемых текстовых файлов могут не существовать. Кроме того, в процессе использования приложения может произойти удаление файлов. Поэтому в программном продукте реализовано автоматическое создание новых пустых файлов, что повышает отказоустойчивость системы.

На рисунке 2.13 представлен случай, когда программа запустилась впервые, без ранее заданных тестовых файлов.

Приложение подразумевает возможность заполнения данных, их редактирования, добавления и удаления (рис.2.14).

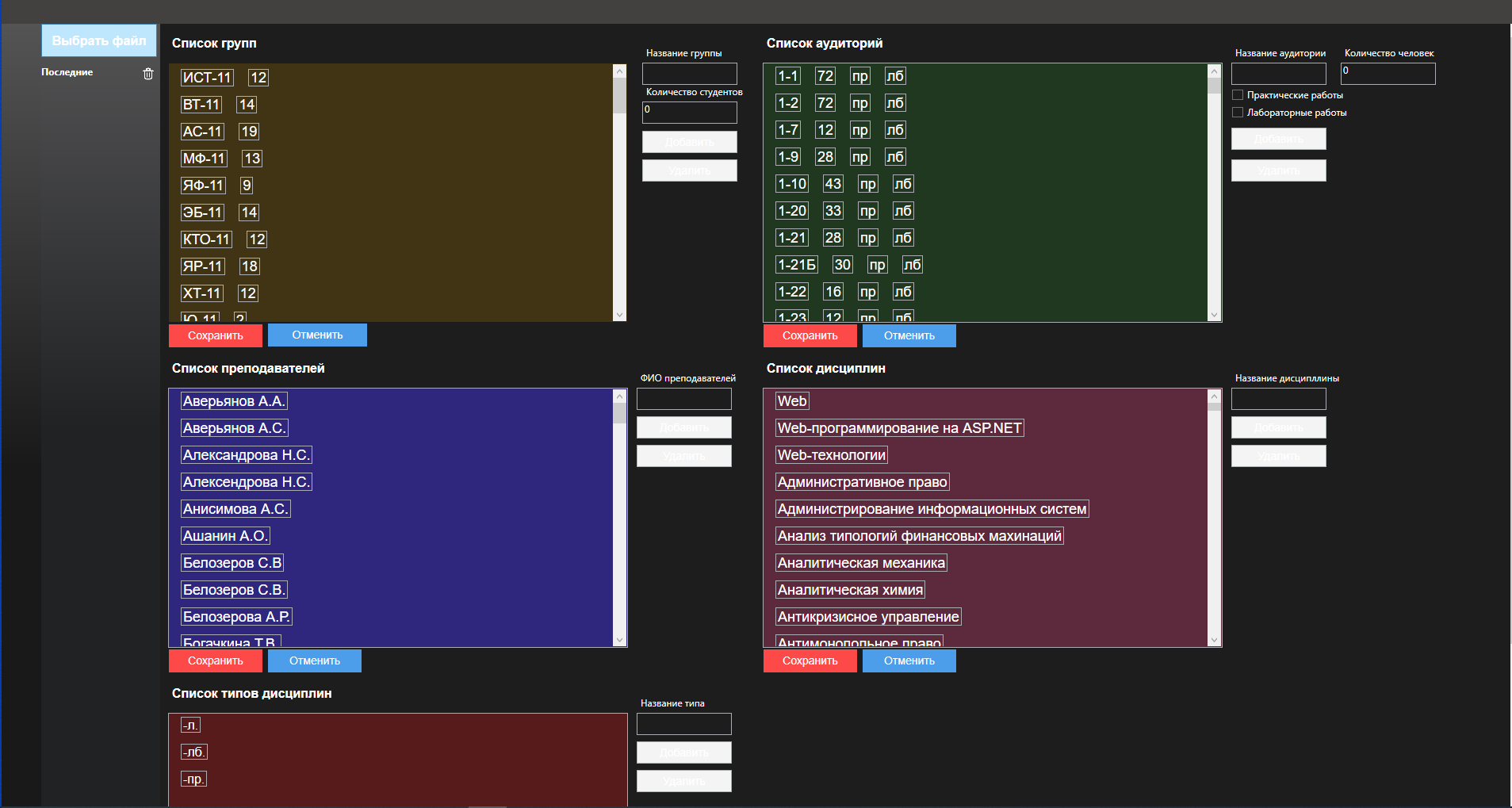


Рисунок 2.14 – Редактирование, добавление, удаление информации в текстовых файлах.

При нажатии на кнопку «Выбрать файл» происходит открытие диалогового окна для выбора файла. Стоит заметить, что программный продукт может работать только с файлами формата .xlsx, поддержка .xls отсутствует, что связано с использованием библиотеки Epplus, которая не поддерживает файлы формата .xls (рис.2.15).

