ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ЦИФРОВЫХ ОБРАЗОВАТЕЛЬНЫХ ТЕХНОЛОГИЙ

ПИ (филиал) ДГТУ в г. Таганроге
ЦМК «Прикладная информатика»

Практикум
по дисциплине

«Осуществление интеграции программных модулей»

Автор
Михайлович Е.В.

Ростов-на-Дону, 2026

Аннотация

Практикум по дисциплине «Осуществление интеграции программных модулей» предназначен для студентов очной формы обучения по специальности 09.02.11 «Разработка и управление программным обеспечением».

Автор

канд. тех. наук, преподаватель ЦМК «Прикладная информатика»
Михайлович Е.В.

Оглавление

Введение 4
Практическая работа № 1 «Разработка структуры проекта» 6
Практическая работа № 2 «Разработка модульной структуры проекта (диаграммы модулей)» 16
Практическая работа № 3 «Разработка перечня артефактов и протоколов проекта» 25
Практическая работа № 4 «настройка работы системы контроля версий (типов импортируемых файлов, путей, фильтров и др. Параметров импорта в репозиторий)» 39
Практическая работа № 5 «Разработка и интеграция модулей проекта (командная работа)» 50
Практическая работа № 6 «Отладка отдельных модулей программного проекта» 66
Практическая работа № 7 «Организация обработки исключений» 97
Практическая работа № 8 «Применение отладочных классов в проекте»108
Практическая работа № 9 «отладка проекта» 114
Практическая работа № 10 «Инспекция кода модулей проекта» 135
Практическая работа № 11 «Тестирование интерфейса пользователя средствами инструментальной среды разработки» 159
Практическая работа № 12 «Разработка тестовых модулей проекта для тестирования отдельных модулей» 170
Практическая работа № 13 «Выполнение функционального тестирования» 182
Практическая работа № 14 «Тестирование интеграции» 192
Практическая работа № 15 «Документирование результатов тестирования» 204
Перечень использованных информационных ресурсов 219

Введение

Учебно-методические указания к практическим занятиям по дисциплине «Осуществление интеграции программных модулей» предназначены для оказания методической помощи обучающимся при выполнении практических работ, направленных на закрепление теоретических знаний и формирование навыков решения задач. В указаниях представлены краткие теоретические сведения по основным темам курса, примеры решения типовых задач, а также задания, подлежащие выполнению на практических занятиях.

Обучающийся должен знать**:** основные принципы проектирования модулей программного обеспечения; языки программирования и технологии для реализации модулей; паттерны проектирования и структуры данных для создания эффективных и масштабируемых модулей; методы анализа требований и способов определения функциональности модуля; принципы создания интерфейсов для взаимодействия с другими модулями и системами; принципы обеспечения безопасности, производительности и масштабируемости при проектировании модулей; методы анализа и оптимизации проектируемых модулей для повышения их эффективности и качества.

Обучающийся должен уметь: проектировать модули, соответствующие бизнес-задачам; создавать архитектурные диаграммы и документацию; определять структуру и интерфейсы модулей; анализировать требования к модулю и определять его функциональность; проектировать архитектуру модуля, включая выбор подходящих паттернов проектирования и структуры данных; создавать диаграммы классов, последовательностей и прочих диаграмм для визуализации проектируемого модуля; выбирать подходящие языки программирования и технологии для реализации модуля; проектировать интерфейсы программного обеспечения для взаимодействия с другими модулями и системами.

Логика построения практического занятия строится на следующей последовательности:

1.Изучение теоретического материала

2.Выполнение практического задания

3.Оформление отчета в соответствии с требованиями к его формированию

4.Ответы на контрольные вопросы

При выполнении практической работы обязательно сохраняйте все скриншоты и промежуточные результаты для оформления отчета. Все команды Git рекомендуется выполнять как через интерфейс Visual Studio, так и через командную строку для лучшего понимания работы системы контроля версий.

Отчет должен содержать:

1.Титульный лист с указанием:

– Названия учебного заведения
– Темы практической работы
– Номера варианта
– ФИО студента и группы
– ФИО преподавателя

2.Цель работы (задачи и методы)

3.Задание

И далее в зависимости от целей работы определяется содержательная часть результатов выполнения, указанная в каждой работе

4. Выводы по работе: какие навыки были получены; какие принципы организации структуры проекта были применены; какие сложности возникли при выполнении работы.

Практическая работа № 1 «Разработка структуры проекта»

Цель работы: Изучить принципы организации структуры программного проекта, освоить методы создания модульной архитектуры приложения на языке C# в среде Visual Studio 2022, научиться применять лучшие практики структурирования кода для обеспечения масштабируемости и поддерживаемости программного обеспечения.

Теоретическая часть

Понятие структуры проекта. Структура проекта — это организация исходного кода, ресурсов, конфигурационных файлов и других компонентов программного обеспечения в логически связанные модули и каталоги. Правильная структура проекта обеспечивает:

- удобство навигации по коду;
- упрощение поддержки и развития системы;
- возможность параллельной работы нескольких разработчиков;
- упрощение тестирования и интеграции модулей;
- повышение переиспользуемости кода.

Принципы организации структуры проекта

1. Модульность Разделение функциональности на независимые модули, каждый из которых отвечает за определенную задачу.

2. Слабая связанность (Low Coupling) Минимизация зависимостей между модулями для обеспечения независимости их разработки и тестирования.

3. Высокая связность (High Cohesion) Группировка тесно связанных элементов внутри одного модуля.

4. Разделение ответственности (Separation of Concerns) Каждый модуль должен решать только одну задачу или группу тесно связанных задач.

5. *Принцип единственной ответственности (Single Responsibility Principle*) Каждый класс или модуль должен иметь только одну причину для изменения.

Архитектурные шаблоны организации проекта

Трехслойная архитектура (Three-tier Architecture)

Наиболее распространенный подход к организации бизнес-

приложений:

Слой представления (Presentation Layer)

- отвечает за взаимодействие с пользователем;
- содержит формы, элементы управления, логику отображения данных;
- в C# реализуется через Windows Forms, WPF, ASP.NET.

Слой бизнес-логики (Business Logic Layer)

- содержит основную логику приложения;
- реализует бизнес-правила и алгоритмы обработки данных;
- не зависит от способа представления и хранения данных.

Слой доступа к данным (Data Access Layer)

- отвечает за взаимодействие с источниками данных;
- содержит код работы с базами данных, файлами, внешними сервисами;
- изолирует детали хранения данных от остальных частей приложения.

Архитектура MVC (Model-View-Controller)

- **Model (Модель)** — данные и бизнес-логика;
- **View (Представление)** — интерфейс пользователя;
- **Controller (Контроллер)** — обработка пользовательского ввода и управление взаимодействием между моделью и представлением.

Архитектура MVVM (Model-View-ViewModel)

Используется в WPF и других XAML-технологиях:

- **Model** — данные и бизнес-логика;
- **View** — пользовательский интерфейс;
- **ViewModel** — посредник между View и Model, содержит логику представления.

Типичная структура проекта на C#

```
SolutionName/
│
├── SolutionName.PresentationLayer/      # Слой представления
│   ├── Forms/                           # Формы приложения
│   ├── UserControls/                    # Пользовательские элементы управления
│   ├── Resources/                       # Ресурсы (изображения, иконки)
│   └── Program.cs                       # Точка входа в приложение
```

```
├── SolutionName.BusinessLogicLayer/     # Слой бизнес-логики
│   ├── Services/                 # Бизнес-сервисы
│   ├── Validators/               # Валидаторы данных
│   ├── Interfaces/               # Интерфейсы бизнес-логики
│   └── Exceptions/               # Пользовательские исключения
│
├── SolutionName.DataAccessLayer/       # Слой доступа к данным
│   ├── Repositories/             # Репозитории для работы с данными
│   ├── Context/                  # Контекст базы данных
│   ├── Migrations/               # Миграции базы данных
│   └── Interfaces/               # Интерфейсы доступа к данным
│
├── SolutionName.Domain/          # Доменная модель
│   ├── Entities/                 # Сущности предметной области
│   ├── Enums/                    # Перечисления
│   └── ValueObjects/             # Объекты-значения
│
├── SolutionName.Common/          # Общие компоненты
│   ├── Helpers/                  # Вспомогательные классы
│   ├── Extensions/               # Методы расширения
│   ├── Constants/                # Константы
│   └── Utilities/                # Утилиты
│
└── SolutionName.Tests/           # Модульные тесты
    ├── UnitTests/                # Юнит-тесты
    └── IntegrationTests/         # Интеграционные тесты
```

Проекты и решения в Visual Studio

Solution (Решение) — контейнер для одного или нескольких связанных проектов. Файл решения имеет расширение .sln.

Project (Проект) — отдельная сборка (assembly), которая компилируется в исполняемый файл (.exe) или библиотеку (.dll). Файл проекта имеет расширение .csproj.

Типы проектов C# в Visual Studio

Class Library (.NET) — библиотека классов, компилируется в DLL.

Console Application — консольное приложение.

Windows Forms App — приложение с графическим интерфейсом на базе Windows Forms.

WPF Application — приложение на базе Windows Presentation Foundation.

ASP.NET Web Application — веб-приложение.

Unit Test Project — проект для модульного тестирования.
Пространства имен (Namespaces). Пространства имен используются для логической группировки классов и предотвращения конфликтов имен**:**

```
namespace LibrarySystem.Domain.Entities
{
   public class Book
   {
      public int Id { get; set; }
      public string Title { get; set; }
      public string Author { get; set; }
   }
}
```

Соглашения по именованию пространств имен:

- используется нотация PascalCase;
- структура: <Company-Name>.<ProductName>.<LayerName>.<ComponentName>;
- пример: MyCompa-ny.LibrarySystem.DataAccess.Repositories.

Зависимости между проектами. При создании многослойной архитектуры важно правильно настроить зависимости:

Допустимые зависимости:

- Presentation Layer → Business Logic Layer
- Presentation Layer → Domain
- Business Logic Layer → Data Access Layer
- Business Logic Layer → Domain
- Data Access Layer → Domain

Недопустимые зависимости:

- Data Access Layer → Business Logic Layer
- Data Access Layer → Presentation Layer
- Domain → любой другой слой

Управление зависимостями через NuGet

NuGet — менеджер пакетов для платформы .NET, позволяющий добавлять в проект внешние библиотеки. Основные способы добавления пакетов:

– через графический интерфейс Visual Studio (Manage NuGet Packages);

– через консоль диспетчера пакетов (Package Manager Console);

– через редактирование файла .csproj.

Конфигурационные файлы

app.config / web.config — XML-файлы конфигурации для .NET Framework.

appsettings.json — JSON-файл конфигурации для .NET Core и .NET 5+.

Пример

```
appsettings.json:
{
  "ConnectionStrings": {
    "DefaultConnection": "Serv-
er=localhost;Database=LibraryDB;Trusted_Connection=True;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information"
    }
  }
}
```

Система контроля версий и структура проекта. При работе с Git важно правильно настроить файл. gitignore, чтобы исключить из репозитория:

- временные файлы (bin/, obj/);
- пользовательские настройки (.vs/, *.user);
- зависимости, управляемые NuGet (packages/, кроме packages.config);
- файлы сборки (*.dll, *.exe из рабочих каталогов).

Visual Studio автоматически создает подходящий. gitignore при инициализации репозитория.

Документирование структуры проекта. Рекомендуется создавать файл README.md в корне решения с описанием:

- назначения проекта;
- структуры решения и назначения каждого проекта;
- используемых технологий и библиотек;
- инструкций по сборке и запуску;
- правил работы с кодом.

Рабочее задание: Разработайте структуру проекта для программной системы, техническое задание на которую вы создали в практической работе № 1. Создайте решение (solution) с необходимыми проектами, организуйте модульную структуру в соответствии с принципами многослойной архитектуры.

Вариант 1. Система учета книг в библиотеке. Разработать программную систему для автоматизации учета книжного фонда библиотеки учебного заведения. Система должна обеспечивать регистрацию книг, учет выдачи и возврата литературы студентам, формирование отчетов о наличии и движении книг.

Вариант 2. Система расписания занятий. Разработать программную систему для автоматизации составления и управления расписанием занятий в колледже. Система должна учитывать доступность аудиторий, нагрузку преподавателей, распределение групп, формировать расписание в различных форматах.

Вариант 3. Система учета успеваемости студентов. Разработать программную систему для автоматизации учета успеваемости студентов. Система должна обеспечивать ввод оценок, расчет среднего балла, формирование ведомостей, отчетов по успеваемости групп и отдельных студентов.

Вариант 4. Система управления складом. Разработать программную систему для автоматизации учета товаров на складе небольшого предприятия. Система должна обеспечивать регистрацию поступления и отпуска товаров, контроль остатков, формирование отчетов о движении товаров.

Вариант 5. Система регистрации заявок в службе технической поддержки

Разработать программную систему для регистрации и обработки заявок пользователей в службе технической поддержки. Система должна обеспечивать создание заявок, назначение исполнителей, отслеживание статуса выполнения, формирование отчетов.

Вариант 6. Система учета посещаемости занятий. Разработать программную систему для автоматизации учета посещаемости студентов. Система должна обеспечивать ежедневную регистрацию посещаемости, расчет процента посещений, формирование отчетов по группам и дисциплинам.

Вариант 7. Система бронирования переговорных комнат. Разработать программную систему для управления бронированием переговорных комнат и конференц-залов в офисе. Система должна отображать доступность помещений, обеспечивать резервирование на определенное время, уведомлять пользователей.

Вариант 8. Система учета рабочего времени сотрудников. Разработать программную систему для учета рабочего времени сотрудников организации. Система должна регистрировать время прихода и ухода, учитывать опоздания, рассчитывать отработанные часы, формировать табели учета рабочего времени.

Вариант 9. Система управления проектами. Разработать программную систему для управления небольшими IT-проектами. Система должна обеспечивать создание проектов, постановку задач, назначение исполнителей, отслеживание прогресса, формирование отчетов о ходе выполнения проектов.

Вариант 10. Система учета оборудования. Разработать программную систему для учета компьютерной техники и оборудования в учебном заведении. Система должна вести реестр оборудования, отслеживать его размещение, регистрировать ремонты и обслуживание, формировать инвентаризационные отчеты.

Требования к выполнению задания

1. Проектирование структуры. Разработайте схему структуры проекта, включающую:

- перечень проектов (projects), входящих в решение;
- назначение каждого проекта;
- основные пространства имен и папки внутри каждого проекта;
- зависимости между проектами (в виде диаграммы или списка).

2. Создание решения в Visual Studio 2022. Создайте новое решение (Solution) с именем, соответствующим вашей предметной области (например, LibraryManagementSystem).
Добавьте в решение следующие проекты:
Проект слоя представления:

- тип проекта: Windows Forms App или WPF Application;
- имя: [ИмяСистемы].PresentationLayer или [ИмяСистемы].UI.

Проект слоя бизнес-логики:

- тип проекта: Class Library;
- имя: [ИмяСистемы].BusinessLogicLayer или [ИмяСистемы].BLL.

Проект слоя доступа к данным:

- тип проекта: Class Library;
- имя: [ИмяСистемы].DataAccessLayer или [ИмяСистемы].DAL.

Проект доменной модели:

- тип проекта: Class Library;
- имя: [ИмяСистемы].Domain или [ИмяСистемы].Models.

Проект общих компонентов (опционально):

- тип проекта: Class Library;
- имя: [ИмяСистемы].Common.

3. Создание структуры каталогов

В каждом проекте создайте необходимые папки (folders) для организации кода:

В проекте Domain:

- Entities — классы сущностей предметной области (не менее 3 классов);
- Enums — перечисления;
- Interfaces — интерфейсы доменной модели.

В проекте DataAccessLayer:

- Repositories — классы репозиториев;
- Interfaces — интерфейсы репозиториев;
- Context — контекст данных (если используется БД).

В проекте BusinessLogicLayer:

- Services — бизнес-сервисы;
- Interfaces — интерфейсы сервисов;
- Validators — классы валидации данных;
- Exceptions — пользовательские исключения.

В проекте PresentationLayer:

- Forms — формы приложения (для Windows Forms);
- Views — представления (для WPF);

- Controllers или ViewModels — контроллеры или модели представления.

4. Создание классов доменной модели. В проекте Domain создайте не менее 3 классов сущностей, соответствующих вашей предметной области, с необходимыми свойствами (properties). Пример для системы учета книг:

```
namespace LibrarySystem.Domain.Entities
{
    public class Book
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Author { get; set; }
        public string ISBN { get; set; }
        public int PublicationYear { get; set; }
        public bool IsAvailable { get; set; }
    }
}
```

5. Настройка зависимостей между проектами. Настройте ссылки (references) между проектами в соответствии с архитектурой:

- PresentationLayer ссылается на BusinessLogicLayer и Domain;
- BusinessLogicLayer ссылается на DataAccessLayer и Domain;
- DataAccessLayer ссылается на Domain;
- Domain не имеет ссылок на другие проекты решения.

6. Создание интерфейсов. Создайте интерфейсы для основных компонентов системы:

- интерфейс репозитория в DataAccessLayer (например, IBookRepository);
- интерфейс бизнес-сервиса в BusinessLogicLayer (например, IBookService).

Пример интерфейса репозитория:

```
namespace LibrarySystem.DataAccessLayer.Interfaces
{
    public interface IBookRepository
    {
        List<Book> GetAll();
        Book GetById(int id);
        void Add(Book book);
        void Update(Book book);
        void Delete(int id);
```

```
    }
}
```

7. Документирование структуры. Создайте файл README.md в корневой папке решения, содержащий:

- название и краткое описание системы;
- описание структуры проекта и назначение каждого проекта;
- диаграмму зависимостей между проектами;
- используемые технологии;
- инструкции по запуску проекта.

8. Инициализация системы контроля версий. Инициализируйте Git-репозиторий для вашего проекта через Visual Studio или командную строку. Убедитесь, что создан файл .gitignore для Visual Studio.

9. Создание первого коммита. Выполните первый коммит с сообщением "Initial project structure" или "Создание структуры проекта".

Требования к отчету. Отчет должен содержать:

1. Описание разработанной структуры проекта: схема структуры решения; описание назначения каждого проекта; диаграмма зависимостей между проектами.

2. Скриншоты: скриншот структуры решения в Solution Explorer; скриншот созданной структуры папок в каждом проекте; скриншот настроенных зависимостей (References) для каждого проекта; скриншот созданных классов сущностей с кодом; скриншот созданных интерфейсов с кодом.

3. Листинги кода: код классов доменной модели (не менее 3 классов); код интерфейсов (не менее 2 интерфейсов).

4. Содержимое файла README.md

5. Описание организации контроля версий: скриншот инициализированного Git-репозитория; скриншот первого коммита.

6. Выводы по работе

Контрольные вопросы

1.Что такое структура проекта и почему правильная организация структуры важна для разработки программного обеспечения?

2.Опишите трехслойную архитектуру приложения. Какие

функции выполняет каждый слой? Почему важно разделение ответственности между слоями?

3.В чем различие между Solution (решением) и Project (проектом) в Visual Studio? Для чего используется решение, включающее несколько проектов?

4.Какие типы проектов C# существуют в Visual Studio? Когда используется Class Library, а когда Console Application, или Windows Forms App?

5.Что такое пространства имен (namespaces) в C#? Как они связаны со структурой проекта? Приведите примеры правильного именования пространств имен.

6.Опишите правила настройки зависимостей между проектами в многослойной архитектуре. Какие зависимости являются допустимыми, а какие нарушают принципы архитектуры?

7.Что такое NuGet и для чего он используется? Как добавить внешнюю библиотеку в проект через NuGet?

8.Зачем нужен файл gitignore в проекте Visual Studio? Какие файлы и папки следует исключать из системы контроля версий?

9.Что такое принцип единственной ответственности (Single Responsibility Principle)? Как он применяется при организации структуры проекта?

10.Какие преимущества дает использование интерфейсов при проектировании архитектуры приложения? Приведите примеры использования интерфейсов в разных слоях приложения.

Время выполнения: 4 часа

Практическая работа № 2 «Разработка модульной структуры проекта (диаграммы модулей)»

Цель работы: Освоить принципы декомпозиции программного обеспечения на модули, изучить методы проектирования модульной архитектуры, научиться разрабатывать и оформлять диаграммы модулей с использованием языка C# и среды разработки Visual Studio 2022.

Теоретическая часть

Понятие модуля и модульной структуры. Модуль — это функционально законченный фрагмент программы, оформленный в виде отдельного файла или со- вокупности файлов с исход-

ным кодом, предназначенный для использования в других программах.

Модульное программирование — это организация программы как совокупности небольших независимых блоков (модулей), структура и поведение которых подчиняются определенным правилам.

Основные принципы модульного проектирования

1.Принцип единственной ответственности (Single Responsibility Principle) — модуль должен иметь только одну причину для изменения, т.е. решать одну конкретную задачу.

2.Принцип слабой связанности (Low Coupling) — модули должны быть минимально зависимы друг от друга.

3.Принцип сильной связности (High Cohesion) — элементы внутри модуля должны быть логически связаны между собой.

4.Принцип инкапсуляции — модуль должен скрывать детали своей внутренней реализации.

5.Принцип повторного использования — модуль должен быть спроектирован так, чтобы его можно было использовать в различных частях системы или в других проектах.

Типы модулей в C#. В языке C# и среде Visual Studio модульная структура может быть реализована на разных уровнях:

1. Пространства имен (Namespaces) Логическая группировка связанных классов и типов.

namespace LibrarySystem.Data

```
{
    // классы для работы с данными
}
```

2. Классы (Classes) Основная единица модульности в объектно-ориентированном программировании.

public class BookRepository

```
{
    // методы работы с книгами
}
```

3. Сборки (Assemblies) Физические единицы развертывания (.dll или .exe файлы), содержащие скомпилированный код.

4. Проекты (Projects) Организационная единица в Visual Studio, компилируемая в отдельную сборку.

Виды модулей по функционально назначению

Модули представления (Presentation Layer) Отвечают за пользовательский интерфейс и взаимодействие с пользователем.
Модули бизнес-логики (Business Logic Layer) Содержат основную логику приложения, правила обработки данных.
Модули доступа к данным (Data Access Layer) Обеспечивают взаимодействие с источниками данных (базы данных, файлы, внешние сервисы).
Модули общего назначения (Common/Utilities) Содержат вспомогательные функции, используемые в различных частях приложения.
Модули моделей данных (Models/Entities) Описывают структуру данных предметной области.
Диаграммы модулей. Диаграммы модулей используются для визуализации структуры программной системы и взаимосвязей между ее компонентами. Основные типы диаграмм:
1. Диаграмма компонентов UML (Component Diagram) Показывает физическую структуру системы, компоненты и их зависимости.
2. Диаграмма пакетов UML (Package Diagram) Отображает организацию системы в виде пакетов и зависимости между ними.
3. Диаграмма классов UML (Class Diagram) Детализирует структуру модулей на уровне классов, их атрибутов и методов.
4. Блок-схема модулей Упрощенное представление модульной структуры с указанием основных связей.
Элементы диаграммы модулей
Модуль (компонент) — изображается прямоугольником с именем модуля и его назначением.
Зависимость — показывается стрелкой от зависимого модуля к модулю, от которого он зависит.
Интерфейс — определяет контракт взаимодействия между модулями.
Слой — группировка модулей по функциональному признаку.
Архитектурные шаблоны модульной организации
Трехслойная архитектура (Three-tier Architecture)

- Слой представления (Presentation Layer)
- Слой бизнес-логики (Business Logic Layer)

- Слой данных (Data Access Layer)

Архитектура MVC (Model-View-Controller)

- Model — модели данных
- View — представления
- Controller — контроллеры

Чистая архитектура (Clean Architecture)

- Entities (сущности)
- Use Cases (бизнес-логика)
- Interface Adapters (адаптеры)
- Frameworks & Drivers (внешние компоненты)

Организация модульной структуры в Visual Studio 2022

Solution (Решение) — контейнер верхнего уровня, содержащий один или несколько проектов.

Project (Проект) — набор файлов исходного кода, компилируемый в одну сборку.

Folder (Папка) — логическая группировка файлов внутри проекта.

File (Файл) — отдельный файл с исходным кодом (.cs).

Правила именования модулей

- Имена должны быть понятными и отражать назначение модуля
- Использовать PascalCase для пространств имен и классов
- Избегать сокращений (кроме общепринятых)
- Группировать связанные модули в пространства имен
- Использовать префиксы для обозначения слоев (например, Data, Business, UI)

Документирование модульной структуры. При проектировании модульной структуры необходимо документировать:

- назначение каждого модуля
- интерфейсы взаимодействия между модулями
- зависимости между модулями
- используемые паттерны проектирования
- правила размещения кода в модулях

Рабочее задание: Выполните разработку модульной структуры проекта в соответствии с вариантом задания. Создайте диаграмму модулей и реализуйте базовую структуру проекта в Visual Studio 2022.

Вариант 1. Система учета книг в библиотеке. Разработать модульную структуру для системы учета книжного фонда библиотеки. Система должна обеспечивать управление каталогом книг, читателями, выдачей и возвратом литературы. Требуемые модули:

- Модуль управления книгами (добавление, редактирование, удаление, поиск)
- Модуль управления читателями (регистрация, учет читателей)
- Модуль выдачи/возврата книг
- Модуль формирования отчетов
- Модуль работы с данными (репозиторий)
- Модуль моделей предметной области

Вариант 2. Система расписания занятий. Разработать модульную структуру для системы управления расписанием занятий в образовательном учреждении. Требуемые модули:

- Модуль управления преподавателями
- Модуль управления группами студентов
- Модуль управления аудиториями
- Модуль управления дисциплинами
- Модуль составления расписания
- Модуль проверки конфликтов расписания
- Модуль формирования и экспорта расписания
- Модуль работы с данными
- Модуль моделей

Вариант 3. Система учета успеваемости студентов. Разработать модульную структуру для системы учета успеваемости студентов. Требуемые модули:

- Модуль управления студентами
- Модуль управления дисциплинами
- Модуль учета оценок
- Модуль расчета рейтингов и средних баллов
- Модуль формирования ведомостей
- Модуль аналитики успеваемости
- Модуль работы с данными
- Модуль моделей

Вариант 4. Система управления складом. Разработать мо-

дульную структуру для системы автоматизации складского учета. Требуемые модули:

- Модуль управления номенклатурой товаров
- Модуль учета поступления товаров
- Модуль учета отпуска товаров
- Модуль управления остатками
- Модуль инвентаризации
- Модуль формирования отчетов о движении товаров
- Модуль работы с данными
- Модуль моделей

Вариант 5. Система регистрации заявок в службе технической поддержки. Разработать модульную структуру для системы обработки заявок службы технической поддержки**.**
Требуемые модули:

- Модуль управления пользователями системы
- Модуль создания и регистрации заявок
- Модуль управления статусами заявок
- Модуль назначения исполнителей
- Модуль приоритизации заявок
- Модуль уведомлений
- Модуль формирования отчетов
- Модуль работы с данными
- Модуль моделей

Вариант 6. Система учета посещаемости занятий. Разработать модульную структуру для системы учета посещаемости студентов. Требуемые модули:

- Модуль управления студентами и группами
- Модуль управления расписанием занятий
- Модуль регистрации посещаемости
- Модуль учета пропусков (по уважительной и неуважительной причинам)
- Модуль расчета статистики посещаемости
- Модуль формирования отчетов
- Модуль работы с данными
- Модуль моделей

Вариант 7. Система бронирования переговорных комнат. Разработать модульную структуру для системы бронирования

переговорных комнат и конференц-залов. Требуемые модули:

- Модуль управления помещениями (переговорные, залы)
- Модуль управления пользователями
- Модуль создания бронирований
- Модуль проверки доступности помещений
- Модуль управления расписанием бронирований
- Модуль уведомлений пользователей
- Модуль формирования отчетов о занятости помещений
- Модуль работы с данными
- Модуль моделей

Вариант 8. Система учета рабочего времени сотрудников. Разработать модульную структуру для системы учета рабочего времени. Требуемые модули:

- Модуль управления сотрудниками
- Модуль регистрации прихода/ухода
- Модуль учета рабочих часов
- Модуль учета опозданий и отсутствий
- Модуль расчета переработок
- Модуль формирования табелей учета рабочего времени
- Модуль формирования отчетов
- Модуль работы с данными
- Модуль моделей

Вариант 9. Система управления проектами. Разработать модульную структуру для системы управления IT-проектами. Требуемые модули:

- Модуль управления проектами
- Модуль управления задачами
- Модуль управления участниками проектов
- Модуль назначения задач исполнителям
- Модуль отслеживания прогресса выполнения
- Модуль управления сроками и дедлайнами
- Модуль формирования отчетов о статусе проектов
- Модуль работы с данными
- Модуль моделей

Вариант 10. Система учета оборудования. Разработать модульную структуру для системы учета компьютерной техники и оборудования. Требуемые модули:

- Модуль управления номенклатурой оборудования
- Модуль учета размещения оборудования
- Модуль регистрации перемещений оборудования
- Модуль учета ремонтов и обслуживания
- Модуль инвентаризации
- Модуль формирования инвентаризационных отчетов
- Модуль работы с данными
- Модуль моделей

Требования к выполнению задания

Часть 1. Проектирование модульной структуры

1.Проанализируйте предметную область в соответствии с вариантом задания: определите основные функциональные области системы; выявите группы связанных функций; определите общие компоненты

2.Спроектируйте модульную структуру: разработайте трехслойную архитектуру (представление, бизнес-логика, данные); определите состав модулей для каждого слоя; опишите назначение каждого модуля (2-3 предложения); определите интерфейсы взаимодействия между модулями; определите зависимости между модулями

3.Разработайте диаграмму модулей, которая должна включать: все основные модули системы, сгруппированные по слоям; связи и зависимости между модулями (стрелками); краткое описание назначения каждого модуля; легенду диаграммы (обозначения). Диаграмма может быть выполнена:

– в виде диаграммы компонентов UML (рекомендуется использовать PlantUML, Draw.io, Lucidchart, StarUML)

– в виде блок-схемы модулей (MS Visio, Draw.io, PowerPoint)

Часть 2. Реализация структуры в Visual Studio 2022

4.Создайте Solution в Visual Studio 2022:

– создайте новое решение (Solution) с именем, соответствующим варианту

– создайте проекты для разных слоев архитектуры:

– Проект типа Class Library (.NET) для слоя моделей (например, LibrarySystem.Models)

– Проект типа Class Library (.NET) для слоя доступа к данным (например, LibrarySystem.Data)

– Проект типа Class Library (.NET) для слоя бизнес-логики (например, LibrarySystem.Business)
– Проект типа Console App или Windows Forms App для слоя представления (например, LibrarySystem.UI)

5.Организуйте структуру каждого проекта: создайте папки для группировки связанных классов; создайте файлы классов в соответствии с диаграммой модулей; настройте пространства имен в соответствии со структурой проекта; установите зависимости между проектами (References)

6.Создайте заготовки классов: для каждого модуля создайте соответствующий класс; добавьте XML-комментарии с описанием назначения класса; определите основные публичные методы (пока без реализации, можно использовать заглушки); добавьте комментарии к методам, описывающие их назначение

Часть 3. Документирование

7.Создайте текстовое описание модульной структуры, которое должно включать: общую архитектуру системы; описание каждого слоя; таблицу модулей с описанием:

Модуль	Слой	Назначение	Основные классы	Зависимости
...	...	...	...	...

8.Сделайте скриншоты: структура Solution в Visual Studio (обозреватель решений); примеры кода заготовок классов (2-3 класса)

Требования к оформлению отчета Отчет должен содержать:

1. **Проектирование модульной структуры:**
   - описание архитектурного подхода
   - обоснование выбора модулей
   - диаграмма модулей
   - таблица модулей с описаниями
2. **Реализация в Visual Studio 2022:**
   - описание структуры Solution
   - скриншот обозревателя решений
   - скриншоты фрагментов кода (2-3 примера классов с комментариями)
   - описание установленных зависимостей между

проектами

3. **Листинги кода** (примеры заготовок классов для основных модулей, 3-5 классов)
4. **Выводы по работе**

Контрольные вопросы

1.Что такое модуль в программировании? Какие преимущества дает модульная организация программного кода?

2.Перечислите основные принципы модульного проектирования. В чем заключается принцип единственной ответственности?

3.Что такое связанность (coupling) и связность (cohesion) модулей? Почему важно стремиться к слабой связанности и сильной связности?

4.Опишите трехслойную архитектуру приложения. Какие задачи решает каждый из слоев?

5.Какие средства модульной организации кода предоставляет язык C# и платформа .NET?

6.В чем различие между проектом (Project) и сборкой (Assembly) в Visual Studio?

7.Что такое диаграмма компонентов UML? Какие элементы она включает?

8.Как в Visual Studio настраиваются зависимости между проектами? Почему важно правильно организовать направление зависимостей?

9.Что такое пространство имен (namespace) в C#? Как оно связано с модульной структурой проекта?

10.Какие архитектурные шаблоны (паттерны) используются при организации модульной структуры приложений? Опишите один из них.

Время выполнения: 4 ч

Практическая работа № 3 «Разработка перечня артефактов и протоколов проекта»

Цель работы: Изучить состав проектной документации про-

граммного проекта, освоить принципы формирования перечня артефактов и протоколов интеграции программных модулей, получить практические навыки разработки и структурирования проектных артефактов для обеспечения эффективной интеграции компонентов программного обеспечения.

Теоретическая часть

Понятие артефакта проекта. Артефакт проекта — это любой измеримый результат работы, создаваемый в процессе разработки программного обеспечения. Артефакты документируют процесс разработки, обеспечивают коммуникацию между участниками команды и служат основой для контроля качества и интеграции программных модулей. Артефакты включают в себя:

- документацию (технические задания, спецификации, руководства)
- исходный код программных модулей
- конфигурационные файлы
- схемы баз данных
- результаты тестирования
- журналы изменений
- протоколы интеграции

Классификация артефактов проекта

1. Управленческие артефакты:

- план проекта
- график работ (диаграмма Гантта)
- протоколы совещаний
- отчеты о статусе проекта
- реестр рисков
- журнал изменений требований

2. Требования и проектирование:

- техническое задание
- спецификация требований (SRS)
- архитектурная документация
- диаграммы UML (Use Case, классов, последовательностей, компонентов)
- описание интерфейсов
- модели данных (ER-диаграммы)

3. Артефакты разработки:

- исходный код модулей
- конфигурационные файлы проекта
- скрипты сборки (build scripts)
- файлы зависимостей (packages.config, .csproj для C#)
- README файлы
- комментарии и документация кода

4. Артефакты тестирования:

- план тестирования
- тест-кейсы
- результаты тестирования
- отчеты о дефектах
- протоколы интеграционного тестирования
- метрики покрытия кода

5. Артефакты развертывания:

- инструкции по установке
- руководство администратора
- руководство пользователя
- release notes (примечания к выпуску)
- конфигурационные файлы окружения

Протоколы интеграции программных модулей. Протокол интеграции — это набор правил и соглашений, определяющих способ взаимодействия программных модулей между собой. Протоколы обеспечивают согласованную работу компонентов системы и определяют формат обмена данными.

Основные типы протоколов интеграции:

1. Протоколы уровня приложения:

– **HTTP/HTTPS** — для веб-сервисов и REST API

– **SOAP** — для веб-сервисов с формальными контрактами

– **gRPC** — для высокопроизводительного взаимодействия микросервисов

– **WebSocket** — для двусторонней связи в реальном времени

2. Протоколы обмена сообщениями:

– **AMQP** (Advanced Message Queuing Protocol) — для очередей сообщений

– **MQTT** — для легковесного обмена сообщениями (IoT)

– **STOMP** — простой текстовый протокол обмена сообщениями

3. Протоколы вызова удаленных процедур:

– **RPC** (Remote Procedure Call)
– **COM/DCOM** — для интеграции компонентов Windows
– **.NET Remoting** — для взаимодействия .NET приложений

4. Протоколы доступа к данным:

- **ODBC/JDBC** — для доступа к базам данных
- **OData** — для REST-подобного доступа к данным
- **GraphQL** — для гибких запросов данных

Стандарты форматирования сообщений. При интеграции модулей используются стандартизированные форматы данных:

1. XML (eXtensible Markup Language):

- расширяемый язык разметки
- используется в SOAP, конфигурационных файлах
- поддерживает схемы валидации (XSD)

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
  <header>
    <timestamp>2024-01-15T10:30:00</timestamp>
    <sender>ModuleA</sender>
  </header>
  <body>
    <data>Информация для передачи</data>
  </body>
</message>
```

2. JSON (JavaScript Object Notation):

- легковесный формат обмена данными
- используется в REST API, конфигурационных файлах
- удобен для чтения человеком и парсинга

```
{
 "header": {
  "timestamp": "2024-01-15T10:30:00",
  "sender": "ModuleA"
 },
 "body": {
  "data": "Информация для передачи"
 }
}
```

3. Protocol Buffers (Protobuf):

- бинарный формат сериализации от Google
- используется в gRPC
- обеспечивает высокую производительность

4. YAML (YAML Ain't Markup Language):

- человекочитаемый формат для конфигураций
- используется в CI/CD конфигурациях, Docker Compose

Структура перечня артефактов проекта. Перечень артефактов должен содержать:

1. Идентификационные данные: наименование артефакта; уникальный идентификатор/код; версия

2. Описательные данные: назначение артефакта; связь с другими артефактами; зависимости

3. Атрибуты управления: ответственный исполнитель; дата создания/обновления; статус (планируется, в разработке, завершен, утвержден); формат хранения; место хранения (путь в репозитории)

4. Метаданные интеграции: используемые протоколы; форматы данных; интерфейсы взаимодействия; требования к версионированию

Организация артефактов в системе контроля версий. При использовании Git (наиболее распространенная система контроля версий) структура проекта может выглядеть следующим образом:

```
ProjectName/
├── docs/                # Документация
│   ├── architecture/    # Архитектурная документация
│   ├── api/             # Описание API
│   ├── requirements/    # Требования
│   └── protocols/       # Протоколы интеграции
├── src/                 # Исходный код
│   ├── Module1/
│   ├── Module2/
│   └── Integration/     # Код интеграционных компонентов
├── tests/               # Тесты
│   ├── unit/
│   ├── integration/
│   └── e2e/
├── config/              # Конфигурационные файлы
├── scripts/             # Скрипты сборки и развертывания
└── README.md            # Описание проекта
```

```
├── CHANGELOG.md          # Журнал изменений
└── .gitignore            # Игнорируемые файлы
```

Документирование протоколов интеграции в C#/.NET проектах

При разработке на C# важно документировать:

1. Интерфейсы взаимодействия:

```
/// <summary>
/// Интерфейс для интеграции с модулем обработки платежей
/// Протокол: REST API over HTTPS
/// Формат данных: JSON
/// </summary>
public interface IPaymentService
{
    /// <summary>
    /// Инициирует платежную транзакцию
    /// </summary>
    /// <param name="request">Данные платежа в формате PaymentRe-
quest</param>
    /// <returns>Результат операции</returns>
    Task<PaymentResponse> ProcessPaymentAsync(PaymentRequest request);
}
```

2. Модели данных (DTO - Data Transfer Objects):

```
/// <summary>
/// Модель запроса платежа для интеграции между модулями
/// Формат сериализации: JSON
/// </summary>
public class PaymentRequest
{
    public string OrderId { get; set; }
    public decimal Amount { get; set; }
    public string Currency { get; set; }
    public DateTime Timestamp { get; set; }
}
```

3. Конфигурационные файлы (appsettings.json):