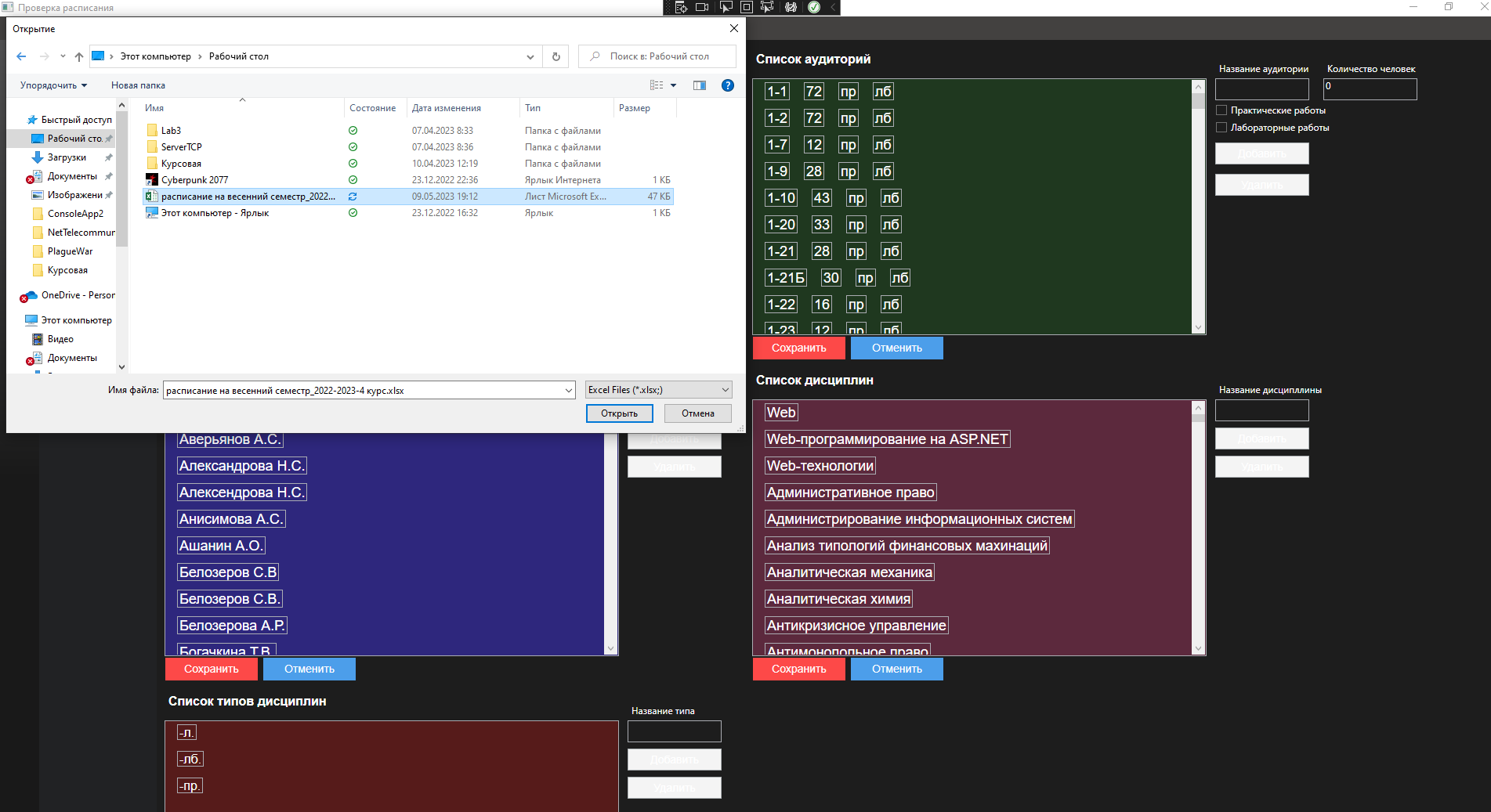


Рисунок 2.15 – Открытие файла расписания.

После выбора файла его название, путь и дата открытия записываются в файл формата .json для обеспечения быстрого открытия и работы с выбранным файлом расписания. При добавлении файла его наименование высвечивается в левом меню. Для работы с файлом необходимо выбрать его кликом левой кнопки мыши. При выборе файла открывается левое меню действий с файлом (рис.16).