```
{
  "IntegrationSettings": {
    "PaymentService": {
      "BaseUrl": "https://api.payment.service",
      "Timeout": 30,
      "Protocol": "REST",
      "DataFormat": "JSON"
    }
  }
}
```

Шаблон описания протокола интеграции. Каждый протокол интеграции должен быть задокументирован по следующему шаблону:

1. Общая информация: Название протокола; Версия; Назначение; Участвующие модули

2. Технические характеристики:

- Транспортный протокол (HTTP, TCP, AMQP и т.д.)
- Формат сообщений (JSON, XML, Protobuf)
- Метод аутентификации
- Шифрование

3. Описание интерфейса: Endpoints/методы; Структура запросов; Структура ответов; Коды ошибок

4. Примеры использования: Примеры запросов; Примеры ответов; Обработка ошибок

5. Требования к реализации: Обработка таймаутов; Повторные попытки; Логирование; Мониторинг

Рабочее задание: Выполните разработку перечня артефактов и протоколов интеграции для программного проекта в соответствии с вариантом задания.

Вариант 1. Интернет-магазин электронных товаров. Разработать перечень артефактов и протоколов интеграции для системы интернет-магазина, состоящей из следующих модулей:

- модуль каталога товаров
- модуль корзины покупок
- модуль обработки заказов
- модуль платежной системы
- модуль управления складом
- модуль уведомлений (email, SMS)

Вариант 2. Система дистанционного обучения. Разработать перечень артефактов и протоколов интеграции для образовательной платформы, состоящей из модулей:

- модуль управления курсами
- модуль тестирования и оценки знаний
- модуль видеоконференций
- модуль форума и обсуждений
- модуль отслеживания прогресса студентов

- модуль выдачи сертификатов

Вариант 3. Банковская система управления счетами. Разработать перечень артефактов и протоколов интеграции для банковской системы, включающей:

- модуль управления счетами клиентов
- модуль проведения транзакций
- модуль кредитования
- модуль отчетности и аналитики
- модуль интеграции с платежными системами
- модуль безопасности и аудита

Вариант 4. Система управления медицинской клиникой. Разработать перечень артефактов и протоколов интеграции для медицинской информационной системы:

- модуль регистрации пациентов
- модуль электронных медицинских карт
- модуль расписания приемов
- модуль лабораторных исследований
- модуль аптечного склада
- модуль биллинга и страхования

Вариант 5. Логистическая система доставки. Разработать перечень артефактов и протоколов интеграции для системы управления доставкой:

- модуль приема заказов
- модуль маршрутизации
- модуль отслеживания транспорта (GPS)
- модуль управления курьерами
- модуль складского учета
- модуль уведомлений клиентов

Вариант 6. Система управления гостиницей. Разработать перечень артефактов и протоколов интеграции для гостиничной системы:

- модуль бронирования номеров
- модуль регистрации гостей (check-in/check-out)
- модуль управления номерным фондом
- модуль дополнительных услуг
- модуль интеграции с системами бронирования (Booking.com и др.)

- модуль биллинга

Вариант 7. Корпоративная система управления проектами. Разработать перечень артефактов и протоколов интеграции для системы управления IT-проектами:

- модуль управления задачами
- модуль учета времени
- модуль управления ресурсами
- модуль документооборота
- модуль интеграции с системой контроля версий (Git)
- модуль отчетности и дашбордов

Вариант 8. Система автоматизации ресторана. Разработать перечень артефактов и протоколов интеграции для ресторанной системы:

- модуль управления меню
- модуль приема заказов
- модуль кухни (Kitchen Display System)
- модуль кассы и оплаты
- модуль управления столиками
- модуль складского учета продуктов

Вариант 9. Платформа для управления фитнес-клубом. Разработать перечень артефактов и протоколов интеграции для системы фитнес-клуба:

- модуль управления абонементами
- модуль расписания тренировок
- модуль управления тренерами
- модуль контроля доступа (турникеты)
- модуль персональных тренировок
- модуль учета посещаемости

Вариант 10. Система умного дома (Smart Home). Разработать перечень артефактов и протоколов интеграции для системы управления умным домом:

- модуль управления освещением
- модуль климат-контроля
- модуль безопасности (камеры, сигнализация)
- модуль управления энергопотреблением
- модуль интеграции с голосовыми помощниками
- модуль сценариев автоматизации

Требования к выполнению задания

1. Проведите анализ архитектуры системы: определите состав модулей системы (не менее 4-6 модулей); опишите назначение каждого модуля; определите связи между модулями; выявите точки интеграции

2. Разработайте перечень артефактов проекта, включающий:

2.1. Управленческие артефакты (не менее 4): план-график разработки модулей; матрица ответственности; журнал рисков интеграции; протоколы совещаний команды

2.2. Артефакты требований и проектирования (не менее 5): техническое задание; архитектурная диаграмма системы (диаграмма компонентов); спецификации интерфейсов модулей; модель данных; диаграммы взаимодействия (sequence diagram)

2.3. Артефакты разработки (не менее 6): структура solution в Visual Studio (.sln); состав проектов (.csproj); конфигурационные файлы (appsettings.json); файлы зависимостей (NuGet packages); стандарты кодирования; шаблоны кода (code templates)

2.4. Артефакты интеграции и тестирования (не менее 5): план интеграционного тестирования; сценарии интеграционных тестов; протоколы тестирования интерфейсов; отчеты о результатах интеграции; матрица трассируемости требований

2.5. Артефакты документирования (не менее 3): руководство разработчика; описание API; инструкция по развертыванию

3. Разработайте описание протоколов интеграции (не менее 3 протоколов). Для каждого протокола укажите:

- **Название и версия протокола**
- **Интегрируемые модули** (источник → приемник)
- **Транспортный протокол** (HTTP/HTTPS, TCP, AMQP и т.д.)
- **Формат данных** (JSON, XML, Binary)
  - **Метод взаимодействия:** синхронный/асинхронный; запрос-ответ / публикация-подписка / очередь сообщений
  - **Структура сообщений:** формат запроса (с примером); формат ответа (с примером); коды ошибок
  - **Требования безопасности:** аутентификация (API ключ, OAuth, JWT); шифрование
  - **Требования к производительности:** максимальное время

отклика; пропускная способность
- **Обработка ошибок:** стратегия повторных попыток; логирование

4. Создайте структуру репозитория проекта. Опишите организацию файлов и папок в системе контроля версий Git:
- корневая структура проекта
- организация документации
- размещение исходного кода модулей
- размещение конфигурационных файлов
- файл .gitignore для проекта C#
- файл README.md с описанием проекта

5. Разработайте матрицу артефактов: Создайте таблицу, содержащую:

№	Наименование артефакта	Тип	Формат	Ответственный	Статус	Местоположение	Связанные артефакты
1	...	...	...	...	...	...	...

6. Оформите спецификацию одного протокола интеграции в виде отдельного документа: Выберите один из протоколов и подробно опишите его с примерами кода на C#:
- интерфейс взаимодействия
- модели данных (DTO классы)
- пример реализации клиента
- пример реализации сервера/обработчика
- конфигурационные параметры

Требования к оформлению отчета. Отчет должен содержать:

1. Результаты анализа архитектуры системы: описание модулей; диаграмма компонентов (можно нарисовать от руки или использовать инструменты); схема взаимодействия модулей
2. Перечень артефактов проекта (с группировкой по категориям)
3. Матрица артефактов (таблица)
4. Описание протоколов интеграции (не менее 3)
5. Подробная спецификация одного протокола с примерами кода на C#
6. Структура репозитория проекта (текстовое описание

или древовидная схема)

7. Выводы по работе: какие артефакты являются критичными для интеграции; какие протоколы наиболее подходят для данной системы; как организация артефактов влияет на эффективность разработки

Пример выполнения (фрагмент)

Система: Интернет-магазин электронных товаров

Модули системы:

1. Модуль каталога товаров (CatalogService)
2. Модуль корзины покупок (CartService)
3. Модуль обработки заказов (OrderService)
4. Модуль платежей (PaymentService)
5. Модуль управления складом (WarehouseService)
6. Модуль уведомлений (NotificationService)

Протокол интеграции № 1: CartService → OrderService

Название: Протокол создания заказа из корзины

Версия: 1.0

Транспорт: HTTP/HTTPS (REST API)

Формат: JSON

Метод: Синхронный запрос-ответ

Структура запроса:

```
POST /api/orders/create
Content-Type: application/json
Authorization: Bearer {jwt_token}

{
 "cartId": "cart-12345",
 "userId": "user-67890",
 "deliveryAddress": {
   "street": "ул. Ленина, д. 10",
   "city": "Москва",
   "postalCode": "101000"
 },
 "paymentMethod": "card"
}
```

Структура ответа (успех):

```
HTTP/1.1 201 Created
Content-Type: application/json

{
```

```
  "orderId": "order-555",
  "status": "created",
  "totalAmount": 15990.00,
  "currency": "RUB",
  "createdAt": "2024-01-15T14:30:00Z"
}
```

Пример кода на C#:

```
// DTO модель запроса
public class CreateOrderRequest
{
    public string CartId { get; set; }
    public string UserId { get; set; }
    public DeliveryAddress DeliveryAddress { get; set; }
    public string PaymentMethod { get; set; }
}
```

```
// Интерфейс интеграции
public interface IOrderService
{
    Task<CreateOrderResponse> CreateOrderAsync(CreateOrderRequest request);
}

// Реализация клиента
public class OrderServiceClient : IOrderService
{
    private readonly HttpClient _httpClient;

    public OrderServiceClient(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }

    public async Task<CreateOrderResponse> CreateOrderA-
sync(CreateOrderRequest request)
    {
        var response = await _httpClient.PostAsJsonAsync("/api/orders/create", re-
quest);
        response.EnsureSuccessStatusCode();
        return await re-
sponse.Content.ReadFromJsonAsync<CreateOrderResponse>();
    }
}
```

Структура репозитория:

OnlineStore/

```
├── docs/
│   ├── architecture/
│   │   └── component-diagram.png
│   ├── protocols/
│   │   ├── cart-to-order-protocol.md
│   │   └── order-to-payment-protocol.md
│   └── api/
│       └── api-specification.md
├── src/
│   ├── CatalogService/
│   │   └── CatalogService.csproj
│   ├── CartService/
│   │   └── CartService.csproj
│   ├── OrderService/
│   │   └── OrderService.csproj
│   ├── Shared/
│   │   ├── DTOs/
│   │   └── Interfaces/
│   └── OnlineStore.sln
├── tests/
│   ├── IntegrationTests/
│   └── UnitTests/
├── config/
│   ├── appsettings.Development.json
│   └── appsettings.Production.json
├── README.md
├── CHANGELOG.md
└── .gitignore
```

Контрольные вопросы

1.Что такое артефакт проекта? Приведите классификацию артефактов программного проекта и примеры для каждой категории.

2.Какова роль протоколов интеграции в обеспечении взаимодействия программных модулей? Перечислите основные типы протоколов интеграции.

3.В чем заключаются различия между форматами JSON и XML для обмена данными? Приведите преимущества и недо-

статки каждого формата.

4.Что должна включать в себя документация протокола интеграции? Почему важно детально документировать протоколы взаимодействия модулей?

5.Как организация артефактов в системе контроля версий (Git) влияет на эффективность командной разработки и интеграции модулей?

6.Какие транспортные протоколы используются для интеграции программных модулей? В каких случаях целесообразно использовать HTTP/REST, а в каких — очереди сообщений (AMQP)?

7.Что такое DTO (Data Transfer Object)? Для чего используются DTO при интеграции модулей в C# приложениях?

8.Какие требования безопасности необходимо учитывать при проектировании протоколов интеграции между модулями системы?

9.Что такое матрица артефактов и какую информацию она должна содержать для эффективного управления проектной документацией?

10.Как связаны артефакты проекта между собой? Приведите примеры зависимостей между различными типами артефактов (например, между техническим заданием и спецификациями интерфейсов).

Время выполнения: 2 ч

Практическая работа № 4 «настройка работы системы контроля версий (типов импортируемых файлов, путей, фильтров и др. Параметров импорта в репозиторий)»

Цель работы: Освоить практические навыки настройки системы контроля версий Git для работы с проектами на языке C# в среде Visual Studio 2022, научиться конфигурировать параметры импорта файлов в репозиторий, создавать файлы gitignore и .gitattributes, настраивать фильтры и пути для оптимальной работы с версионным контролем.

Теоретическая часть

Система контроля версий Git. Система контроля версий

(Version Control System, VCS) — это программное обеспечение, которое позволяет отслеживать изменения в файлах проекта, сохранять историю модификаций, работать в команде над одним проектом и при необходимости возвращаться к предыдущим версиям кода.

Git — распределенная система контроля версий, которая является стандартом де-факто в современной разработке программного обеспечения. Основные преимущества Git:

- распределенная архитектура (каждый разработчик имеет полную копию репозитория);
- высокая скорость работы;
- эффективная работа с ветками;
- надежность и целостность данных;
- широкая поддержка инструментов и платформ.

Структура репозитория Git. Репозиторий (repository) — это хранилище файлов проекта и истории их изменений. Репозиторий Git содержит:

- **Рабочая директория (working directory)** — файлы проекта, с которыми вы работаете;
- **Индекс (staging area)** — промежуточная область для подготовки коммитов;
- **Директория .git** — служебная папка, содержащая всю историю изменений, конфигурацию и метаданные репозитория.

Файл .gitignore. Файл .gitignore определяет, какие файлы и директории не должны отслеживаться системой контроля версий. Это критически важно для исключения:

- временных файлов компиляции;
- файлов IDE и редакторов;
- файлов конфигурации пользователя;
- скомпилированных исполняемых файлов;
- файлов с конфиденциальной информацией (пароли, ключи API);
- больших бинарных файлов, которые не нужно версионировать.

Синтаксис .gitignore:

Комментарий

*.log # Игнорировать все файлы с расширением .log
/build/ # Игнорировать папку build в корне
temp/ # Игнорировать все папки temp
*.dll # Игнорировать все DLL файлы
!important.dll # Но отслеживать important.dll
node_modules/ # Игнорировать папку node_modules
**/*.tmp # Игнорировать .tmp файлы во всех поддиректориях

Типичные файлы для игнорирования в проектах C#:

- bin/, obj/ — директории с результатами компиляции;
- *.user, *.suo — пользовательские настройки Visual Studio;
- packages/ — папка с NuGet пакетами (если используется packages.config);
- .vs/ — папка с настройками Visual Studio 2015+;
- *.exe, *.dll, *.pdb — исполняемые и отладочные файлы.

Файл .gitattributes. Файл .gitattributes управляет атрибутами файлов в репозитории Git. Он позволяет:

- нормализовать окончания строк (LF/CRLF);
- определять типы файлов для правильного отображения diff;
- настраивать стратегии слияния для конкретных файлов;
- указывать файлы как бинарные;
- настраивать фильтры обработки файлов.

Примеры использования .gitattributes:

```
# Автоматическая нормализация окончаний строк
* text=auto

# Файлы C# всегда с CRLF
*.cs text eol=crlf
*.csproj text eol=crlf

# Файлы shell скриптов всегда с LF
*.sh text eol=lf

# Бинарные файлы
*.dll binary
*.exe binary
*.png binary
*.jpg binary
```

```
# Файлы для linguist (статистика языков на GitHub)
*.generated.cs linguist-generated=true
```

Настройка Git в Visual Studio 2022. Visual Studio 2022 имеет встроенную поддержку Git, которая включает:

- создание и клонирование репозиториев;
- коммиты и синхронизацию изменений;
- работу с ветками;
- просмотр истории изменений;
- разрешение конфликтов;
- интеграцию с удаленными репозиториями (GitHub, Azure DevOps, GitLab и др.).

Основные элементы интерфейса Git в Visual Studio:

- **Git Changes** (Git-изменения) — окно для просмотра изменений и создания коммитов;
- **Git Repository** (Репозиторий Git) — окно для работы с ветками и историей;
- **Team Explorer** — альтернативный интерфейс для работы с Git (унаследован из VS 2019).

Конфигурация Git. Git хранит конфигурацию на трех уровнях:

1.Системный уровень — применяется ко всем пользователям системы;

2.Глобальный уровень — применяется к текущему пользователю;

3.Локальный уровень — применяется к конкретному репозиторию.

Основные параметры конфигурации:

```
# Имя пользователя
git config --global user.name "Иван Иванов"

# Email пользователя
git config --global user.email "ivanov@example.com"

# Редактор по умолчанию
git config --global core.editor "code --wait"

# Автоматическое исправление окончаний строк (Windows)
git config --global core.autocrlf true

# Цветной вывод в консоли
git config --global color.ui auto
```

Стратегии работы с бинарными файлами. Бинарные файлы требуют особого подхода в Git:

- **Небольшие бинарные файлы** (иконки, изображения UI) — можно хранить в репозитории;
- **Большие бинарные файлы** (видео, крупные изображения, скомпилированные библиотеки) — использовать Git LFS (Large File Storage);
- **Генерируемые файлы** (результаты компиляции) — добавлять в .gitignore;
- **Зависимости** (NuGet пакеты) — восстанавливать при сборке, не хранить в репозитории.

Git LFS (Large File Storage). **Git LFS** — расширение Git для эффективной работы с большими файлами. Вместо хранения больших файлов непосредственно в репозитории, Git LFS сохраняет в репозитории только указатели на файлы, а сами файлы хранятся отдельно**.**

Установка и настройка Git LFS:

```
# Установка Git LFS
git lfs install

# Отслеживание файлов определенного типа
git lfs track "*.psd"
git lfs track "*.mp4"

# Просмотр отслеживаемых файлов
git lfs track

# Добавление .gitattributes в репозиторий
git add .gitattributes
```

Фильтры путей в Git. Git позволяет применять операции только к определенным путям:

```
# Добавить только файлы из папки src
git add src/

# Просмотреть изменения только в файлах .cs
git diff -- "*.cs"

# Коммит изменений только в конкретной папке
git commit -- src/Models/
```

Шаблоны .gitignore для проектов C#. Для проектов C# в Visual

Studio рекомендуется использовать стандартный шаблон .gitignore, который включает:

- файлы компиляции (bin/, obj/);
- пользовательские настройки Visual Studio;
- файлы ReSharper;
- файлы тестирования;
- файлы публикации;
- временные файлы.

Готовые шаблоны доступны по адресу: https://github.com/github/gitignore

Рабочее задание: Выполните настройку системы контроля версий Git для проекта C# в соответствии с вариантом задания. Создайте необходимые конфигурационные файлы и проведите инициализацию репозитория с учетом специфики проекта.

Вариант 1. Настройка репозитория для консольного приложения C#. 1.Создайте консольное приложение C# в Visual Studio 2022 для калькулятора математических операций. 2.Инициализируйте Git-репозиторий для проекта. 3.Создайте файл .gitignore, исключающий:

- папки bin/ и obj/;
- файлы пользовательских настроек Visual Studio (.vs/, *.user, *.suo);
- файлы компиляции (*.exe, *.dll, *.pdb).

4.Создайте файл .gitattributes для нормализации окончаний строк в файлах C#. 5.Настройте глобальные параметры Git (имя пользователя и email). 6.Выполните первый коммит с сообщением "Initial commit: Console Calculator application". 7.Создайте файл README.md с описанием проекта и добавьте его в репозиторий.

Вариант 2. Настройка репозитория для Windows Forms приложения. 1.Создайте Windows Forms приложение C# для управления списком задач (To-Do List). 2.Инициализируйте Git-репозиторий для проекта. 3.Создайте файл .gitignore, исключающий:

- директории компиляции;
- временные файлы дизайнера форм;
- файлы IntelliSense (.vs/);

- файлы пакетов NuGet (packages/), если используется packages.config.

4.Настройте .gitattributes для правильной обработки файлов .resx и .Designer.cs. 5.Выполните коммит начальной структуры проекта. 6.Добавьте несколько изображений (иконки для кнопок) и убедитесь, что они корректно обрабатываются как бинарные файлы. 7.Создайте ветку feature/ui-improvements для дальнейшей работы над интерфейсом.

Вариант 3. Настройка репозитория для WPF приложения с ресурсами. 1.Создайте WPF приложение C# для просмотра галереи изображений. 2.Инициализируйте Git-репозиторий. 3.Создайте расширенный файл .gitignore, включающий:

- стандартные исключения для C#;
- временные файлы XAML дизайнера;
- папку с тестовыми изображениями (/TestImages/).

4.Настройте .gitattributes для: нормализации окончаний строк в .xaml файлах; обработки изображений .png и .jpg как бинарных файлов. 5.Добавьте в проект несколько небольших изображений (иконки) и зафиксируйте их в репозитории. 6.Создайте файл .gitattributes с пометкой файлов *.Designer.cs как автоматически генерируемых. 7.Выполните коммит с подробным описанием структуры проекта.

Вариант 4. Настройка репозитория для ASP.NET Core Web API проекта. 1.Создайте ASP.NET Core Web API проект для простого REST API управления пользователями. 2.Инициализируйте Git-репозиторий. 3.Создайте файл .gitignore, исключающий:

- папки bin/, obj/, wwwroot/;
- файлы логов (*.log);
- файлы настроек окружения (appsettings.Development.json — создайте шаблон appsettings.Development.template.json);
- папки публикации (/publish/).

4.Настройте .gitattributes для файлов .json и .cs. 5.Создайте файл .gitignore для исключения конфиденциальных данных (строки подключения к БД). 6.Настройте локальную конфигурацию Git для этого проекта (отдельный email для рабочих проектов).

7.Создайте файл CONTRIBUTING.md с правилами работы с репозиторием и зафиксируйте его.

Вариант 5. Настройка репозитория для библиотеки классов (Class Library). 1.Создайте проект Class Library (.NET Standard 2.0) для математических операций с матрицами. 2.Инициализируйте Git-репозиторий. 3.Создайте файл .gitignore для библиотеки, исключающий:

- результаты компиляции;
- NuGet пакеты;
- файлы документации XML, генерируемые автоматически.

4.Создайте .gitattributes для правильной обработки файлов .cs и .csproj. 5.Настройте игнорирование файлов пользовательских настроек NuGet (*.nuget.props, *.nuget.targets). 6.Создайте структуру папок: /src, /tests, /docs.
7.Выполните первый коммит и создайте тег v0.1.0-alpha.

Вариант 6. Настройка репозитория для многопроектного решения (Solution). 1.Создайте решение Visual Studio, содержащее:

- консольное приложение (основной проект);
- библиотеку классов (бизнес-логика);
- проект Unit-тестов (xUnit или NUnit).

2.Инициализируйте Git-репозиторий на уровне Solution.
3.Создайте файл .gitignore, исключающий:

- папки bin/ и obj/ для всех проектов;
- результаты тестирования (TestResults/);
- файлы покрытия кода (*.coverage, *.coveragexml).

4.Настройте .gitattributes для различных типов файлов в решении. 5.Создайте отдельные README.md для каждого проекта в решении. 6.Выполните коммит структуры решения.
7.Создайте ветки develop и main (или master), настройте develop как ветку по умолчанию.

Вариант 7. Настройка репозитория с использованием Git LFS. 1.Создайте Windows Forms приложение для работы с мультимедиа-файлами. 2.Инициализируйте Git-репозиторий.
3.Установите и настройте Git LFS для проекта. 4.Настройте отслеживание больших файлов через Git LFS:

- видеофайлы (*.mp4, *.avi);
- аудиофайлы (*.mp3, *.wav);
- большие изображения (*.psd, *.tiff).

5.Создайте файл .gitignore для стандартных файлов C#. 6.Добавьте несколько тестовых файлов разных типов и убедитесь, что большие файлы обрабатываются через LFS. 7.Просмотрите содержимое файла .gitattributes и объясните назначение каждой строки.

Вариант 8. Настройка репозитория с конфиденциальными данными. 1.Создайте консольное приложение для работы с API (например, погодный сервис). 2.Инициализируйте Git-репозиторий. 3.Создайте файл конфигурации appsettings.json с API ключами. 4.Настройте .gitignore для исключения файла с реальными ключами (appsettings.json). 5.Создайте шаблонный файл appsettings.template.json с примерами значений и добавьте его в репозиторий. 6.Создайте файл README.md с инструкциями по настройке конфигурации для других разработчиков. 7.Настройте глобальный .gitignore для вашей системы, чтобы исключить файлы операционной системы (Thumbs.db, .DS_Store). 8.Выполните коммит проекта, убедившись, что конфиденциальные данные не попали в репозиторий.

Вариант 9. Настройка репозитория для Entity Framework проекта. 1.Создайте консольное приложение C# с использованием Entity Framework Core для работы с базой данных SQLite. 2.Инициализируйте Git-репозиторий. 3.Создайте файл .gitignore, исключающий:

- файлы базы данных (*.db, *.sqlite, *.db-shm, *.db-wal);
- папки миграций (если они генерируются автоматически);
- логи Entity Framework.

4.Создайте файл .gitattributes для .cs и .csproj файлов. 5.Добавьте в репозиторий пример структуры БД (SQL-скрипт для создания схемы). 6.Создайте файл с начальными данными (seed data) и добавьте его в репозиторий. 7.Выполните коммит с описанием структуры данных проекта.

Вариант 10. Настройка репозитория с подмодулями Git. 1.Создайте основной проект консольного приложения C#. 2.Создайте отдельный репозиторий для библиотеки утилит

(вспомогательных функций). 3.Инициализируйте Git-репозиторий для основного проекта. 4.Добавьте библиотеку утилит как подмодуль Git в основной проект:
.git submodule add <URL-репозитория-библиотеки> libs/utilities
5.Создайте файл .gitignore для основного проекта.
6.Настройте .gitmodules для управления подмодулем.
7.Выполните коммит основного проекта вместе с подмодулем.
8.Создайте документацию по работе с подмодулями в файле README.md.

Требования к выполнению задания

1.Создание проекта: Проект должен быть создан в Visual Studio 2022; Использовать актуальную версию .NET (6.0 или выше); Проект должен компилироваться без ошибок.
2.Инициализация репозитория: Репозиторий должен быть инициализирован через Visual Studio или командную строку Git ; Проверить наличие папки .git в корне проекта.
3.Создание .gitignore: Файл должен содержать все необходимые исключения для проектов C#; Использовать комментарии для пояснения назначения правил; Проверить корректность работы файла (исключенные файлы не должны отображаться в Git Changes).
4.Создание .gitattributes: Файл должен содержать правила для нормализации окончаний строк; Указать бинарные типы файлов; Добавить правила для специфичных типов файлов проекта.
5.Настройка конфигурации Git: Установить корректное имя пользователя и email; При необходимости настроить локальную конфигурацию для конкретного репозитория.
6.Выполнение коммитов: Первый коммит должен содержать базовую структуру проекта; Сообщения коммитов должны быть информативными и следовать best practices; Использовать формат: <тип>: <краткое описание> (например, feat: Add Calculator class).
7.Документация: Создать файл README.md с описанием проекта; Указать инструкции по клонированию и запуску проекта; При необходимости добавить файлы CONTRIBUTING.md или LICENSE.

Содержание отчета. Отчет по практической работе должен содержать следующие разделы:

1.Ход выполнения работы: Скриншот созданного проекта в Visual Studio 2022; Описание процесса инициализации Git-репозитория (команды или действия в Visual Studio); Содержимое файла .gitignore с комментариями; Содержимое файла .gitattributes с пояснениями; Скриншот окна Git Changes в Visual Studio с подготовленными к коммиту файлами; Скриншот выполненного первого коммита; Скриншот истории коммитов в Git Repository; Содержимое файла README.md.; Результат выполнения команды git config --list (конфигурация Git); Структура каталогов проекта (скриншот Solution Explorer).

2.Анализ выполненной работы: Какие файлы были исключены из отслеживания и почему; Обоснование выбора правил в .gitignore и .gitattributes; Какие проблемы возникли при выполнении работы и как они были решены; Какие преимущества дает правильная настройка системы контроля версий для командной разработки.

3.Выводы по работе

Контрольные вопросы

1.Что такое система контроля версий и какие задачи она решает в процессе разработки программного обеспечения?

2.В чем различие между централизованными и распределенными системами контроля версий? Приведите примеры.

3.Каково назначение файла .gitignore? Какие файлы и директории следует исключать из отслеживания в проектах C#?

4.Объясните назначение файла .gitattributes. Для чего необходима нормализация окончаний строк в файлах проекта?

5.Какие три уровня конфигурации существуют в Git? Приведите примеры использования каждого уровня.

6.Что такое Git LFS и в каких случаях его следует использовать? Какие типы файлов рекомендуется хранить через LFS?

7.Объясните разницу между рабочей директорией (working directory), индексом (staging area) и репозиторием в Git.

8.Какие файлы, создаваемые Visual Studio, следует добавить в .gitignore? Почему важно исключать файлы пользовательских

настроек из репозитория?

9.Что такое коммит в Git? Какие правила следует соблюдать при написании сообщений коммитов?

10.Почему важно правильно настроить систему контроля версий на начальном этапе проекта? К каким проблемам может привести неправильная конфигурация?

Время выполнения: 2 часа

Практическая работа № 5 «Разработка и интеграция модулей проекта (командная работа)»

Цель работы: Освоить практические навыки командной разработки программного обеспечения с использованием системы контроля версий Git, изучить процесс интеграции программных модулей, разработанных различными участниками команды, научиться применять инструменты Visual Studio 2022 для совместной работы над проектом на языке C#.

Теоретическая часть

Командная разработка программного обеспечения. Командная разработка — это процесс создания программного обеспечения группой специалистов, работающих над общим проектом. Эффективная командная работа требует четкой организации, распределения ролей и использования специализированных инструментов.

Основные принципы командной разработки:

- модульность кода — разделение проекта на независимые модули;
- стандартизация — единые правила оформления кода и именования;
- документирование — описание интерфейсов и функциональности модулей;
- контроль версий — отслеживание изменений в коде;
- непрерывная интеграция — регулярное объединение модулей;
- код-ревью — взаимная проверка кода участниками команды.

Система контроля версий Git. **Git** — распределенная система контроля версий, позволяющая отслеживать изменения в файлах проекта и координировать работу нескольких разработчиков.
Основные понятия Git:
Репозиторий (repository) — хранилище файлов проекта вместе с историей их изменений. Репозитории бывают локальными (на компьютере разработчика) и удаленными (на сервере, например, GitHub, GitLab).
Коммит (commit) — фиксация изменений в локальном репозитории. Каждый коммит содержит информацию о внесенных изменениях, авторе и времени.
Ветка (branch) — независимая линия разработки. Позволяет вести параллельную работу над разными функциями без влияния на основной код.
Слияние (merge) — объединение изменений из разных веток в одну.
Конфликт (conflict) — ситуация, когда несколько разработчиков изменили одни и те же строки кода. Требует ручного разрешения.
Основные команды Git:

```
git init                    # Инициализация нового репозитория
git clone <url>             # Клонирование удаленного репозитория
git status                  # Просмотр состояния рабочей директории
git add <file>              # Добавление файла в индекс
git commit -m "message"     # Фиксация изменений
git push                    # Отправка изменений в удаленный репозиторий
git pull                    # Получение изменений из удаленного репозитория
git branch <name>           # Создание новой ветки
git checkout <branch>       # Переключение на ветку
git merge <branch>          # Слияние ветки с текущей
```

Структура проекта в Visual Studio
Решение (Solution) — контейнер для одного или нескольких проектов. Файл решения имеет расширение .sln.
Проект (Project) — набор файлов, компилируемых в единую сборку (assembly). Файл проекта C# имеет расширение .csproj.

Типовая структура проекта:

```
SolutionName/
├── SolutionName.sln
├── ProjectName/
│   ├── ProjectName.csproj
│   ├── Program.cs
│   ├── Models/          # Модели данных
│   ├── Services/        # Бизнес-логика
│   ├── Repositories/    # Работа с данными
│   ├── Utilities/       # Вспомогательные классы
│   └── Interfaces/      # Интерфейсы
└── Tests/               # Проект с тестами
```

Интеграция программных модулей. Интеграция программных модулей — процесс объединения отдельно разработанных компонентов в единую систему с проверкой их совместной работы.
Уровни интеграции:

1. Интеграция компонентов — объединение отдельных классов и методов.
2. Интеграция подсистем — объединение модулей в функциональные подсистемы.
3. Интеграция системы — объединение всех подсистем в готовый продукт.

Стратегии интеграции:
Снизу вверх (Bottom-Up) — сначала интегрируются низкоуровневые модули, затем модули более высокого уровня.
Сверху вниз (Top-Down) — сначала интегрируются модули верхнего уровня, для нижних используются заглушки (stubs).
Большой взрыв (Big Bang) — все модули интегрируются одновременно. Не рекомендуется для крупных проектов.
Непрерывная интеграция (Continuous Integration) — модули интегрируются постепенно, по мере готовности.
Интерфейсы в C#. Интерфейсы играют ключевую роль в командной разработке, определяя контракт взаимодействия между модулями. *Пример интерфейса:*

```
public interface IDataRepository
{
    void Add(object item);
```

```
    void Remove(int id);
    object GetById(int id);
    List<object> GetAll();
}
```

Преимущества использования интерфейсов:

- определяют четкий контракт между модулями;
- позволяют разрабатывать модули независимо;
- упрощают тестирование (mock-объекты);
- обеспечивают слабую связанность (loose coupling).

Работа с Git в Visual Studio 2022. Visual Studio 2022 имеет встроенную поддержку Git. Основные действия**:**

1.Создание репозитория:

- Файл → Создать → Репозиторий
- Или: Git → Создать репозиторий Git

2.Клонирование репозитория:

- Git → Клонировать репозиторий
- Ввод URL удаленного репозитория

3.Коммит изменений:

- Git → Зафиксировать или отложить
- Ввод сообщения коммита
- Кнопка "Зафиксировать все"

4.Работа с ветками:

- Переключение: Git → Управление ветвями
- Создание новой ветки
- Слияние веток

5.Синхронизация:

- Git → Отправить (Push)
- Git → Получить (Pull)
- Git → Синхронизировать

Разрешение конфликтов. Конфликты возникают при одновременном изменении одних и тех же участков кода разными разработчиками. Процесс разрешения конфликта:

1. При попытке слияния Git сообщает о конфликте
2. Открыть файл с конфликтом (помечен маркерами)
3. Visual Studio предлагает инструмент сравнения
4. Выбрать нужный вариант или объединить изменения вручную
5. Удалить маркеры кон- фликта

6. Сохранить файл
7. Выполнить коммит разрешенного конфликта

Маркеры конфликта в коде:

<<<<<<< HEAD
// Ваши изменения
=======
// Изменения другого разработчика
>>>>>>> branch-name

Соглашения о коде (Code Conventions)

Для успешной командной работы необходимо соблюдать единый стиль кодирования.

Основные соглашения для C#:

- Имена классов: PascalCase (UserRepository)
- Имена методов: PascalCase (GetUserById)
- Имена переменных: camelCase (userId)
- Имена интерфейсов: I + PascalCase (IUserService)
- Константы: UPPER_CASE (MAX_COUNT)
- Отступы: 4 пробела
- Фигурные скобки: на новой строке

Комментарии XML для документирования:

```
/// <summary>
/// Получает пользователя по идентификатору
/// </summary>
/// <param name="id">Идентификатор пользователя</param>
/// <returns>Объект пользователя или null</returns>
public User GetUserById(int id)
{
    // Реализация
}
```

Рабочее задание: Выполните разработку программной системы в команде из 3-4 человек в соответствии с вариантом задания. Каждый участник команды разрабатывает свой модуль. Интеграция модулей осуществляется с использованием системы контроля версий Git.

Вариант 1. Система управления библиотекой. Разработать консольное приложение для управления библиотечным фондом с возможностью учета книг, читателей и операций выдачи/возврата. Распределение модулей:

Модуль 1 — Управление книгами (BookManagement)

- Класс Book (свойства: Id, Title, Author, ISBN, Year, IsAvailable)
- Интерфейс IBookRepository
- Класс BookRepository (методы: Add, Remove, Update, GetById, GetAll, Search)
- Сохранение данных в текстовый файл или JSON

Модуль 2 — Управление читателями (ReaderManagement)

- Класс Reader (свойства: Id, FullName, Phone, Email, RegistrationDate)
- Интерфейс IReaderRepository
- Класс ReaderRepository (методы: Add, Remove, Update, GetById, GetAll, Search)
- Сохранение данных в текстовый файл или JSON

Модуль 3 — Операции выдачи/возврата (LoanManagement)

- Класс Loan (свойства: Id, BookId, ReaderId, IssueDate, ReturnDate, Status)
- Интерфейс ILoanService
- Класс LoanService (методы: IssueBook, ReturnBook, GetActiveLoans, GetHistory)
- Проверка доступности книги, расчет сроков

Модуль 4 — Пользовательский интерфейс и интеграция (UI & Integration)

- Класс Program (главное меню приложения)
- Класс MenuHandler (обработка выбора пользователя)
- Интеграция всех модулей
- Обработка исключений
- Валидация ввода данных

Вариант 2. Система учета студентов и оценок. Разработать консольное приложение для учета студентов, дисциплин и их успеваемости. Распределение модулей:

Модуль 1 — Управление студентами (StudentManagement)

- Класс Student (свойства: Id, FullName, Group, EnrollmentYear, StudentId)
- Интерфейс IStudentRepository
- Класс StudentRepository (методы: Add, Remove, Update, GetById, GetAll, GetByGroup)

- Сохранение данных в файл

Модуль 2 — Управление дисциплинами (SubjectManagement)

- Класс Subject (свойства: Id, Name, Code, Hours, Semester)
- Интерфейс ISubjectRepository
- Класс SubjectRepository (методы: Add, Remove, Update, GetById, GetAll)
- Сохранение данных в файл

Модуль 3 — Управление оценками (GradeManagement)

- Класс Grade (свойства: Id, StudentId, SubjectId, Score, Date, GradeType)
- Интерфейс IGradeService
- Класс GradeService (методы: AddGrade, UpdateGrade, GetStudentGrades, CalculateAverage)
- Формирование ведомостей

Модуль 4 — Отчетность и интеграция (Reporting & Integration)

- Класс Program (главное меню)
- Класс ReportGenerator (методы: GenerateStudentReport, GenerateGroupReport, GenerateSubjectReport)
- Интеграция модулей
- Расчет среднего балла, рейтинга студентов

Вариант 3. Система управления складом. Разработать консольное приложение для учета товаров на складе с возможностью регистрации поступлений и отгрузок. Распределение модулей:

Модуль 1 — Управление товарами (ProductManagement)

- Класс Product (свойства: Id, Name, Category, Unit, Price, MinStock)
- Интерфейс IProductRepository
- Класс ProductRepository (методы: Add, Remove, Update, GetById, GetAll, GetByCategory)
- Сохранение данных в файл

Модуль 2 — Учет остатков (StockManagement)

- Класс Stock (свойства: ProductId, Quantity, LastUpdated, Location)
- Интерфейс IStockRepository
- Класс StockRepository (методы: GetStock, UpdateStock, CheckMinStock, GetLowStockProducts)

- Контроль минимальных остатков

Модуль 3 — Операции прихода/расхода (TransactionManagement)

- Класс Transaction (свойства: Id, ProductId, Type, Quantity, Date, Note)
- Enum TransactionType (Receipt, Issue)
- Интерфейс ITransactionService
- Класс TransactionService (методы: RegisterReceipt, RegisterIssue, GetTransactionHistory)

Модуль 4 — Отчетность и интеграция (Reporting & Integration)

- Класс Program (главное меню)
- Класс ReportGenerator (методы: GenerateStockReport, GenerateTransactionReport, GenerateLowStockAlert)
- Интеграция модулей
- Валидация операций

Вариант 4. Система учета заявок технической поддержки. Разработать консольное приложение для регистрации и обработки заявок в службе технической поддержки. Распределение модулей:

Модуль 1 — Управление пользователями (UserManagement)

- Класс User (свойства: Id, FullName, Email, Phone, Department, Role)
- Enum UserRole (Client, Technician, Manager)
- Интерфейс IUserRepository
- Класс UserRepository (методы: Add, Remove, Update, GetById, GetAll, GetByRole)

Модуль 2 — Управление заявками (TicketManagement)

- Класс Ticket (свойства: Id, ClientId, Title, Description, Priority, Status, CreatedDate)
- Enum Priority (Low, Medium, High, Critical)
- Enum TicketStatus (New, InProgress, Resolved, Closed)
- Интерфейс ITicketRepository
- Класс TicketRepository (методы: Add, Update, GetById, GetAll, GetByStatus)

Модуль 3 — Обработка заявок (TicketProcessing)

- Класс Assignment (свойства: TicketId, TechnicianId, AssignedDate, CompletedDate)

- Интерфейс ITicketService
- Класс TicketService (методы: AssignTicket, UpdateStatus, CompleteTicket, GetTechnicianWorkload)
- Логика назначения исполнителей

Модуль 4 — Отчетность и интеграция (Reporting & Integration)

- Класс Program (главное меню)
- Класс ReportGenerator (методы: GenerateTicketReport, GenerateTechnicianReport, GeneratePriorityReport)
- Интеграция модулей
- Статистика по заявкам

Вариант 5. Система управления проектами и задачами.

Разработать консольное приложение для управления проектами и задачами с распределением исполнителей. Распределение модулей:

Модуль 1 — Управление сотрудниками (EmployeeManagement)

- Класс Employee (свойства: Id, FullName, Position, Email, HireDate)
- Интерфейс IEmployeeRepository
- Класс EmployeeRepository (методы: Add, Remove, Update, GetById, GetAll)
- Сохранение данных в файл

Модуль 2 — Управление проектами (ProjectManagement)

- Класс Project (свойства: Id, Name, Description, StartDate, EndDate, Status, ManagerId)
- Enum ProjectStatus (Planning, Active, Completed, Cancelled)
- Интерфейс IProjectRepository
- Класс ProjectRepository (методы: Add, Remove, Update, GetById, GetAll, GetByStatus)

Модуль 3 — Управление задачами (TaskManagement)

- Класс Task (свойства: Id, ProjectId, Title, Description, AssignedTo, Deadline, Status, Priority)
- Enum TaskStatus (ToDo, InProgress, Done)
- Интерфейс ITaskService
- Класс TaskService (методы: CreateTask, AssignTask, UpdateStatus, GetProjectTasks, GetEmployeeTasks)

Модуль 4 — Отчетность и интеграция (Reporting & Integration)

- Класс Program (главное меню)

- Класс ReportGenerator (методы: GenerateProjectReport, GenerateEmployeeWorkload, GenerateOverdueTasks)
- Интеграция модулей
- Расчет прогресса проектов

Вариант 6. Система учета автомобилей и сервисного обслуживания. Разработать консольное приложение для учета автомобилей клиентов и регистрации сервисного обслуживания.
Распределение модулей:
Модуль 1 — Управление клиентами (ClientManagement)

- Класс Client (свойства: Id, FullName, Phone, Email, RegistrationDate)
- Интерфейс IClientRepository
- Класс ClientRepository (методы: Add, Remove, Update, GetById, GetAll, Search)
- Сохранение данных

Модуль 2 — Управление автомобилями (VehicleManagement)

- Класс Vehicle (свойства: Id, ClientId, Brand, Model, Year, VIN, PlateNumber)
- Интерфейс IVehicleRepository
- Класс VehicleRepository (методы: Add, Remove, Update, GetById, GetByClient, Search)
- Связь с клиентами

Модуль 3 — Сервисное обслуживание (ServiceManagement)

- Класс Service (свойства: Id, VehicleId, ServiceType, Description, Date, Cost, Mileage)
- Интерфейс IServiceRepository
- Класс ServiceRepository (методы: Add, GetByVehicle, GetServiceHistory, CalculateTotalCost)
- История обслуживания

Модуль 4 — Отчетность и интеграция (Reporting & Integration)

- Класс Program (главное меню)
- Класс ReportGenerator (методы: GenerateClientReport, GenerateVehicleHistory, GenerateRevenueReport)
- Интеграция модулей
- Статистика по обслуживанию

Вариант 7. Система учета посещаемости и рабочего времени. Разработать консольное приложение для учета посещаемости сотрудников и расчета рабочего времени. Распределение модулей:

Модуль 1 — Управление сотрудниками (EmployeeManagement)

- Класс Employee (свойства: Id, FullName, Position, Department, WorkSchedule)
- Интерфейс IEmployeeRepository
- Класс EmployeeRepository (методы: Add, Remove, Update, GetById, GetAll, GetByDepartment)
- Сохранение данных

Модуль 2 — Учет посещаемости (AttendanceTracking)

- Класс Attendance (свойства: Id, EmployeeId, Date, CheckIn, CheckOut, Status)
- Enum AttendanceStatus (Present, Absent, Late, Sick, Vacation)
- Интерфейс IAttendanceRepository
- Класс AttendanceRepository (методы: RegisterCheckIn, RegisterCheckOut, GetByDate, GetByEmployee)

Модуль 3 — Расчет рабочего времени (TimeCalculation)

- Класс WorkTime (свойства: EmployeeId, Date, HoursWorked, Overtime, Breaks)
- Интерфейс ITimeCalculationService
- Класс TimeCalculationService (методы: CalculateWorkHours, CalculateOvertime, GetMonthlyReport)
- Расчет переработок

Модуль 4 — Отчетность и интеграция (Reporting & Integration)

- Класс Program (главное меню)
- Класс ReportGenerator (методы: GenerateDailyReport, GenerateMonthlyReport, GenerateEmployeeTimesheet)
- Интеграция модулей
- Формирование табелей

Вариант 8. Система управления курсами и студентами. Разработать консольное приложение для управления учебными курсами, записью студентов и контролем прогресса. Распределение модулей:

Модуль 1 — Управление курсами (CourseManagement)

- Класс Course (свойства: Id, Name, Description, Duration, Instructor, MaxStudents)
- Интерфейс ICourseRepository
- Класс CourseRepository (методы: Add, Remove, Update, GetById, GetAll, GetAvailableCourses)
- Контроль количества мест

Модуль 2 — Управление студентами (StudentManagement)

- Класс Student (свойства: Id, FullName, Email, Phone, EnrollmentDate)
- Интерфейс IStudentRepository
- Класс StudentRepository (методы: Add, Remove, Update, GetById, GetAll)
- Сохранение данных

Модуль 3 — Запись на курсы (EnrollmentManagement)

- Класс Enrollment (свойства: Id, StudentId, CourseId, EnrollmentDate, Status, Progress)
- Enum EnrollmentStatus (Active, Completed, Cancelled)
- Интерфейс IEnrollmentService
- Класс EnrollmentService (методы: EnrollStudent, CancelEnrollment, UpdateProgress, GetStudentCourses)

Модуль 4 — Отчетность и интеграция (Reporting & Integration)

- Класс Program (главное меню)
- Класс ReportGenerator (методы: GenerateCourseReport, GenerateStudentProgress, GenerateCompletionStatistics)
- Интеграция модулей
- Статистика по курсам

Вариант 9. Система учета медицинских приемов. Разработать консольное приложение для регистрации пациентов, врачей и записи на прием в медицинском учреждении. Распределение модулей:

Модуль 1 — Управление пациентами (PatientManagement)

- Класс Patient (свойства: Id, FullName, DateOfBirth, Phone, Address, InsuranceNumber)
- Интерфейс IPatientRepository
- Класс PatientRepository (методы: Add, Remove, Update, GetById, GetAll, Search)
- Сохранение данных

Модуль 2 — Управление врачами (DoctorManagement)
- Класс Doctor (свойства: Id, FullName, Specialization, Phone, Schedule)
- Интерфейс IDoctorRepository
- Класс DoctorRepository (методы: Add, Remove, Update, GetById, GetAll, GetBySpecialization)
- Расписание врачей

Модуль 3 — Управление приемами (AppointmentManagement)
- Класс Appointment (свойства: Id, PatientId, DoctorId, DateTime, Reason, Status)
- Enum AppointmentStatus (Scheduled, Completed, Cancelled, NoShow)
- Интерфейс IAppointmentService
- Класс AppointmentService (методы: ScheduleAppointment, CancelAppointment, UpdateStatus, CheckAvailability)

Модуль 4 — Отчетность и интеграция (Reporting & Integration)
- Класс Program (главное меню)
- Класс ReportGenerator (методы: GenerateDailySchedule, GenerateDoctorWorkload, GeneratePatientHistory)
- Интеграция модулей
- Проверка доступности времени

Вариант 10. Система управления рестораном (заказы и меню). Разработать консольное приложение для управления меню ресторана, приема заказов и расчета стоимости. Распределение модулей:

Модуль 1 — Управление меню (MenuManagement)
- Класс MenuItem (свойства: Id, Name, Category, Description, Price, IsAvailable)
- Enum Category (Appetizer, MainCourse, Dessert, Beverage)
- Интерфейс IMenuRepository
- Класс MenuRepository (методы: Add, Remove, Update, GetById, GetAll, GetByCategory)

Модуль 2 — Управление столиками (TableManagement)
- Класс Table (свойства: Id, Number, Capacity, Status, Location)
- Enum TableStatus (Free, Occupied, Reserved)
- Интерфейс ITableRepository

- Класс TableRepository (методы: Add, Update, GetById, GetAll, GetAvailableTables)

Модуль 3 — Управление заказами (OrderManagement)

- Класс Order (свойства: Id, TableId, OrderItems, OrderDate, Status, TotalAmount)
- Класс OrderItem (свойства: MenuItemId, Quantity, Price)
- Enum OrderStatus (New, InProgress, Ready, Served, Paid)
- Интерфейс IOrderService
- Класс OrderService (методы: CreateOrder, AddItem, CalculateTotal, UpdateStatus, CloseOrder)

Модуль 4 — Отчетность и интеграция (Reporting & Integration)

- Класс Program (главное меню)
- Класс ReportGenerator (методы: GenerateDailyRevenue, GeneratePopularItems, GenerateTableOccupancy)
- Интеграция модулей
- Расчет выручки

Требования к выполнению задания

1. Организация командной работы

Формирование команды: Создать команду из 3-4 человек; Назначить лидера команды (ответственный за интеграцию); Распределить модули между участниками

Планирование: Определить интерфейсы взаимодействия между модулями; Согласовать структуру данных; Установить соглашения о кодировании; Определить формат хранения данных (JSON, текстовые файлы, XML)

2. Настройка системы контроля версий

Создание репозитория: Лидер команды создает репозиторий на GitHub/GitLab; Добавляет участников команды с правами доступа; Создает файл .gitignore для C# проектов; Создает файл README.md с описанием проекта

Стратегия ветвления: Основная ветка main — стабильная версия; Ветка develop — интеграционная ветка; Персональные ветки для каждого участника: feature/module-name

Рабочий процесс Git: Клонирование репозитория; Создание персональной ветки; Разработка модуля с регулярными коммитами; Отправка изменений в удаленный репозиторий; Создание Pull Request для слияния с develop; Код-ревью другими участни-

ками команды; Слияние после одобрения

3. Требования к разработке модулей

Каждый модуль должен содержать: Определение интерфейсов (контракты взаимодействия); Реализацию классов в соответствии с интерфейсами; Валидацию входных данных; Обработку исключений; XML-комментарии для публичных методов; Методы сохранения и загрузки данных

Соблюдение принципов: SOLID принципы; Разделение ответственности (Separation of Concerns); DRY (Don't Repeat Yourself); Инкапсуляция данных

Именование: Использование осмысленных имен; Следование соглашениям C# (Code Conventions); Префикс I для интерфейсов

4. Интеграция модулей

Процесс интеграции: Каждый участник завершает разработку своего модуля; Модули последовательно интегрируются в ветку develop; Лидер команды разрешает конфликты слияния; Проводится интеграционное тестирование; Устраняются выявленные проблемы; Финальная версия сливается в main

Проверка интеграции: Корректность взаимодействия модулей; Отсутствие дублирования функциональности; Единообразие обработки ошибок; Согласованность данных

5. Тестирование

Модульное тестирование: Каждый участник тестирует свой модуль отдельно; Проверка граничных случаев; Проверка обработки некорректных данных

Интеграционное тестирование: Проверка совместной работы модулей; Сквозное тестирование основных сценариев; Проверка сохранения и загрузки данных

Тестовые сценарии: Создание и сохранение объектов; Поиск и фильтрация данных; Обновление и удаление объектов; Взаимодействие связанных сущностей; Обработка ошибочного ввода

6. Документирование

Документация проекта должна включать:

README.md: Название проекта; Описание функциональности; Требования к окружению; Инструкция по сборке и запуску; Распределение ролей в команде

Документация кода: XML-комментарии для публичных мето-

дов и классов; Описание параметров и возвращаемых значений; Примеры использования

Техническая документация: Диаграмма классов; Описание интерфейсов взаимодействия; Формат хранения данных; Принятые архитектурные решения

Содержание отчета. Отчет оформляется командой и должен содержать следующие разделы:

1. Организация работы команды: Состав команды (ФИО, роли); Распределение модулей между участниками; Установленные соглашения (Code Conventions); Стратегия ветвления Git; График выполнения работ

2. Описание модулей. Для каждого модуля указать: Название модуля и разработчик; Назначение модуля; Реализованные интерфейсы; Реализованные классы; Основные методы с описанием; Формат хранения данных; Скриншоты кода ключевых фрагментов

3. Процесс интеграции: Описание последовательности интеграции модулей; Возникшие конфликты и способы их разрешения; Использованные команды Git; Скриншоты графа коммитов; Скриншоты Pull Requests

4. Результаты тестирования: Описание тестовых сценариев; Результаты модульного тестирования; Результаты интеграционного тестирования; Выявленные ошибки и их исправление; Скриншоты работы программы.

5. Структура проекта: Дерево каталогов и файлов проекта; Описание назначения основных файлов

6. Диаграммы: Диаграмма классов; Диаграмма компонентов (модулей); Диаграмма последовательности для основного сценария

7. Выводы и 8. Приложения: Листинги кода основных классов; Примеры файлов данных; Инструкция пользователя

Контрольные вопросы

1.Что такое система контроля версий и для чего она используется при командной разработке программного обеспечения?

2.Объясните разницу между локальным и удаленным репози-

торием Git. Какие команды используются для синхронизации?

3.Что такое ветка (branch) в Git? Опишите стратегию ветвления, которую вы использовали в проекте.

4.Что такое конфликт слияния (merge conflict)? Как его можно избежать и как разрешить, если он возник?

5.Что такое Pull Request и какова его роль в процессе командной разработки?

6.Объясните понятие "интеграция программных модулей". Какие стратегии интеграции вы знаете?

7.Зачем нужны интерфейсы при разработке модульной архитектуры? Какие преимущества они дают при командной работе?

8.Что такое соглашения о кодировании (Code Conventions) и почему важно их соблюдать в команде?

9.Какие принципы SOLID вы применяли при разработке вашего модуля? Приведите примеры.

10.Опишите структуру решения (Solution) в Visual Studio. Чем отличается решение от проекта?

11.Какие уровни тестирования существуют? В чем разница между модульным и интеграционным тестированием?

12.Что такое непрерывная интеграция (Continuous Integration) и какие преимущества она дает?

13.Какие способы хранения данных вы использовали в проекте? Обоснуйте выбор.

14.Как организовать взаимодействие между модулями, разработанными разными участниками команды?

15.Какие инструменты Visual Studio 2022 вы использовали для работы с Git?

Время выполнения: 4 ч

Практическая работа № 6 «Отладка отдельных модулей программного проекта»

Цель работы: Изучить методы и инструменты отладки программных модулей в среде Visual Studio 2022, освоить приёмы пошаговой отладки, использования точек останова, анализа значений переменных и стека вызовов при разработке программного обеспечения на языке C#.

Теоретическая часть

Понятие отладки программного обеспечения. Отладка — это процесс обнаружения, локализации и устранения ошибок в программном коде. Отладка является неотъемлемой частью разработки программного обеспечения и выполняется на всех этапах создания программного продукта.

Классификация ошибок

Синтаксические ошибки — нарушения правил языка программирования, обнаруживаются компилятором на этапе компиляции.

Логические ошибки — ошибки в алгоритме программы, приводящие к неправильным результатам работы при синтаксически корректном коде.

Ошибки времени выполнения — возникают в процессе работы программы (деление на ноль, обращение к несуществующему элементу массива, нарушение доступа к памяти и т.д.).

Методы отладки

Пошаговое выполнение — последовательное выполнение программы по одной инструкции с контролем изменения состояния.

Точки останова (Breakpoints) — метки в коде, при достижении которых выполнение программы приостанавливается.

Просмотр значений переменных — анализ текущих значений переменных в процессе выполнения программы.

Трассировка стека вызовов — анализ последовательности вызовов методов, приведших к текущему состоянию программы.

Окно отладки — специализированные окна IDE для вывода диагностической информации.

Инструменты отладки Visual Studio 2022. Основные команды отладки:

- **F5 (Start Debugging)** — запуск программы в режиме отладки
- **F9** — установка/снятие точки останова на текущей строке
- **F10 (Step Over)** — выполнение текущей строки с переходом к следующей (без захода в методы)
- **F11 (Step Into)** — выполнение текущей строки с заходом внутрь вызываемых методов
- **Shift+F11 (Step Out)** — выход из текущего метода
- **Ctrl+F10 (Run to Cursor)** — выполнение программы до

позиции курсора
- **Shift+F5** — остановка отладки
- **Ctrl+Shift+F5** — перезапуск отладки

Окна отладки Visual Studio:
Locals (Локальные переменные) — отображает все локальные переменные текущего контекста выполнения и их значения.
Autos — автоматически отображает переменные, используемые в текущей и предыдущей строках кода.
Watch (Окно контрольных значений) — позволяет добавлять выражения для постоянного наблюдения за их значениями.
Call Stack (Стек вызовов) — показывает последовательность вызовов методов, приведших к текущей точке выполнения.
Immediate Window (Окно интерпретации) — позволяет выполнять команды и вычислять выражения во время отладки.
Output (Вывод) — отображает диагностические сообщения, выводимые через Debug.WriteLine() или Console.WriteLine().
Exception Settings — настройка поведения отладчика при возникновении исключений.
Типы точек останова
Обычная точка останова — выполнение останавливается каждый раз при достижении указанной строки.
Условная точка останова — выполнение останавливается только при выполнении определённого условия.
Точка останова с числом попаданий — срабатывает после определённого количества проходов через данную строку.
Точка трассировки (Tracepoint) — выводит сообщение в окно Output без остановки выполнения.
Использование класса Debug. Класс Debug из пространства имён System.Diagnostics предоставляет методы для вывода диагностической информации:
Debug.WriteLine("Сообщение");
Debug.Assert(условие, "Сообщение об ошибке");
Debug.WriteLineIf(условие, "Условное сообщение");
Эти методы работают только в режиме Debug и исключаются из Release-сборки.
Отладка модулей в контексте интеграции
При интеграции программных модулей особое внимание уделя-

ется:
– **Проверке корректности передачи данных** между модулями
– **Валидации входных параметров** методов
– **Обработке исключительных ситуаций** на границах модулей
– **Контролю состояния** объектов после вызова методов других модулей
- Анализу последовательности вызовов при взаимодействии модулей

Стратегии отладки

Отладка "сверху вниз" — начинается с высокоуровневых модулей, постепенно спускаясь к деталям реализации.
Отладка "снизу вверх" — начинается с отладки элементарных модулей, затем переходит к более сложным компонентам.
Метод "разделяй и властвуй" — разбиение проблемы на части для локализации источника ошибки.
Обратная трассировка — анализ последовательности действий, приведших к ошибке, начиная с момента её проявления.

Лучшие практики отладки

- Использовать осмысленные имена переменных для упрощения анализа
- Применять модульное тестирование для выявления ошибок на ранних этапах
- Документировать обнаруженные ошибки и способы их устранения
- Использовать систему контроля версий для отслеживания изменений
- Регулярно проводить code review для предотвращения ошибок
- Применять defensive programming — проверять входные данные и предусловия

Рабочее задание: Выполните отладку программного модуля в соответствии с вариантом задания. Необходимо обнаружить и устранить все ошибки в предоставленном коде, используя инструменты отладки Visual Studio 2022.

Вариант 1. Калькулятор простых операций. Дан модуль калькулятора с ошибками. Необходимо найти и исправить все ошибки**.**

```
using System;
namespace CalculatorModule
{
   public class Calculator
   {
      public double Add(double a, double b)
      {
         return a - b; // Ошибка в операции
      }

      public double Subtract(double a, double b)
      {
         return a - b;
      }

      public double Multiply(double a, double b)
      {
         return a * b;
      }

      public double Divide(double a, double b)
      {
         return a / b; // Отсутствует проверка деления на ноль
      }

      public double Power(double baseNum, int exponent)
      {
          double result = 1;
         for (int i = 1; i <= exponent; i++) // Ошибка в условии
         {
            result *= baseNum;
         }
         return result;
      }

      public double Average(params double[] numbers)
      {
         double sum = 0;
         for (int i = 0; i <= numbers.Length; i++) // Выход за границы массива
         {
            sum += numbers[i];
         }
         return sum / numbers.Length;
      }
```

```
    }

    class Program
    {
        static void Main(string[] args)
        {
            Calculator calc = new Calculator();

            Console.WriteLine("Сложение 5 + 3 = " + calc.Add(5, 3));
            Console.WriteLine("Вычитание 10 - 4 = " + calc.Subtract(10, 4));
            Console.WriteLine("Умножение 6 * 7 = " + calc.Multiply(6, 7));
            Console.WriteLine("Деление 15 / 3 = " + calc.Divide(15, 3));
            Console.WriteLine("Деление 10 / 0 = " + calc.Divide(10, 0));
            Console.WriteLine("Степень 2^3 = " + calc.Power(2, 3));
            Console.WriteLine("Среднее [1,2,3,4,5] = " + calc.Average(1, 2, 3, 4, 5));
        }
    }
}
```

Вариант 2. Работа с массивами. Дан модуль для работы с массивами с ошибками. Необходимо найти и исправить все ошибки.

```
using System;

namespace ArrayModule
{
    public class ArrayProcessor
    {
        public int FindMax(int[] array)
        {
            int max = 0; // Неправильная инициализация
            for (int i = 0; i < array.Length; i++)
            {
                if (array[i] > max)
                    max = array[i];
            }
            return max;
        }

        public int FindMin(int[] array)
        {
            int min = array[0];
            for (int i = 1; i < array.Length; i++)
            {
                if (array[i] > min) // Невер-     ный знак сравнения
```

```
            min = array[i];
        }
        return min;
    }

    public double CalculateSum(int[] array)
    {
        int sum = 0;
        foreach (int num in array)
        {
            sum += num;
        }
        return sum;
    }

    public int[] ReverseArray(int[] array)
    {
        int[] reversed = new int[array.Length];
        for (int i = 0; i < array.Length; i++)
        {
            reversed[i] = array[array.Length - i]; // Выход за границы
        }
        return reversed;
    }

    public bool Contains(int[] array, int value)
    {
        for (int i = 0; i <= array.Length; i++) // Выход за границы
        {
            if (array[i] == value)
                return true;
        }
        return false;
    }

    public int CountEven(int[] array)
    {
        int count = 0;
        foreach (int num in array)
        {
            if (num % 2 == 1) // Неверное условие
                count++;
        }
        return count;
```

```
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ArrayProcessor processor = new ArrayProcessor();
            int[] testArray = { -5, 3, 8, -2, 10, 0, 7 };

            Console.WriteLine("Массив: " + string.Join(", ", testArray));
            Console.WriteLine("Максимум: " + processor.FindMax(testArray));
            Console.WriteLine("Минимум: " + processor.FindMin(testArray));
            Console.WriteLine("Сумма: " + processor.CalculateSum(testArray));
            Console.WriteLine("Перевернутый: " + string.Join(", ", proces-
sor.ReverseArray(testArray)));
            Console.WriteLine("Содержит 8: " + processor.Contains(testArray, 8));
            Console.WriteLine("Количество четных: " + proces-
sor.CountEven(testArray));
        }
    }
}
```

Вариант 3. Обработка строк. Дан модуль для обработки строк с ошибками. Необходимо найти и исправить все ошибки.

```
using System;

namespace StringModule
{
    public class StringProcessor
    {
        public int CountWords(string text)
        {
            if (text == null)
                return 0;

            string[] words = text.Split(' ');
            return words.Length;
        }

        public string ReverseString(string text)
        {
            char[] chars = text.ToCharArray();
            string reversed = "";
```

```
    for (int i = chars.Length; i >= 0; i--) // Выход за границы
    {
        reversed += chars[i];
    }
    return reversed;
}

public bool IsPalindrome(string text)
{
    text = text.ToLower().Replace(" ", "");
    int left = 0;
    int right = text.Length;

    while (left < right)
    {
        if (text[left] != text[right]) // Выход за границы
            return false;
        left++;
        right--;
    }
    return true;
}

public int CountVowels(string text)
{
    int count = 0;
    string vowels = "аеёиоуыэюяАЕЁИОУЫЭЮЯ";

    foreach (char c in text)
    {
        if (vowels.IndexOf(c) > 0) // Неверное условие
            count++;
    }
    return count;
}

public string RemoveSpaces(string text)
{
    string result = "";
    for (int i = 0; i < text.Length; i++)
    {
        if (text[i] == ' ')
            result += text[i]; // Логическая ошибка
    }
```

```
        return result;
    }

    public string Capitalize(string text)
    {
        if (string.IsNullOrEmpty(text))
            return text;

        return text[0].ToString().ToUpper() + text.Substring(1).ToUpper(); // Не-
верная логика
    }
}

class Program
{
    static void Main(string[] args)
    {
        StringProcessor processor = new StringProcessor();

        Console.WriteLine("Количество слов в 'Hello World': " +
            processor.CountWords("Hello World"));
        Console.WriteLine("Реверс 'Hello': " + processor.ReverseString("Hello"));
        Console.WriteLine("'шалаш' палиндром: " +
processor.IsPalindrome("шалаш"));
        Console.WriteLine("Гласных в 'программа': " +
            processor.CountVowels("программа"));
        Console.WriteLine("Удаление пробелов из 'a b c': " +
            processor.RemoveSpaces("a b c"));
        Console.WriteLine("Капитализация 'hello': " +
            processor.Capitalize("hello"));
    }
}
}
```

Вариант 4. Класс "Банковский счет". Дан модуль банковского счета с ошибками. Необходимо найти и исправить все ошибки**.**

```
using System;

namespace BankAccountModule
{
    public class BankAccount
    {
        private string accountNumber;
        private decimal balance;
```

```
    private string owner;

    public BankAccount(string accountNumber, string owner, decimal initial-
Balance)
    {
        this.accountNumber = accountNumber;
        this.owner = owner;
        this.balance = initialBalance;
    }

    public void Deposit(decimal amount)
    {
        if (amount < 0) // Неверная проверка
        {
            balance += amount;
            Console.WriteLine($"Зачислено: {amount}");
        }
        else
        {
            Console.WriteLine("Сумма должна быть положительной");
        }
    }

    public void Withdraw(decimal amount)
    {
        if (amount > 0 && amount <= balance)
        {
            balance += amount; // Неверная операция
            Console.WriteLine($"Снято: {amount}");
        }
        else
        {
            Console.WriteLine("Недостаточно средств или неверная сумма");
        }
    }

    public decimal GetBalance()
    {
        return balance;
    }

    public void Transfer(BankAccount targetAccount, decimal amount)
    {
        if (amount > 0 && amount <= balance)
```

```
        {
            this.balance -= amount;
            targetAccount.balance += amount;
            Console.WriteLine($"Переведено {amount} на счет {targetAc-
count.accountNumber}");
        }
        else
        {
            Console.WriteLine("Невозможно выполнить перевод");
        }
    }

    public void AddInterest(double rate)
    {
        decimal interest = balance * (decimal)rate / 100;
        balance += interest;
        Console.WriteLine($"Начислены проценты: {interest}");
    }

    public void DisplayInfo()
    {
        Console.WriteLine($"Счет: {accountNumber}");
        Console.WriteLine($"Владелец: {owner}");
        Console.WriteLine($"Баланс: {balance}"); // Отсутствует форматирова-
ние
    }
}

class Program
{
    static void Main(string[] args)
    {
        BankAccount account1 = new BankAccount("001", "Иванов И.И.", 1000);
        BankAccount account2 = new BankAccount("002", "Петров П.П.", 500);

        account1.DisplayInfo();
        account1.Deposit(500);
        account1.Withdraw(200);
        account1.Transfer(account2, 300);
        account1.AddInterest(5);
        account1.DisplayInfo();
        account2.DisplayInfo();
    }
}
```

```
}
```

Вариант 5. Класс "Студент". Дан модуль для работы со студентами с ошибками. Необходимо найти и исправить все ошибки.

```
using System;
using System.Collections.Generic;

namespace StudentModule
{
    public class Student
    {
        public string Name { get; set; }
        public string Group { get; set; }
        private List<int> grades;

        public Student(string name, string group)
        {
            Name = name;
            Group = group;
            grades = new List<int>();
        }

        public void AddGrade(int grade)
        {
            if (grade >= 2 || grade <= 5) // Неверное условие
            {
                grades.Add(grade);
                Console.WriteLine($"Оценка {grade} добавлена");
            }
            else
            {
                Console.WriteLine("Оценка должна быть от 2 до 5");
            }
        }

        public double GetAverageGrade()
        {
            if (grades.Count == 0)
                return 0;

            int sum = 0;
            foreach (int grade in grades)
            {
                sum += grade;
```

```
    }
    return sum / grades.Count; // Потеря точности
}

public int GetGradeCount()
{
    return grades.Count;
}

public bool IsExcellent()
{
    foreach (int grade in grades)
    {
        if (grade < 5)
            return true; // Неверная логика
    }
    return false;
}

public void RemoveLowestGrade()
{
    if (grades.Count > 0)
    {
        int min = grades[0];
        int minIndex = 0;

        for (int i = 1; i <= grades.Count; i++) // Выход за границы
        {
            if (grades[i] < min)
            {
                min = grades[i];
                minIndex = i;
            }
        }

        grades.RemoveAt(minIndex);
        Console.WriteLine($"Удалена оценка {min}");
    }
}

public void DisplayInfo()
{
    Console.WriteLine($"Студент: {Name}");
    Console.WriteLine($"Группа: {Group}");
```

```
            Console.WriteLine($"Оценки: {string.Join(", ", grades)}");
            Console.WriteLine($"Средний балл: {GetAverageGrade()}");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Student student = new Student("Сидоров С.С.", "ПО-21");

            student.AddGrade(5);
            student.AddGrade(4);
            student.AddGrade(5);
            student.AddGrade(3);
            student.AddGrade(5);

            student.DisplayInfo();
            Console.WriteLine($"Отличник: {student.IsExcellent()}");
            student.RemoveLowestGrade();
            student.DisplayInfo();
        }
    }
}
```

Вариант 6. Класс "Книга" и "Библиотека". Дан модуль библиотеки с ошибками. Необходимо найти и исправить все ошибки.

```
using System;
using System.Collections.Generic;

namespace LibraryModule
{
    public class Book
    {
        public string Title { get; set; }
        public string Author { get; set; }
        public int Year { get; set; }
        public bool IsAvailable { get; set; }

        public Book(string title, string author, int year)
        {
            Title = title;
            Author = author;
            Year = year;
```

```
        IsAvailable = true;
    }
}

public class Library
{
    private List<Book> books;

    public Library()
    {
        books = null; // Ошибка инициализации
    }

    public void AddBook(Book book)
    {
        books.Add(book);
        Console.WriteLine($"Книга '{book.Title}' добавлена");
    }

    public Book FindBookByTitle(string title)
    {
        for (int i = 0; i < books.Count; i++)
        {
            if (books[i].Title = title) // Ошибка сравнения
                return books[i];
        }
        return null;
    }

    public List<Book> FindBooksByAuthor(string author)
    {
        List<Book> result = new List<Book>();
        foreach (Book book in books)
        {
            if (book.Author.ToLower() == author.ToLower())
                result.Add(book);
        }
        return result;
    }

    public void BorrowBook(string title)
    {
        Book book = FindBookByTitle(title);
        if (book != null || book.IsAvailable) // Неверное условие
```

```
        {
            book.IsAvailable = false;
            Console.WriteLine($"Книга '{title}' выдана");
        }
        else
        {
            Console.WriteLine($"Книга '{title}' недоступна");
        }
    }

    public void ReturnBook(string title)
    {
        Book book = FindBookByTitle(title);
        if (book != null)
        {
            book.IsAvailable = true;
            Console.WriteLine($"Книга '{title}' возвращена");
        }
    }

    public int GetAvailableBooksCount()
    {
        int count = 0;
        foreach (Book book in books)
        {
            if (!book.IsAvailable) // Неверная логика
                count++;
        }
        return count;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Library library = new Library();

        library.AddBook(new Book("Война и мир", "Толстой Л.Н.", 1869));
        library.AddBook(new Book("Преступление и наказание", "Достоевский
Ф.М.", 1866));
        library.AddBook(new Book("Анна Каренина", "Толстой Л.Н.", 1877));

        Console.WriteLine($"Доступно книг: {li-
```

```
brary.GetAvailableBooksCount()}");
            library.BorrowBook("Война и мир");
            Console.WriteLine($"Доступно книг: {li-
brary.GetAvailableBooksCount()}");
            library.ReturnBook("Война и мир");
        }
    }
}
```

Вариант 7. Класс "Дата". Дан модуль для работы с датами с ошибками. Необходимо найти и исправить все ошибки.

```
using System;

namespace DateModule
{
    public class CustomDate
    {
        private int day;
        private int month;
        private int year;

        public CustomDate(int day, int month, int year)
        {
            this.day = day;
            this.month = month;
            this.year = year;
        }

        public bool IsLeapYear()
        {
            if (year % 4 == 0)
                return true;
            else
                return false; // Неполная проверка високосного года
        }

        public int GetDaysInMonth()
        {
            int[] daysInMonth = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

            if (month == 2 && IsLeapYear())
                return 29;

            return daysInMonth[month]; // Выход за границы массива
        }
```

```
public bool IsValidDate()
{
    if (year < 1 && year > 9999) // Неверное условие
        return false;

    if (month < 1 || month > 12)
        return false;

    if (day < 1 || day > GetDaysInMonth())
        return false;

    return true;
}

public void AddDays(int days)
{
    day += days;

    while (day > GetDaysInMonth())
    {
        day -= GetDaysInMonth();
        month++;

        if (month > 12)
        {
            month = 1;
            year--;  // Неверная операция
        }
    }
}

public int DayOfYear()
{
    int dayCount = 0;

    for (int m = 1; m <= month; m++) // Неверное условие
    {
        CustomDate tempDate = new CustomDate(1, m, year);
        dayCount += tempDate.GetDaysInMonth();
    }

    dayCount += day;
    return dayCount;
```

```
    }

    public override string ToString()
    {
        return $"{day}/{month}/{year}";
    }
}

class Program
{
    static void Main(string[] args)
    {
        CustomDate date = new CustomDate(28, 2, 2024);

        Console.WriteLine($"Дата: {date}");
        Console.WriteLine($"Високосный год: {date.IsLeapYear()}");
        Console.WriteLine($"Дней в месяце: {date.GetDaysInMonth()}");
        Console.WriteLine($"Корректная дата: {date.IsValidDate()}");
        Console.WriteLine($"День года: {date.DayOfYear()}");

        date.AddDays(5);
        Console.WriteLine($"После добавления 5 дней: {date}");
    }
}
}
```

Вариант 8. Класс "Товар" и "Корзина". Дан модуль интернет-магазина с ошибками. Необходимо найти и исправить все ошибки.

```
using System;
using System.Collections.Generic;

namespace ShoppingModule
{
    public class Product
    {
        public string Name { get; set; }
        public decimal Price { get; set; }
        public int Quantity { get; set; }

        public Product(string name, decimal price, int quantity)
        {
            Name = name;
            Price = price;
```

```
        Quantity = quantity;
    }

    public decimal GetTotalPrice()
    {
        return Price + Quantity; // Неверная операция
    }
}

public class ShoppingCart
{
    private List<Product> products;

    public ShoppingCart()
    {
        products = new List<Product>();
    }

    public void AddProduct(string name, decimal price, int quantity)
    {
        if (quantity < 0) // Неверное условие
        {
            Product product = new Product(name, price, quantity);
            products.Add(product);
            Console.WriteLine($"Товар '{name}' добавлен в корзину");
        }
        else
        {
            Console.WriteLine("Количество должно быть положительным");
        }
    }

    public void RemoveProduct(string name)
    {
        for (int i = 0; i < products.Count; i++)
        {
            if (products[i].Name == name)
            {
                products.Remove(products[i]);
                i++; // Ошибка при удалении элемента
                Console.WriteLine($"Товар '{name}' удален из корзины");
            }
        }
    }
```

```
    public decimal GetTotalCost()
    {
        decimal total = 0;
        foreach (Product product in products)
        {
            total += product.GetTotalPrice();
        }
        return total;
    }

    public void ApplyDiscount(decimal discountPercent)
    {
        foreach (Product product in products)
        {
            product.Price = product.Price * discountPercent / 100; // Неверная логи-
ка
        }
        Console.WriteLine($"Применена скидка {discountPercent}%");
    }

    public Product FindMostExpensive()
    {
        if (products.Count == 0)
            return null;

        Product maxProduct = products[0];
        for (int i = 1; i < products.Count; i++)
        {
            if (products[i].GetTotalPrice() < maxProduct.GetTotalPrice()) // Невер-
ное сравнение
                maxProduct = products[i];
        }
        return maxProduct;
    }

    public void DisplayCart()
    {
        Console.WriteLine("\n=== Корзина ===");
        foreach (Product product in products)
        {
            Console.WriteLine($"{product.Name} - {product.Price} руб. x {prod-
uct.Quantity} = {product.GetTotalPrice()} руб.");
        }
```

```
        Console.WriteLine($"Итого: {GetTotalCost()} руб.\n");
    }
}

class Program
{
    static void Main(string[] args)
    {
        ShoppingCart cart = new ShoppingCart();

        cart.AddProduct("Ноутбук", 50000, 1);
        cart.AddProduct("Мышь", 500, 2);
        cart.AddProduct("Клавиатура", 1500, 1);

        cart.DisplayCart();
        cart.ApplyDiscount(10);
        cart.DisplayCart();

        Product expensive = cart.FindMostExpensive();
        if (expensive != null)
            Console.WriteLine($"Самый дорогой товар: {expensive.Name}");
    }
}
}
```

Вариант 9. Класс "Точка" и "Геометрические фигуры". Дан модуль геометрии с ошибками. Необходимо найти и исправить все ошибки.