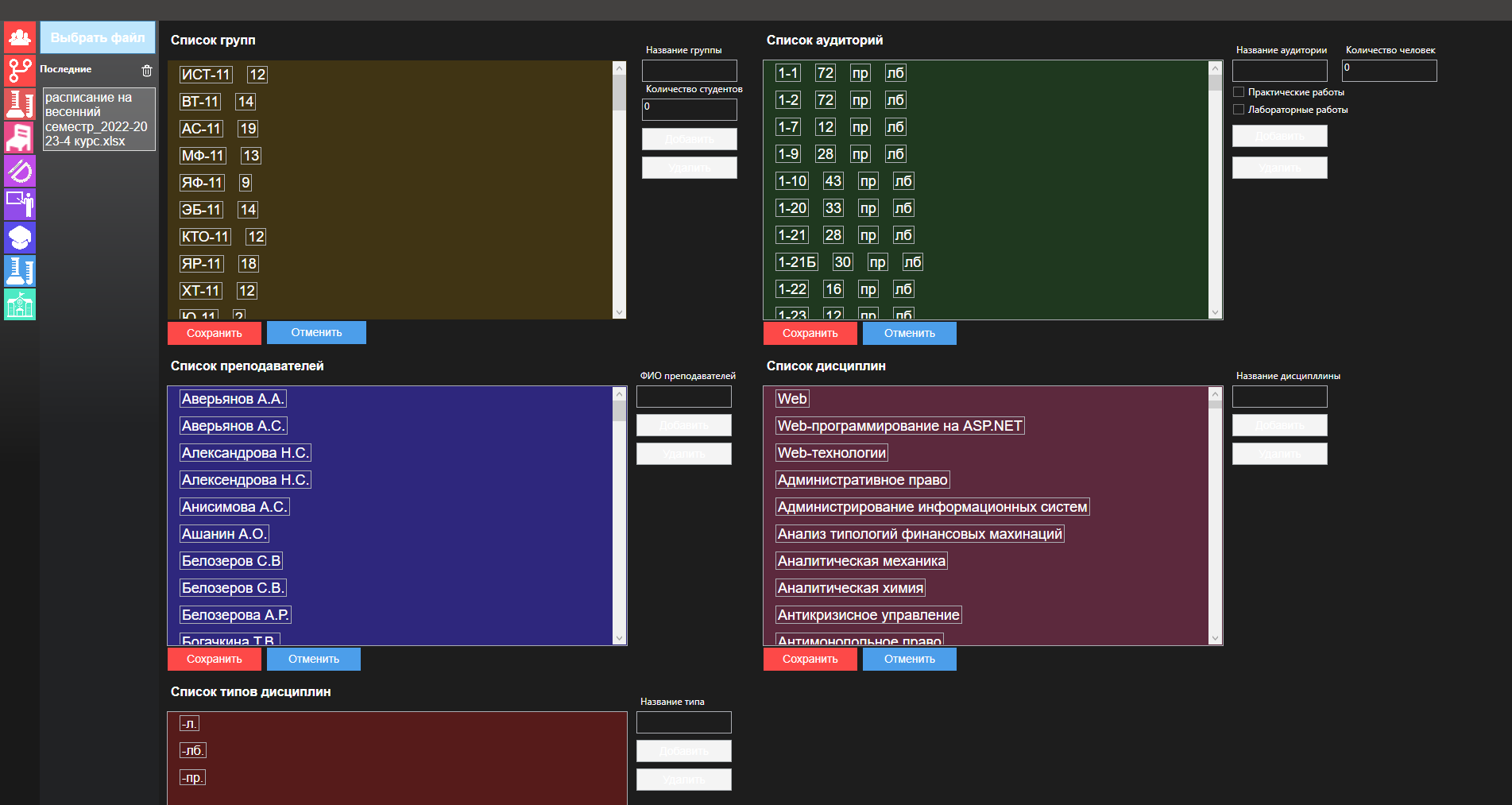


Рисунок 2.16 – Меню действий с файлом расписания.

Первым пунктом меню действий является проверка расписания на проведение двух разных занятий в одной аудитории. Для реализации данного алгоритма создается класс, код которого представлен ниже:

public class ClassroomsOnDoubleLessons

{ public ClassroomsAll Classrooms { get; set; }

public LessonsAll Lessons { get; set; }

public object Row { get; set; }

public object Week { get; set; }

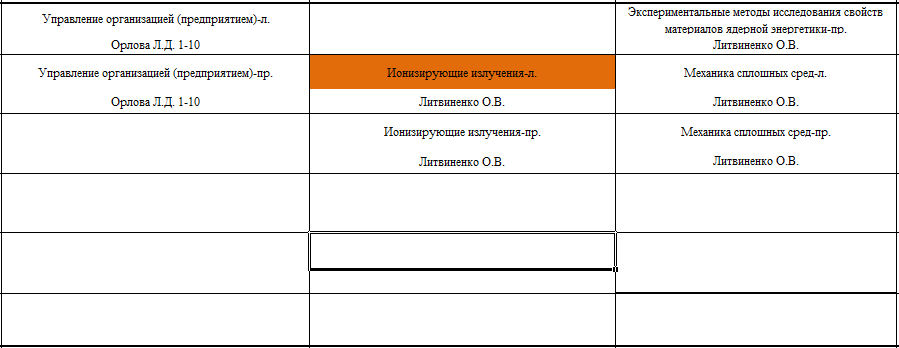
public object Col { get; set; }

public object Page { get; set; }}

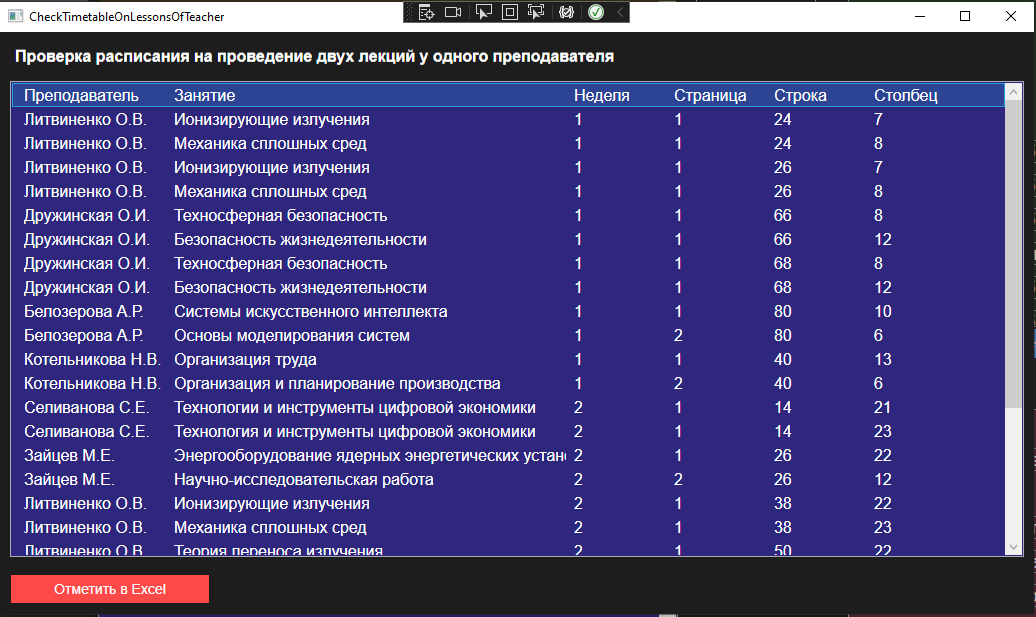
Создается список из объектов данного класса List<T>, программа проходит все расписание по листам в Excel файле, проверяет наличие аудиторий, которые находятся на одной строке в файле расписания (т.е. занятия проводятся в одно время). Если в одной строке есть общая аудитория, и при этом названия предметов не совпадают, то программа считает это за ошибку (рис.2.17).

  
Рисунок 2.17 – Проверка расписания на проведение разных занятий в одной аудитории в одно время.

При выборе некорректной строки и нажатии на кнопку «Отметить в Excel» происходит поиск выбранной информации по используемому файлу расписания. Все данные, соответствующие выбранным, отмечаются красным цветом фона, (рис.2.18).

  
Рисунок 2.18 – Выделение некорректных данных в файле расписания.

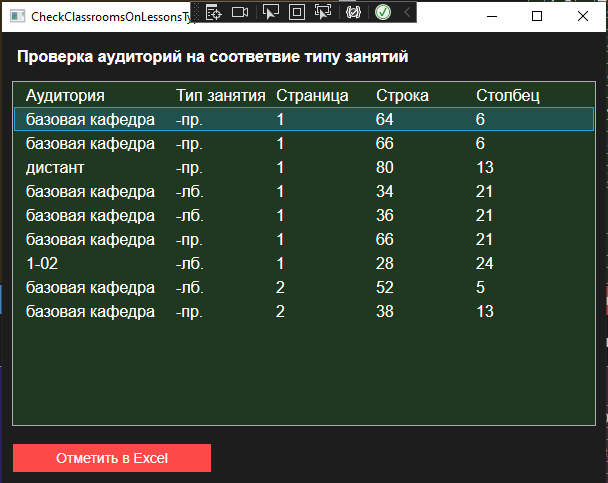
Вторым пунктом меню является проверка расписания на проведение одним преподавателем в одно время разных предметов (рис.2.19). Поиск данных происходит аналогично ранее рассмотренному алгоритму. Выделение данных в файле расписания аналогично выше описанному.

  
Рисунок 2.19 – Поиск в расписании разных занятий, проводимых одним преподавателем в одно время.

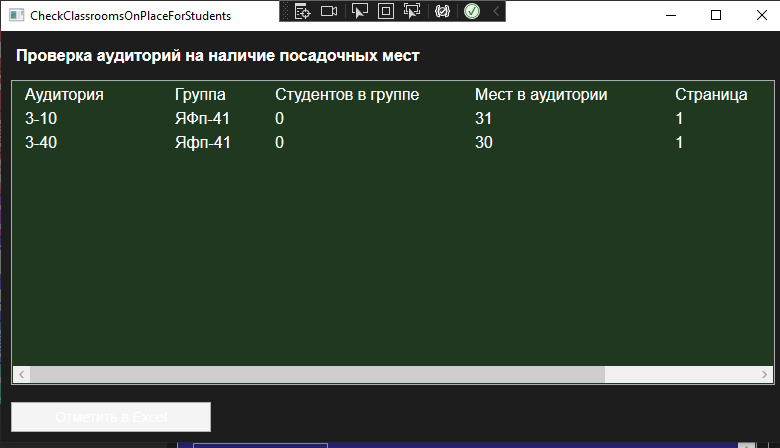
Аналогично реализован третий пункт меню – проверка аудиторий на соответствие аудиторий типу проводимого занятия. В расписании имеется 3 типа занятий:

* .л – Лекция;
* .п – Практическое занятие;
* .лб – Лабораторное занятие.

Для проведения практического и лабораторного занятия зачастую необходимо определенное оборудование. Для кафедры информационных технологий – это компьютеры, для физиков и химиков – реактивы и специальные установки. Проведение многих занятий данного типа недопустимо в неоснащенных аудиториях. На открываемой форме происходит вывод некорректных данных (рис.2.20).

  
Рисунок 2.20 – Проверка соответствия аудиторий типу проводимого занятия.

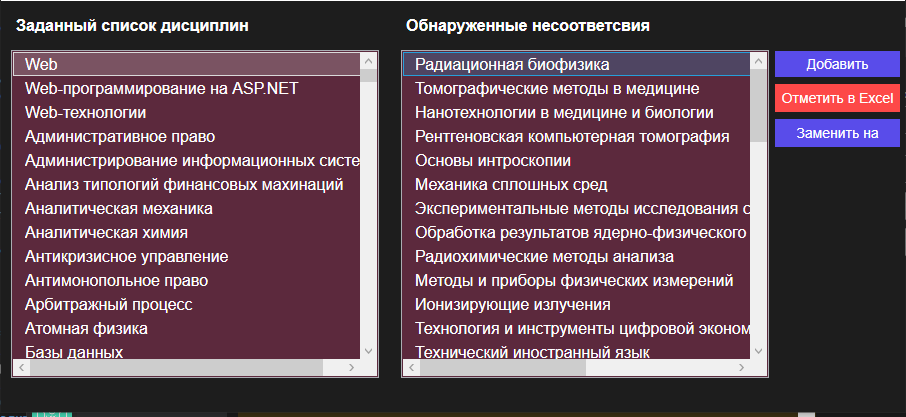
В зависимости от группы, на занятиях может присутствовать различное количество студентов. Некоторые аудитории недостаточно большие, чтобы вместить всех обучающихся. Для более корректного составления расписания четвёртый пункт меню проверяет аудитории на количество посадочных мест. При отсутствии аудитории в списках, недостаточном количестве мест, или если в группе нет студентов, то это считается некорректными данными и выводится в специальном окне проверки, в котором можно отметить некорректные данные и выделить их в файле расписания (рис.2.21).

  
Рисунок 2.21 – Проверка в расписании аудиторий на количество посадочных мест.

Все рассмотренные проверки базируются на данных из упомянутых текстовых файлов, в которых хранятся списки, соответствующие эталонным значениям. Все значения, которые не соответствуют эталонным в предыдущих проверках, тоже выводятся. Например, если при проверке аудиторий на количество посадочных мест встретится аудитория, которая отсутствует в списках, она так же выведется как некорректное значение.

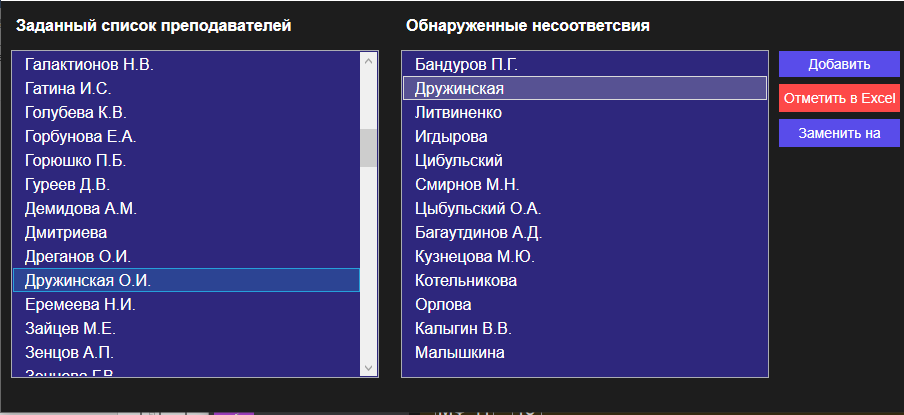
С 5 по 9 пункты меню предназначены для проверки в расписании тех или иных данных на соответствие их эталонным значениям из текстовых файлов. Пункты меню продемонстрированы на рисунке 2.16.

Пункт меню 5 предназначен для проверки файла на соответствие встречающихся в расписании дисциплин с эталонными значениями (рис.2.22).

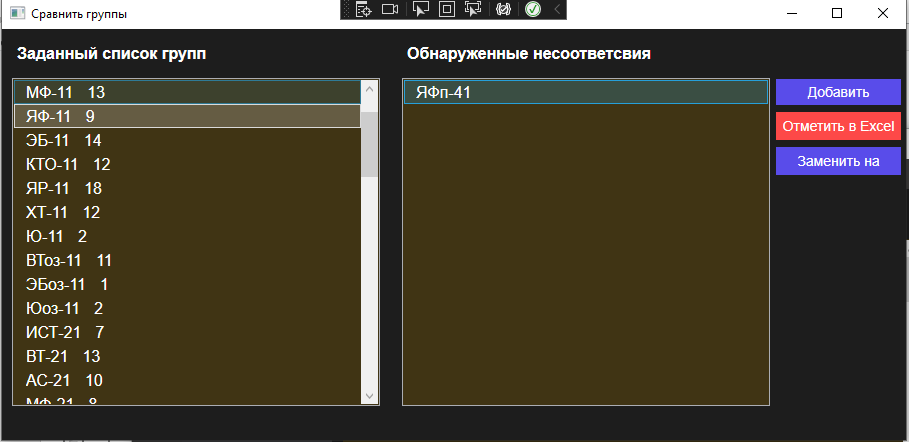
  
Рисунок 2.22 – Проверка дисциплин из файла расписания.

Если наименование дисциплины отсутствует в списке эталонных значений, то данное наименование выводится в список в компоненте ListBox справа. На представленном рисунке выявлено большое количество несоответствий, что связано с неполноценным заполнением списков дисциплин, предоставленных в изучаемом комплексе программ. Для уменьшения количества несовпадений, необходимо дополнить файл. Полученные наименования можно добавить в списки эталонных, отметить в файле расписания, или, выбрав значение из списка слева, заменить его во всём файле расписания. Например, если в расписании встречается дисциплина с наименованием «Веб технологии», а в списках имеется дисциплина с наименованием «Веб-технологии», то можно выделить строки с представленными данными, нажать на кнопку «Заменить на». При этом сработает алгоритм поиска всех ячеек с некорректным наименованием и замены в них имеющегося значения на выбранное корректное наименование из списка.

Аналогично устроена проверка ФИО преподавателей, представленная 6 пунктом в меню. Функционал, внешний вид и способ компоновки аналогичен выше рассмотренному (рис.2.23).

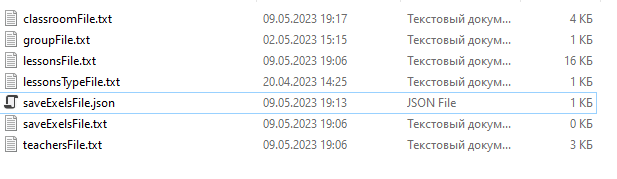
  
Рисунок 2.23 – Проверка ФИО преподавателей.

Аналогично работает проверка групп в расписании, представленная 7 пунктом меню. Однако, для быстродействия было решено искать наименования групп не во всём расписании, а только в его 7-ой строке, т.к. за 5 лет составления расписания наименования групп всегда находились на данной позиции (рис.2.24).

  
Рисунок 2.24 – Проверка наименований групп.

8 и 9 пункты меню – это проверка типов занятий и номеров аудиторий на соответствие эталонным спискам. Интерфейс, возможности и алгоритм поиска схож с ранее рассматриваемыми примерами.

Эталонные значения хранятся в текстовых файлах в папке file программного продукта (рис.2.25).

  
Рисунок 2.25 – Список используемых файлов программного продукта.

Наполнение данных файлов осуществляется определенным образом. Данные разделяются друг от друга знаком | (символ вертикальной черты). Отсутствие ошибок гарантируется тем, что в основном формирование файла происходит через специальные функции, которые производят запись определенным образом. При каждом изменении данных происходит полная перезапись всего файла. Каждое новое значение пишется с новой строки.

Рассмотрим в качестве примера способ записи списка эталонных аудиторий. 1-1|72|1|1 – строка из текстового файла, где 1-1 – наименование аудитории, 72 – количество посадочных мест, а две последние 1 – можно ли проводить лабораторные и практические работы в данной аудитории (0 – нет, 1 – да).

Аналогично происходит запись остальных текстовых файлов, кроме файла формата .json. Он хранит список ранее открытых файлов. Пример кода из файла .json представлен ниже:

{"Date":"2023-04-25T10:07:03.6663555+04:00","Name":"C:\\Users\\ti.salmina\\Desktop\\расписание на весенний семестр\_2022-2023-4 курс.xlsx","Path":"C:\\Users\\ti.salmina\\Desktop\\расписание на весенний семестр\_2022-2023-4 курс.xlsx"}

Для считывания данного файла используется метод десериализации. Ниже представлен фрагмент кода, считывающий данные из файла формата json.

public async Task<List<OldFile>> Read()

{List<OldFile> files = new List<OldFile>();

if (!File.Exists(path)){

File.CreateText(path); }

using (StreamReader fs = new StreamReader(path, true))

{

while (fs.Peek() >= 0)

{

files.Add(JsonConvert.DeserializeObject<OldFile>(fs.ReadLine().ToString()));

}

fs.Close();

}

return files;}

Для записи добавляемых файлов используется метод сериализации. Фрагмент кода добавления новой записи представлен ниже:

public async Task Save(OldFile file)

{

if (!File.Exists(path))

{

File.CreateText(path);

}

// сохранение данных

using (StreamWriter fs = new StreamWriter(path, true))

{

fs.WriteLine(JsonConvert.SerializeObject(file).ToString());

fs.Close();

}

}

При проверке файла расписания в ячейках часто встречается вспомогательная информация (например, время, номер пары, день недели).

Для того чтобы вспомогательные данные не попадали в отчет по некорректным данным, считываемая информация проверяется, в том числе с помощью регулярных выражений.

Регулярные выражения представляют эффективный и гибкий метод по обработке больших текстов, позволяя в то же время существенно уменьшить объемы кода по сравнению с использованием стандартных операций со строками [5].

Основная функциональность регулярных выражений в .NET сосредоточена в пространстве имен System.Text.RegularExpressions. А центральным классом при работе с регулярными выражениями является класс Regex [6].

Для реализации проверки расписания на сопровождающую информацию использовался следующий фрагмент кода:

if (!item.Cells[j+1, i].Value.ToString().ToLower().Contains("понедельник") &&

!item.Cells[j+1, i].Value.ToString().ToLower().Contains("вторник") &&

!item.Cells[j+1, i].Value.ToString().ToLower().Contains("среда") &&

!item.Cells[j+1, i].Value.ToString().ToLower().Contains("четверг") &&

!item.Cells[j+1, i].Value.ToString().ToLower().Contains("пятница") &&

!item.Cells[j+1, i].Value.ToString().ToLower().Contains("суббота") &&

!Regex.IsMatch(item.Cells[j+1, i].Value.ToString(), @"^[0-9]{3,5}.[0-9]{3,5}", RegexOptions.IgnoreCase) &&

item.Cells[j+1, i].Value.ToString() != "1" &&

item.Cells[j+1, i].Value.ToString() != "2" &&

item.Cells[j+1, i].Value.ToString() != "3" &&

item.Cells[j+1, i].Value.ToString() != "4" &&

item.Cells[j+1, i].Value.ToString() != "5" &&

item.Cells[j+1, i].Value.ToString() != "6" &&

item.Cells[j+1, i].Value.ToString() != "7")

Так как данные, проверяемые в расписании, всегда находятся в одних и тех же диапазонах строк (это связано с тем, что расписании имеет зафиксированные размеры в 6-7 пар, 6 дней, т.е. порядка от 72 до 84 строк), то было решено проводить проверку в диапазоне от 8 до 88 строки, т.к. 7 строка – это список групп. Все проверяемые данные находятся друг по отношению к другу через строку, поэтому для проверки данных использовались следующие циклы:

* Для типов занятий и наименования занятий:

for (int j = 8; j < 87; j=j+2)

* Для ФИО преподавателей и аудиторий:

for (int j = 9; j < 88; j = j + 2)

* Для списка групп:

int col = item.Dimension.End.Column;

for (int i = 1; i < col; i++)

## Тестирование программного продукта

Для тестирования программного продукта использовался метод белого ящика. Основным типом тестирования были функциональные тесты. Кроме того, программный продукт проходил тестирование на отказ.

Программа проверялась на ошибки (некорректное использование программного продукта), дефекты (ошибки программистов при разработке) и на сбои.

Тестирование кода было как статическим, так и динамическим.

Для большой части функций были написаны Unit тесты. Для реализации тестов использовалась платформа MSTest, которая поддерживает модульное тестирование в Visual Studio [7].

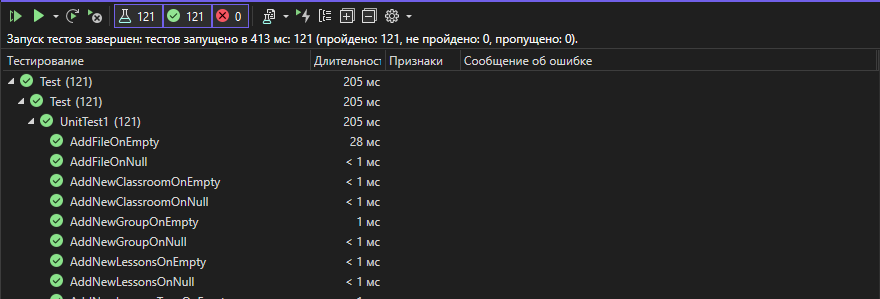
Основными направлениями тестирования функций были проверки на отказ:

* при передаче в качестве аргумента значения null;
* при передаче в качестве одного из ключевых значений объекта значения null;
* при передаче в качестве аргумента значения empty (пустая строка, 0);
* при запуске функций, не принимающих аргументы.

Для проверки работоспособности функций был написан 121 тестовый метод. В полном объёме эти методы выполняются за 205 мс при следующих характеристиках системы:

* Процессор Intel(R) Core(TM) i5-11400 11 поколения;
* 16 Гб оперативной памяти DDR4;
* Диски Samsung SSD 980 и ST1000DM010-2EP102;
* Операционная система Windows 10 Pro, версия 20H2, сборка 19042.685;
* Visual Studio Enterprise 2022 17.2.3.

По результатам такого тестирования можно утверждать, что приложение готово к использованию, т.к. указанные методы тестирования не выявили некорректностей в работе (рис.2.26).

  
Рисунок 2.26 – Результат прохождения Unit тестов программным продуктом.

Важную роль в обеспечении безотказности системы играет проверка существования наличия файлов и директории при работе с текстовыми файлами.

При каждом обращении к любому из файлов происходит сначала проверка на существование указанной директории, а если директория не обнаружена, то создается новая папка. Далее проверяется наличие искомого файла, и если файл не обнаружен, то происходит его генерация. Данные проверки позволяют исключить вероятность возникновения исключения при обращении к файлу или папке, которых не существует. Ниже представлен фрагмент кода, выполняющий данную проверку:

if (!Directory.Exists("file"))

{

Directory.CreateDirectory("file");

}

if (!File.Exists(path))

{

File.CreateText(path);

}

Для проверки работоспособности и соответствия приложения требованиям было проведено функциональное тестирование. Оно проверяет только результат некоторого действия и не проверяет промежуточные состояния системы при выполнении этого действия.

Для каждого возможного действия было проведено функциональное тестирование, которое показало, что продукт соответствует требованиям данной работы.

После выполнения функциональных тестов было выполнено сквозное тестирование по различным сценариям, которые отличаются сложностью и количеством выполняемых действий. Сквозное тестирование показало, что программный продукт соответствует поставленным требованиям.

Стоит отметить, что сквозное тестирование было выполнено вручную, так как в условиях данной работы написание полноценных автотестов было бы нерационально, так как сквозное тестирование, хоть и очень полезно, но ресурсозатратно. Для полноценной реализации автоматических сквозных тестов необходимо большее количество времени, чем выделено в условиях данной работы.

Для написания автоматических тестов интерфейса в будущем планируется использовать одну из данных библиотек: .Net UI Automation, White project, Test Complete, Coded UI test. Использование Coded UI test предпочтительнее, так как данная библиотека поставляется как компонент Visual Studio 2019-2022, что подразумевает более удобное использование и наличие официальной документации.

## Перспективы развития

В дальнейшем планируется реализовать возможность проверки расписания на наличие свободных в данный момент аудиторий, что позволяет в любой день и к началу любой пары иметь список пустующих аудиторий. Это важно, например, при необходимости провести презентацию с помощью проектора.

Кроме того, планируется реализовать полноценную базу данных, которая хранила бы данные, представленные в текстовых файлах. Кроме того, для надежности подключения при отсутствии базы данных программный продукт бы генерировал новую базу данных, заполненную информацией по умолчанию.

Для решения ранее описанной проблемы с необходимостью установки дополнительных программных продуктов планируется реализовать файл программы установщика, которая бы занималась установкой программного продукта и всех необходимых для его работы компонентов.

Следует также рассмотреть возможность реализации логирования. Для этого можно использовать одну из четырех библиотек Log4Net, SeriLog, NLog, Microsoft.Extensions.Logging.

Выводы

Представленный программный комплекс имеет множество дефектов, что связано с недостатком опыта в разработке у студентов. Для комплекса было выполнено функциональное, сквозное, интеграционное тестирование, выявлены недочеты и баги. На основе анализа был разработан новый программный продукт, сочетающий в себе функционал всех программ, представленных в программном комплексе.

Для разработанного программного продукта было произведено функциональное и сквозное тестирование. Тестирование проводилось по методу белого ящика, были написаны unit тесты практически для всех функций проекта.

# ЗАКЛЮЧЕНИЕ

Проблема проверки расписания остро стоит во многих учебных заведениях, так как корректно составленное расписание – залог успешной учебной и рабочей деятельности.

Для решения данной проблемы был изучен ранее созданный комплекс программных продуктов, проанализированы его недочеты и преимущества. На основе анализа был разработан новый программный продукт, сочетающий в себе функционал всех программ, представленных в программном комплексе

Для реализации проекта было решено использовать язык программирования C# на целевой платформе .Net 6.0. Контроль версий реализован с помощью Git. Репозиторий проекта располагается на GitHub.

Для разработанного программного продукта было произведено функциональное и сквозное тестирование. Тестирование проводилось по методу белого ящика, были написаны unit тесты практически для всех функций проекта.

В качестве перспектив развития проекта можно предложить:

* добавление логирования;
* реализация базы данных;
* реализация файла инсталляции программного окружения;
* поиск дополнительных некорректностей в расписании;
* реализация возможностей составления расписания через созданный программный продукт.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Добро пожаловать в интегрированную среду разработки Visual Studio: [Электронный ресурс] // Microsoft. Режим доступа https://learn.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2019 (Дата обращения: 25.02.2023 г.);
2. Язык C# и платформа .NET : [Электронный ресурс] // Metanit. Режим доступа: https://metanit.com/sharp/tutorial/1.1.php (Дата обращения: 25.02.2023 г.);
3. Архитектура .NET [Электронный ресурс] // Microsoft. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/ (Дата обращения: 25.02.2023 г.);
4. Документация по Windows Presentation Foundation : [Электронный ресурс] // Microsoft. Режим доступа <https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/?view=netdesktop-7.0> (Дата обращения: 25.02.2023 г.);
5. Регулярные выражения : [Электронный ресурс] // Metanit. Режим доступа: <https://metanit.com/sharp/tutorial/7.4.php> (Дата обращения: 06.04.2023 г.);
6. Регулярные выражения .NET: [Электронный ресурс] // Microsoft. Режим доступа <https://learn.microsoft.com/ru-ru/dotnet/standard/base-types/regular-expressions> (Дата обращения: 06.04.2023 г.);
7. Использование закодированных тестов пользовательского интерфейса для проверки кода : [Электронный ресурс] // Microsoft. Режим доступа <https://learn.microsoft.com/ru-ru/visualstudio/test/use-ui-automation-to-test-your-code?view=vs-2022> (Дата обращения: 08.05.2023 г.);

# Приложение А Unit Тест Модуля поиска в расписании занятий у разных преподавателей, проводимых одновременно в одной аудитории

namespace TestProject1

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestSpaceEqual()

{

//arage

string str = "Обычная строка";

string expected = "Обычная строка";

//act

string test= ConsoleApp2.Program.Space(str);

//assert

Assert.AreEqual(expected, test);

}

[TestMethod]

public void TestSpaceTwoSpace\_expected\_empty()

{

//arage

string str = " ";

string expected = "";

//act

string test = ConsoleApp2.Program.Space(str);

//assert

Assert.AreEqual(expected, test);

}

[TestMethod]

public void TestSpaceOneSpace\_expected\_empty()

{

//arage

string str = " ";

string expected = "";

//act

string test = ConsoleApp2.Program.Space(str);

//assert

Assert.AreEqual(expected, test);

}

[TestMethod]

public void TestSpaceThreeSpace\_expected\_empty()

{

//arage

string str = " ";

string expected = "";

//act

string test = ConsoleApp2.Program.Space(str);

//assert

Assert.AreEqual(expected, test);

}

[TestMethod]

public void TestSpaceEmptySpace\_expected\_empty()

{

//arage

string str = "";

string expected = "";

//act

string test = ConsoleApp2.Program.Space(str);

//assert

Assert.AreEqual(expected, test);

}

[TestMethod]

public void TestSpaceNull\_expected\_null()

{

//arage

string str = null;

string expected = null;

//act

string test = ConsoleApp2.Program.Space(str);

//assert

Assert.AreEqual(expected, test);

}

[TestMethod]

public void TestSpaceWhithNewStr\_expected\_null()

{

//arage

string str = "\n";

string expected = "";

//act

string test = ConsoleApp2.Program.Space(str);

//assert

Assert.AreEqual(expected, test);

}

[TestMethod]

public void TestSpaceWhithNewStrAndData\_expected\_null()

{

//arage

string str = "Тест\n Текст";

string expected = "Тест\n Текст";

//act

string test = ConsoleApp2.Program.Space(str);

//assert

Assert.AreEqual(expected, test);

}

[TestMethod]

public void TestSpaceTab\_expected\_null()

{

//arage

string str = " ";

string expected = "";

//act

string test = ConsoleApp2.Program.Space(str);

//assert

Assert.AreEqual(expected, test);

}

[TestMethod]

public void TestSpaceTabData\_expected\_Data()

{

//arage

string str = "Test test";

string expected = "Test test";

//act

string test = ConsoleApp2.Program.Space(str);

//assert

Assert.AreEqual(expected, test);

}

[TestMethod]

public void TestSpaceTabDataEnd\_expected\_Data()

{

//arage

string str = "Test ";

string expected = "Test";

//act

string test = ConsoleApp2.Program.Space(str);

//assert

Assert.AreEqual(expected, test);

}

[TestMethod]

public void TestSpaceTabDataStart\_expected\_Data()

{

//arage

string str = " Test";

string expected = "Test";

//act

string test = ConsoleApp2.Program.Space(str);

//assert

Assert.AreEqual(expected, test);

}

}

}