```
using System;

namespace GeometryModule
{
    public class Point
    {
        public double X { get; set; }
        public double Y { get; set; }

        public Point(double x, double y)
        {
            X = x;
            Y = y;
        }

        public double DistanceTo(Point other)
        {
```

```
        double dx = X - other.X;
        double dy = Y - other.Y;
        return Math.Sqrt(dx + dy); // Неверная формула
    }
}

public class Rectangle
{
    private Point topLeft;
    private Point bottomRight;

    public Rectangle(Point topLeft, Point bottomRight)
    {
        this.topLeft = topLeft;
        this.bottomRight = bottomRight;
    }

    public double GetWidth()
    {
        return bottomRight.X - topLeft.X;
    }

    public double GetHeight()
    {
        return bottomRight.Y - topLeft.Y;
    }

    public double GetArea()
    {
        return GetWidth() + GetHeight(); // Неверная формула
    }

    public double GetPerimeter()
    {
        return 2 * GetWidth() * GetHeight(); // Неверная формула
    }

    public bool Contains(Point point)
    {
        return point.X > topLeft.X && point.X < bottomRight.X &&
               point.Y > topLeft.Y && point.Y < bottomRight.Y; // Неверное усло-
вие для координат Y
    }
}
```

```
public class Circle
{
    private Point center;
    private double radius;

    public Circle(Point center, double radius)
    {
        this.center = center;
        this.radius = radius;
    }

    public double GetArea()
    {
        return Math.PI * radius; // Неверная формула
    }

    public double GetCircumference()
    {
        return 2 * Math.PI * radius;
    }

    public bool Contains(Point point)
    {
        return center.DistanceTo(point) < radius;
    }

    public bool Intersects(Circle other)
    {
        double distance = center.DistanceTo(other.center);
        return distance > (radius + other.radius); // Неверное условие
    }
}

class Program
{
    static void Main(string[] args)
    {
        Point p1 = new Point(0, 0);
        Point p2 = new Point(3, 4);

        Console.WriteLine($"Расстояние между точками: {p1.DistanceTo(p2)}");

        Rectangle rect = new Rectangle(new Point(0, 5), new Point(10, 0));
```

```
        Console.WriteLine($"Площадь прямоугольника: {rect.GetArea()}");
        Console.WriteLine($"Периметр прямоугольника: {rect.GetPerimeter()}");
        Console.WriteLine($"Содержит точку (5, 2): {rect.Contains(new Point(5,
2))}");

        Circle circle = new Circle(new Point(0, 0), 5);
        Console.WriteLine($"Площадь круга: {circle.GetArea()}");
        Console.WriteLine($"Длина окружности: {circle.GetCircumference()}");
    }
  }
}
```

Вариант 10. Класс "Список задач" (To-Do List). Дан модуль списка задач с ошибками. Необходимо найти и исправить все ошибки.

```
using System;
using System.Collections.Generic;

namespace TodoModule
{
  public class Task
  {
    public string Title { get; set; }
    public string Description { get; set; }
    public bool IsCompleted { get; set; }
    public DateTime DueDate { get; set; }

    public Task(string title, string description, DateTime dueDate)
    {
      Title = title;
      Description = description;
      IsCompleted = false;
      DueDate = dueDate;
    }

    public bool IsOverdue()
    {
      return DateTime.Now < DueDate && !IsCompleted; // Неверное условие
    }
  }

  public class TodoList
  {
    private List<Task> tasks;
```

```
public TodoList()
{
    tasks = new List<Task>();
}

public void AddTask(string title, string description, DateTime dueDate)
{
    Task task = new Task(title, description, dueDate);
    tasks.Add(task);
    Console.WriteLine($"Задача '{title}' добавлена");
}

public void CompleteTask(string title)
{
    foreach (Task task in tasks)
    {
        if (task.Title = title) // Ошибка присваивания вместо сравнения
        {
            task.IsCompleted = true;
            Console.WriteLine($"Задача '{title}' выполнена");
            return;
        }
    }
    Console.WriteLine($"Задача '{title}' не найдена");
}

public void RemoveTask(string title)
{
    for (int i = 0; i < tasks.Count; i--)  // Неверное изменение счетчика
    {
        if (tasks[i].Title == title)
        {
            tasks.RemoveAt(i);
            Console.WriteLine($"Задача '{title}' удалена");
            return;
        }
    }
}

public List<Task> GetOverdueTasks()
{
    List<Task> overdue = new List<Task>();
    foreach (Task task in tasks)
    {
```

```
            if (task.IsOverdue())
                overdue.Add(task);
        }
        return overdue;
    }

    public int GetCompletedCount()
    {
        int count = 0;
        foreach (Task task in tasks)
        {
            if (task.IsCompleted)
                count--;  // Неверная операция
        }
        return count;
    }

    public void DisplayTasks()
    {
        Console.WriteLine("\n=== Список задач ===");
        for (int i = 1; i <= tasks.Count; i++)
        {
            Task task = tasks[i]; // Выход за границы
            string status = task.IsCompleted ? "[✓]" : "[ ]";
            Console.WriteLine($"{status} {task.Title} - до
{task.DueDate.ToShortDateString()}");
        }
        Console.WriteLine($"Выполнено: {GetCompleted-
Count()}/{tasks.Count}\n");
    }
}

class Program
{
    static void Main(string[] args)
    {
        TodoList todoList = new TodoList();

        todoList.AddTask("Подготовить отчет", "Квартальный отчет",
DateTime.Now.AddDays(7));
        todoList.AddTask("Встреча с клиентом", "Обсудить требования",
DateTime.Now.AddDays(-2));
        todoList.AddTask("Код-ревью", "Проверить pull request",
DateTime.Now.AddDays(1));
```

```
            todoList.DisplayTasks();
            todoList.CompleteTask("Код-ревью");
            todoList.DisplayTasks();

            Console.WriteLine($"Просроченных задач: {todo-
List.GetOverdueTasks().Count}");
        }
    }
}
```

Требования к выполнению задания

1.Создайте новый проект в Visual Studio 2022 типа Console Application (.NET Framework или .NET Core) с именем, соответствующим вашему варианту.

2.Скопируйте код из вашего варианта задания в проект.

3.Выполните первоначальный запуск программы без отладки и зафиксируйте результаты работы (ошибки, некорректные значения).

4.Проведите отладку программы, используя инструменты Visual Studio: – Установите точки останова на ключевых строках кода; – Используйте пошаговое выполнение (F10, F11); – Анализируйте значения переменных через окна Locals, Autos, Watch; – Просматривайте стек вызовов при возникновении исключений; – Используйте окно Immediate для проверки выражений

5.Найдите все ошибки в коде (каждый вариант содержит 6-8 ошибок различных типов).

6.Исправьте обнаруженные ошибки и протестируйте корректность работы программы.

7.Добавьте в код вывод отладочной информации с использованием класса Debug для контроля ключевых операций.

8.Документируйте процесс отладки: для каждой найденной ошибки укажите: Описание ошибки; Симптомы (как проявлялась ошибка); Способ обнаружения (какой инструмент отладки использовался); Исправление (что было изменено в коде)

Содержание отчета. Отчет должен содержать:

1.Исходный код программы с ошибками.

2.Результаты первоначального запуска программы (скриншоты, описание поведения).

3.Процесс отладки: Таблицу с описанием всех найденных ошибок по следующей структуре:

№	Строка кода	Описание ошибки	Симптомы	Инструмент отладки	Исправление
1					
2					

Скриншоты процесса отладки (окна Locals, Watch, точки останова, стек вызовов)

4.Исправленный код программы с комментариями к исправлениям.

5.Результаты работы исправленной программы (скриншоты корректного выполнения).

6.Код с добавленными отладочными сообщениями (использование Debug.WriteLine).

7.Выводы по работе

Контрольные вопросы

1.Что такое отладка программного обеспечения? Какие типы ошибок существуют в программировании?

2.Какие основные инструменты отладки предоставляет Visual Studio 2022? Опишите назначение окон Locals, Autos, Watch, Call Stack.

3.Какая разница между командами Step Over (F10) и Step Into (F11)? В каких случаях следует использовать каждую из них?

4.Что такое точка останова (breakpoint)? Какие типы точек останова существуют и когда их следует применять?

5.Как использовать условные точки останова? Приведите примеры ситуаций, когда они необходимы.

6.Что такое стек вызовов и как он помогает в процессе отладки?

7.В чем отличие между классом Debug и классом Console при выводе диагностической информации?

8.Что происходит с кодом, использующим Debug.WriteLine(), при компиляции в режиме Release?

9.Какие стратегии отладки существуют? Опишите метод

"разделяй и властвуй" применительно к отладке программ.

10.Почему важно использовать отладку при интеграции программных модулей? Какие специфические проблемы могут возникать на стыке модулей?

11.Как можно отследить изменение значения переменной на протяжении выполнения программы без установки множества точек останова?

12.Что такое окно Immediate и как его можно использовать в процессе отладки?

13.Как обрабатывать исключения во время отладки? Как настроить Visual Studio для остановки выполнения при возникновении определённого типа исключения?

14.Какие лучшие практики отладки вы можете назвать? Как предотвратить появление ошибок на этапе написания кода?

15.Как документировать найденные и исправленные ошибки для последующего использования командой разработчиков?

Время выполнения: 2 ч

Практическая работа № 7 «Организация обработки исключений»

Цель работы: Изучить механизмы обработки исключений в языке программирования C#, освоить структуру блоков try-catch-finally, научиться создавать пользовательские классы исключений и применять лучшие практики обработки ошибок при интеграции программных модулей.

Теоретическая часть

Понятие исключения. Исключение (Exception) — это объект, который создается при возникновении ошибки во время выполнения программы. Механизм исключений позволяет передавать управление из точки возникновения ошибки в точку её обработки, разделяя логику нормального выполнения программы и логику обработки ошибок**.**

Иерархия исключений в C#. Все исключения в C# наследуются от базового класса System.Exception. Основные категории исключений**:**

SystemException — базовый класс для исключений времени выполнения

– NullReferenceException — обращение к null-ссылке
– IndexOutOfRangeException — выход за границы массива
– DivideByZeroException — деление на ноль
– InvalidOperationException — недопустимая операция
– ArgumentException — некорректный аргумент метода
– FormatException — ошибка формата данных
– IOException — ошибки ввода-вывода

Структура обработки исключений

Базовый синтаксис try-catch

```
try
{
   // Код, который может вызвать исключение
}
catch (ТипИсключения переменная)
{
   // Обработка исключения
}
```

Множественная обработка исключений

```
try
{
    // Потенциально опасный код
}
catch (DivideByZeroException ex)
{
    // Обработка деления на ноль
    Console.WriteLine($"Ошибка: {ex.Message}");
}
catch (FormatException ex)
{
    // Обработка ошибки формата
    Console.WriteLine($"Неверный формат: {ex.Message}");
}
catch (Exception ex)
{
    // Обработка всех остальных исключений
    Console.WriteLine($"Общая ошибка: {ex.Message}");
}
```

Важно: Более специфичные исключения должны обрабатываться раньше общих.

Блок finally. Блок finally выполняется всегда, независимо от того, возникло исключение или нет. Используется для освобождения ресурсов.

```
FileStream file = null;
try
{
    file = new FileStream("data.txt", FileMode.Open);
    // Работа с файлом
}
catch (FileNotFoundException ex)
{
    Console.WriteLine("Файл не найден");
}
finally
{
    // Освобождение ресурсов
    if (file != null)
        file.Close();
}
```

Конструкция using для автоматического освобождения ресурсов

```
using (FileStream file = new FileStream("data.txt", FileMode.Open))
```

```
{
   // Работа с файлом
   // Файл автоматически закроется при выходе из блока
}
```

Генерация исключений (throw). Оператор throw используется для явной генерации исключения:

```
public void SetAge(int age)
{
   if (age < 0 || age > 150)
   {
      throw new ArgumentException("Возраст должен быть от 0 до 150");
   }
   this.age = age;
}
```

Создание пользовательских исключений. Для создания собственных типов исключений необходимо наследоваться от класса Exception:

```
public class ValidationException : Exception
{
   public ValidationException() : base() { }

   public ValidationException(string message) : base(message) { }

   public ValidationException(string message, Exception innerException)
      : base(message, innerException) { }
}
```

Использование:

```
if (value < 0)
{
   throw new ValidationException("Значение не может быть отрицательным");
}
```

Свойства класса Exception

– *Message* — описание ошибки
– *StackTrace* — трассировка стека вызовов
– *Source* — имя приложения или объекта, вызвавшего ошибку
– *InnerException* — внутреннее исключение, вызвавшее текущее
– *HelpLink* — ссылка на справочную информацию

Фильтры исключений (Exception Filters)

C# позволяет добавлять условия к блокам catch:

```
try
{
   // код
```

```
}
catch (HttpException ex) when (ex.StatusCode == 404)
{
    // Обработка только ошибок 404
}
catch (HttpException ex) when (ex.StatusCode == 500)
{
    // Обработка только ошибок 500
}
```

Лучшие практики обработки исключений

1.Обрабатывайте только те исключения, которые можете корректно обработать
2.Не используйте пустые блоки catch — это скрывает ошибки
3.Не используйте исключения для управления потоком выполнения — они предназначены для исключительных ситуаций
4.Предоставляйте информативные сообщения об ошибках
5.Освобождайте ресурсы в блоке finally или используйте using
6.Логируйте исключения для последующего анализа
7.Создавайте пользовательские исключения для специфичных бизнес-логики ошибок

Обработка исключений при интеграции модулей. При интеграции программных модулей обработка исключений особенно важна:

- Валидация входных данных от внешних модулей
- Обработка ошибок сетевого взаимодействия
- Контроль доступа к ресурсам (файлы, БД)
- Обработка ошибок парсинга данных (JSON, XML)
- Транзакционная целостность при работе с БД
- Логирование ошибок интеграции для диагностики

Рабочее задание: Выполните разработку программного модуля с организацией обработки исключений в соответствии с вариантом задания.

Вариант 1. Калькулятор с обработкой исключений. Разработать консольное приложение-калькулятор, выполняющее четыре арифметических операции (+, -, *, /). Программа должна:

- Запрашивать у пользователя два числа и операцию
- Обрабатывать исключения при вводе некорректных данных (FormatException)

- Обрабатывать деление на ноль (DivideByZeroException)
- Обрабатывать переполнение при вычислениях (OverflowException)
- Предоставлять возможность повторного выполнения операций
- Логировать все возникшие ошибки в текстовый файл с указанием даты и времени

Вариант 2. Работа с файлами и обработка исключений. Разработать программу для чтения данных из текстового файла и записи обработанных данных в другой файл. Программа должна:

- Читать числа из входного файла (каждое число на отдельной строке)
- Вычислять квадратный корень каждого числа
- Обрабатывать отсутствие входного файла (FileNotFoundException)
- Обрабатывать ошибки доступа к файлам (UnauthorizedAccessException)
- Обрабатывать некорректный формат данных (FormatException)
- Обрабатывать отрицательные числа (создать пользовательское исключение NegativeNumberException)
- Гарантировать закрытие файловых потоков через finally или using
- Записывать результаты в выходной файл, а ошибочные строки — в файл лога ошибок

Вариант 3. Система регистрации пользователей с валидацией. Разработать программу регистрации пользователей с валидацией данных. Программа должна:

- Запрашивать: имя пользователя, email, возраст, пароль
- Создать пользовательские исключения:

– InvalidUsernameException — имя короче 3 символов или содержит недопустимые символы
– InvalidEmailException — некорректный формат email
– InvalidAgeException — возраст меньше 18 или больше 100
– WeakPasswordException — пароль короче 8 символов

- Обрабатывать все пользовательские исключения с выво-

дом конкретных сообщений
- Давать возможность повторного ввода при ошибке
- Сохранять успешно зарегистрированных пользователей в файл
- Логировать все попытки регистрации (успешные и не-успешные)

Вариант 4. Работа с массивами и обработка исключений.
Разработать программу для работы с массивом целых чисел. Программа должна:
- Создавать массив заданного размера (размер вводит пользователь)
- Заполнять массив случайными числами от -100 до 100
- Реализовать методы:

– Получение элемента по индексу
– Вычисление среднего арифметического
– Поиск максимального и минимального элемента
– Деление всех элементов на заданное число
- Обрабатывать исключения:

– IndexOutOfRangeException — при некорректном индексе
– DivideByZeroException — при делении на ноль
– OverflowException — при переполнении
– ArgumentException — при некорректном размере массива
- Использовать множественные блоки catch для разных типов исключений
- Реализовать метод безопасного доступа к элементам массива

Вариант 5. Парсинг данных с обработкой исключений.
Разработать программу для парсинга CSV-файла с данными о студентах. Формат: Фамилия, Имя, Группа, Средний_балл. Программа должна:
- Читать данные из CSV-файла
- Парсить каждую строку и создавать объекты класса Student
- Обрабатывать исключения:

– FileNotFoundException — файл не найден
– FormatException — некорректный формат данных
- Создать InvalidDataException для строк с неполными данными

– Создать InvalidGradeException для среднего балла вне диапазона 2.0-5.0

- Пропускать ошибочные строки с записью в лог
- Выводить статистику: сколько записей обработано успешно, сколько с ошибками
- Сохранять корректные данные в новый файл
- Вычислять средний балл по всем студентам

Вариант 6. Конвертер валют с обработкой исключений.
Разработать программу-конвертер валют. Программа должна:

- Хранить курсы валют в словаре (USD, EUR, GBP к RUB)
- Запрашивать у пользователя: сумму, исходную валюту, целевую валюту
- Выполнять конвертацию валют
- Обрабатывать исключения:

– FormatException — некорректный ввод суммы
– Создать CurrencyNotFoundException — валюта не найдена
– Создать NegativeAmountException — отрицательная сумма
– OverflowException — переполнение при конвертации

- Загружать курсы валют из текстового файла при запуске
- Обрабатывать ошибки чтения файла с курсами
- Предоставлять значения по умолчанию при ошибках загрузки
- Логировать все операции конвертации

Вариант 7. База данных книг с обработкой исключений.
Разработать программу для управления коллекцией книг. Программа должна:

- Использовать класс Book (название, автор, год издания, ISBN)
- Хранить книги в списке List
- Реализовать операции: добавление, поиск, удаление книги
- Обрабатывать исключения:

– Создать DuplicateISBNException — книга с таким ISBN уже существует
– Создать BookNotFoundException — книга не найдена
– Создать InvalidYearException — год издания вне допустимого диапазона (1450-текущий год)

– ArgumentNullException — пустые обязательные поля

- Сохранять коллекцию в файл при выходе
- Загружать коллекцию из файла при запуске
- Обрабатывать ошибки сериализации/десериализации
- Создавать резервную копию при критических ошибках

Вариант 8. Система бронирования с обработкой исключений. Разработать программу бронирования мест в кинотеатре. Программа должна:

- Создать зал с сеткой мест (например, 5 рядов по 10 мест)
- Реализовать функции: просмотр свободных мест, бронирование, отмена бронирования
- Обрабатывать исключения:

– Создать SeatAlreadyBookedException — место уже забронировано
– Создать SeatNotFoundException — место не существует
– Создать InvalidRowException — некорректный номер ряда
– Создать InvalidSeatException — некорректный номер места

- Сохранять состояние бронирований в файл
- Обрабатывать ошибки загрузки/сохранения данных
- Реализовать откат изменений при ошибках
- Вести лог всех операций бронирования

Вариант 9. Обработка математических выражений с исключениями. Разработать программу для вычисления математических выражений. Программа должна:

- Принимать строку с математическим выражением (например, "2 + 3 * 4")
- Поддерживать операции: +, -, *, /, (), возведение в степень (^)
- Обрабатывать исключения:

– Создать InvalidExpressionException — некорректное выражение
– Создать UnbalancedParenthesesException — несбалансированные скобки
– DivideByZeroException — деление на ноль
– OverflowException — переполнение результата
– FormatException — некорректный формат чисел

- Парсить выражение с обработкой всех ошибок

- Выводить подробное описание ошибки с указанием позиции в строке
- Предоставлять возможность ввода нового выражения при ошибке
- Сохранять историю вычислений в файл

Вариант 10. Загрузка и обработка конфигурационных файлов. Разработать программу для работы с конфигурационными файлами в формате INI. Программа должна:

- Читать конфигурационный файл формата:
- [Database]
- Server=localhost
- Port=5432
- Username=admin
- [Application]
- Timeout=30
- MaxConnections=100
- Парсить секции и параметры
- Обрабатывать исключения:

– FileNotFoundException — файл конфигурации не найден
– Создать InvalidConfigFormatException — некорректный формат файла
– Создать MissingSectionException — отсутствует обязательная секция
– Создать MissingParameterException — отсутствует обязательный параметр
– FormatException — некорректный тип данных параметра

- Создавать файл конфигурации по умолчанию при его отсутствии
- Валидировать значения параметров (например, Port должен быть в диапазоне 1-65535)
- Реализовать сохранение изменений в конфигурационный файл
- Вести лог всех операций с конфигурацией

Требования к выполнению задания

1.Реализовать все указанные в варианте функции с корректной обработкой исключений
2.Использовать множественные блоки catch для обработки раз-

личных типов исключений
3.Создать не менее двух пользовательских классов исключений (для вариантов, где это требуется)
4.Применить блок finally или конструкцию using для освобождения ресурсов (где применимо)
5.Реализовать информативные сообщения об ошибках для пользователя
6.Логировать исключения в файл (с указанием типа исключения, сообщения, даты и времени)
7.Обеспечить возможность продолжения работы программы после обработки исключения
8.Протестировать программу на различных сценариях, включая некорректные входные данные
9.Добавить комментарии к коду, поясняющие логику обработки исключений

Содержание отчета. Отчет должен содержать:
1.Теоретическая часть: Краткое описание механизма обработки исключений в C#; Описание используемых типов исключений (стандартных и пользовательских); Обоснование выбора подходов к обработке ошибок
2.Практическая часть: Описание разработанной программы; UML-диаграмма классов (при наличии пользовательских классов исключений); Полный листинг программного кода с комментариями; Скриншоты работы программы (нормальное выполнение и различные сценарии ошибок); Примеры содержимого файлов (входных данных, выходных данных, логов)
3.Тестирование: Таблица тестовых сценариев:

№	Описание сценария	Входные данные	Ожидаемый результат	Фактический результат	Статус
1	Корректные данные	...	...	...	✓
2	Некорректный формат	...	FormatException	...	✓
...	...	...	...	...	...

4.Выводы:

Контрольные вопросы

1.Что такое исключение в языке программирования C#? В чем разница между исключением и ошибкой компиляции?

2.Какова иерархия классов исключений в .NET? Назовите базовый класс для всех исключений.

3.Объясните назначение блоков try, catch и finally. Обязательно ли наличие всех трех блоков?

4.В каком порядке должны располагаться блоки catch при обработке нескольких типов исключений? Почему?

5.Когда и для чего используется оператор throw? Приведите примеры.

6.Как создать пользовательский класс исключения? Какие конструкторы рекомендуется реализовывать?

7.Объясните назначение конструкции using. Какие объекты можно использовать в блоке using?

8.Перечислите основные свойства класса Exception. Для чего используется свойство InnerException?

9.Что такое фильтры исключений (exception filters) в C#? Приведите пример использования.

10.Почему не рекомендуется использовать пустые блоки catch? К каким проблемам это может привести?

11.В каких ситуациях следует использовать исключения, а в каких — проверку условий через if?

12.Какие лучшие практики обработки исключений следует применять при интеграции программных модулей?

13.Объясните разницу между обработкой исключения (catch) и повторной генерацией (throw). Когда используется каждый подход?

14.Как гарантировать освобождение ресурсов (закрытие файлов, соединений с БД) при возникновении исключений?

15.Какую информацию следует включать в лог при регистрации исключения для эффективной диагностики проблем?

Время выполнения: 2 ч

Практическая работа № 8 «Применение отладочных классов в проекте»

Цель работы: Изучить назначение и возможности отладочных классов в .NET Framework, освоить практические навыки применения классов Debug и Trace для отладки и мониторинга программных модулей на языке C# в среде Visual Studio 2022.

Теоретическая часть

Отладочные классы в .NET Framework. Отладочные классы — это специальные программные средства, предназначенные для диагностики, отладки и мониторинга работы приложений на этапах разработки и тестирования. В платформе .NET Framework основными отладочными классами являются **Debug** и **Trace**, расположенные в пространстве имен **System.Diagnostics**.

Класс Debug. Класс **Debug** используется для вывода отладочной информации и проверки условий исключительно в отладочной (Debug) конфигурации проекта. Код, использующий методы класса Debug, автоматически исключается из релизной (Release) сборки приложения благодаря директиве условной компиляции.

Основные методы класса Debug:

– **Debug.Write()** — выводит текст в окно вывода без перехода на новую строку

– **Debug.WriteLine()** — выводит текст с переходом на новую строку

– **Debug.WriteIf()** — выводит текст при истинности условия

– **Debug.WriteLineIf()** — выводит текст с новой строки при истинности условия

– **Debug.Assert()** — проверяет условие и выводит сообщение, если условие ложно

– **Debug.Print()** — форматированный вывод (аналог WriteLine)

– **Debug.Indent()** — увеличивает отступ для вывода

– **Debug.Unindent()** — уменьшает отступ

– **Debug.Flush()** — принудительно записывает все буферизованные данные

Класс Trace. Класс **Trace** предоставляет аналогичную функциональность, но работает как в Debug, так и в Release конфигурациях. Это позволяет вести логирование и мониторинг приложе-

ния в продуктивной среде.

Основные методы класса Trace:

- **Trace.Write()** — выводит текст
- **Trace.WriteLine()** — выводит текст с новой строки
- **Trace.WriteIf()** — условный вывод
- **Trace.WriteLineIf()** — условный вывод с новой строки
- **Trace.Assert()** — проверка утверждений
- **Trace.TraceInformation()** — вывод информационного сообщения
- **Trace.TraceWarning()** — вывод предупреждения
- **Trace.TraceError()** — вывод сообщения об ошибке
- **Trace.Indent() / Trace.Unindent()** — управление отступами
- **Trace.Flush()** — очистка буфера

Прослушиватели (Listeners). Отладочные классы используют механизм **прослушивателей** (Listeners) для определения, куда направлять выходные данные. По умолчанию вывод направляется в окно Output в Visual Studio.

Основные типы прослушивателей:

- **DefaultTraceListener** — вывод в окно Output IDE и OutputDebugString API
- **TextWriterTraceListener** — запись в текстовый файл или поток
- **EventLogTraceListener** — запись в журнал событий Windows
- **ConsoleTraceListener** — вывод в консоль
- **XmlWriterTraceListener** — запись в XML-файл

Пример добавления прослушивателя:

```
Trace.Listeners.Add(new TextWriterTraceListener("log.txt"));
Trace.AutoFlush = true; // автоматическая запись в файл
```

Условная компиляция. Для управления включением отладочного кода используются директивы условной компиляции:

```
#if DEBUG
    Debug.WriteLine("Это сообщение появится только в Debug-сборке");
#endif
```

Символы **DEBUG** и **TRACE** определяются автоматически в за-

висимости от конфигурации проекта в свойствах сборки.
Проверка утверждений (Assertions). Метод **Assert()** позволяет проверять критические условия в программе:

```
Debug.Assert(value > 0, "Значение должно быть положительным",
    $"Получено значение: {value}");
```

Если условие ложно, выводится диалоговое окно с сообщением об ошибке и информацией для отладки.
Преимущества использования отладочных классов

- Не требуют установки точек останова
- Позволяют отслеживать последовательность выполнения программы
- Обеспечивают вывод значений переменных в критических точках
- Помогают выявлять логические ошибки
- Упрощают отладку многопоточных приложений
- Позволяют вести журналирование работы приложения
- Автоматически удаляются из Release-сборки (Debug) или остаются для мониторинга (Trace)

Окно вывода (Output Window) в Visual Studio. Для просмотра отладочных сообщений в Visual Studio используется окно **Output** (Вывод), которое можно открыть через меню **View → Output** или сочетанием клавиш **Ctrl+Alt+O**. В выпадающем списке необходимо выбрать источник **Debug.**

Рабочее задание: Выполните разработку программного модуля в соответствии с вариантом задания. Примените отладочные классы Debug и Trace для контроля выполнения программы, вывода промежуточных результатов и проверки корректности работы алгоритмов. Настройте запись отладочной информации в текстовый файл.

Вариант 1. Калькулятор с историей операций. Разработать консольное приложение-калькулятор, выполняющее базовые арифметические операции (сложение, вычитание, умножение, деление). Использовать Debug для вывода каждой операции и ее результата. Использовать Trace для записи истории всех вычислений в файл. Применить Debug.Assert() для проверки деления на ноль.

Вариант 2. Система управления списком студентов. Разработать приложение для управления списком студентов (добавление, удаление, поиск, вывод всех). Использовать Debug.WriteLine() для отслеживания каждой операции над коллекцией. Применить Trace для логирования всех изменений в файл. Использовать Debug.Assert() для проверки корректности входных данных (непустые строки, положительный возраст).

Вариант 3. Анализатор текстового файла. Разработать приложение для анализа текстового файла: подсчет количества строк, слов, символов, поиск самого длинного слова. Использовать Debug для вывода промежуточных результатов на каждом этапе анализа. Применить Trace.TraceInformation() и Trace.TraceError() для логирования успешных операций и ошибок чтения файла.

Вариант 4. Конвертер систем счисления. Разработать приложение для конвертации чисел между различными системами счисления (двоичная, восьмеричная, десятичная, шестнадцатеричная). Использовать Debug.WriteLine() для вывода промежуточных шагов преобразования. Применить Debug.Assert() для проверки корректности входных данных. Использовать Trace для записи всех конвертаций в файл.

Вариант 5. Генератор и анализатор паролей. Разработать приложение для генерации случайных паролей с заданными параметрами (длина, наличие цифр, спецсимволов) и проверки надежности паролей. Использовать Debug для вывода критериев проверки надежности. Применить Trace для логирования сгенерированных паролей и результатов проверки в файл. Использовать Debug.Assert() для проверки минимальной длины пароля.

Вариант 6. Учет товаров на складе. Разработать приложение для учета товаров на складе (добавление товара, списание, поиск, вывод остатков). Использовать Debug.WriteLine() для отслеживания каждой операции с товарами. Применить Debug.Assert() для проверки, что количество товара не может быть отрицательным. Использовать Trace для записи всех операций в файл журнала.

Вариант 7. Обработка массива чисел. Разработать приложение для работы с массивом целых чисел: заполнение слу-

чайными числами, поиск минимума/максимума, сортировка, вычисление среднего значения. Использовать Debug.WriteLine() для вывода массива на каждом этапе обработки. Применить Debug.Indent()/Unindent() для структурирования вывода. Использовать Trace для логирования результатов операций**.**

Вариант 8. Система регистрации пользователей. Разработать приложение для регистрации пользователей с проверкой уникальности логина, валидацией email и пароля. Использовать Debug.WriteLineIf() для условного вывода сообщений валидации. Применить Debug.Assert() для проверки формата email. Использовать Trace.TraceWarning() для предупреждений о слабых паролях, Trace.TraceError() для ошибок регистрации.

Вариант 9. Библиотека математических функций. Разработать библиотеку с математическими функциями (факториал, числа Фибоначчи, проверка простоты числа, НОД, НОК). Использовать Debug.WriteLine() для вывода промежуточных шагов вычислений. Применить Debug.Assert() для проверки входных параметров (неотрицательные числа). Использовать Trace для логирования вызовов функций с параметрами и результатами.

Вариант 10. Планировщик задач. Разработать приложение для управления списком задач (добавление задачи с приоритетом и сроком, удаление, отметка выполненной, вывод по критериям). Использовать Debug для отслеживания изменений состояния задач. Применить Trace.TraceInformation() для логирования создания задач, Trace.TraceWarning() для задач с истекшим сроком. Использовать Debug.Assert() для проверки корректности дат.

Требования к выполнению задания

1.Создайте консольное приложение на C# в Visual Studio 2022, реализующее функционал в соответствии с вариантом.

2.Подключите пространство имен System.Diagnostics**:** using System.Diagnostics;

3.Примените класс Debug:

– Используйте Debug.WriteLine() минимум в 5 местах для вывода хода выполнения программы

– Используйте Debug.Assert() минимум в 2 местах для про-

верки критических условий

– Примените Debug.WriteLineIf() минимум в 1 месте для условного вывода

– При необходимости используйте Debug.Indent()/Unindent() для структурирования вывода

4.Примените класс Trace:

– Добавьте TextWriterTraceListener для записи в файл

– Используйте Trace.WriteLine() минимум в 3 местах

– Используйте Trace.TraceInformation(), Trace.TraceWarning() или Trace.TraceError() минимум в 2 местах

– Установите Trace.AutoFlush = true

5.Организуйте вывод информации: Выводите информативные сообщения о выполняемых операциях; Выводите значения переменных в ключевых точках программы; Структурируйте вывод для удобства чтения

6.Протестируйте приложение:

– Выполните программу в конфигурации Debug

– Просмотрите вывод в окне Output (View → Output, выбрать Debug)

– Проверьте создание и содержимое файла журнала

– Убедитесь, что Debug.Assert() срабатывает при нарушении условий

7.Сравните конфигурации Debug и Release:

– Переключите проект в режим Release

– Убедитесь, что Debug-сообщения не выводятся

– Проверьте, что Trace-сообщения продолжают записываться в файл

Содержание отчета. Отчет должен содержать:

1.Листинг программы с комментариями, объясняющими применение отладочных классов.

2.Скриншоты: Окно Output в Visual Studio с отладочными сообщениями; Содержимое файла журнала (log-файла); Примеры срабатывания Debug.Assert() (при наличии); Главное окно программы с результатами работы

3.Описание использованных методов: Перечень использованных методов Debug и Trace; Назначение каждого метода в контексте вашей программы

4.Анализ результатов: Как отладочные классы помогли в разработке и тестировании; Различия в поведении Debug и Trace; Различия между конфигурациями Debug и Release

5.Выводы.

Контрольные вопросы

1.В чем заключается основное различие между классами Debug и Trace? В каких конфигурациях проекта они работают?

2.Какие основные методы класса Debug вы знаете? Опишите назначение методов WriteLine(), Assert(), WriteLineIf().

3.Что такое прослушиватели (Listeners) в контексте отладочных классов? Какие типы прослушивателей существуют в .NET Framework?

4.Как настроить запись отладочной информации в текстовый файл с использованием класса Trace?

5.Для чего используется метод Debug.Assert()? Что происходит при нарушении проверяемого условия?

6.В чем преимущество использования Trace.TraceInformation(), Trace.TraceWarning() и Trace.TraceError() по сравнению с обычным Trace.WriteLine()?

7.Как просмотреть вывод отладочных сообщений в Visual Studio? Какое окно для этого используется?

8.Что означает свойство Trace.AutoFlush и зачем его устанавливать в true?

9.Как директивы условной компиляции (#if DEBUG) связаны с работой отладочных классов?

10.Почему использование отладочных классов предпочтительнее, чем Console.WriteLine() для отладки приложений?

Время выполнения: 2 ч

Практическая работа № 9 «отладка проекта»

Цель работы: Изучить методы и инструменты отладки программных модулей в среде Visual Studio 2022, освоить технологии поиска и устранения ошибок в программном коде на языке C#, приобрести практические навыки использования отладчика для анализа состояния программы во время выполнения.

Теоретическая часть

Понятие отладки программного обеспечения. Отладка (debugging) — это процесс обнаружения, локализации и устранения ошибок в программном коде. Отладка является неотъемлемой частью процесса разработки и интеграции программных модулей. *Основные цели отладки:*

- обнаружение логических ошибок в алгоритмах;
- выявление ошибок времени выполнения;
- проверка корректности работы интегрированных модулей;
- анализ состояния переменных и объектов в процессе выполнения программы;
- оптимизация производительности кода.

Типы ошибок в программном обеспечении

1. Синтаксические ошибки (Syntax Errors) Ошибки, связанные с нарушением правил языка программирования. Обнаруживаются компилятором на этапе компиляции.

Пример:

```
int x = 5
Console.WriteLine(x); // отсутствует точка с запятой
```

2. Логические ошибки (Logic Errors) Ошибки в алгоритме программы, при которых программа компилируется и выполняется, но выдает некорректные результаты.

Пример:

```
int sum = a - b; // должно быть a + b
```

3. Ошибки времени выполнения (Runtime Errors) Ошибки, возникающие во время работы программы и приводящие к её аварийному завершению. Примеры: деление на ноль, обращение к несуществующему элементу массива, работа с null-ссылками.

Инструменты отладки в Visual Studio 2022. Точки останова (Breakpoints) Точка останова — это маркер в коде, при достижении которого выполнение программы приостанавливается, что позволяет проанализировать текущее состояние программы. *Типы точек останова:*

– обычная точка останова (F9) — останов на определенной строке;
– условная точка останова — останов при выполнении определенного условия;
– точка трассировки — вывод сообщения без остановки выполнения;
– временная точка останова — срабатывает один раз.

Пошаговое выполнение

- **F10 (Step Over)** — выполнить текущую строку и перейти к следующей, не заходя внутрь вызываемых методов;
- **F11 (Step Into)** — выполнить текущую строку с заходом внутрь вызываемых методов;
- **Shift+F11 (Step Out)** — выйти из текущего метода;
- **F5 (Continue)** — продолжить выполнение до следующей точки останова.

Окна отладки Visual Studio

1. Окно Locals (Локальные переменные) Отображает все локальные переменные текущей области видимости и их значения.
2. Окно Watch (Контрольные значения) Позволяет добавлять произвольные выражения для отслеживания их значений.
3. Окно Call Stack (Стек вызовов) Показывает последовательность вызовов методов, приведших к текущей точке выполнения.
4. Окно Autos Автоматически отображает переменные, используемые в текущей и предыдущей строках кода.
5. Окно Immediate (Интерпретация) Позволяет выполнять команды и вычислять выражения во время отладки.
6. Окно Output (Вывод) Отображает диагностическую информацию и сообщения отладки.

Методы отладки

1. Отладка по принципу "разделяй и властвуй" Разделение кода на логические блоки и последовательная проверка каждого блока.
2. Отладка методом обратного прослеживания Начиная с места возникновения ошибки, движение в обратном направлении по коду для поиска первопричины.
3. Использование логирования Вывод промежуточных значений переменных и состояний программы в консоль или файл.
Пример:

```
Debug.WriteLine($"Значение переменной x: {x}");
Console.WriteLine($"Итерация {i}: результат = {result}");
```

4. Модульное тестирование Создание тестовых методов для проверки отдельных модулей программы.
Обработка исключений. Конструкция try-catch-finally **используется для перехвата и обработки исключений:**

```
try
{
    // Код, который может вызвать исключение
    int result = 10 / 0;
}
catch (DivideByZeroException ex)
{
    // Обработка конкретного типа исключения
    Console.WriteLine($"Ошибка: {ex.Message}");
}
catch (Exception ex)
{
    // Обработка всех остальных исключений
    Console.WriteLine($"Общая ошибка: {ex.Message}");
}
finally
{
    // Код, выполняемый в любом случае
    Console.WriteLine("Завершение операции");
}
```

Основные классы исключений в C#:

- Exception — базовый класс для всех исключений;
- SystemException — системные исключения;
- ApplicationException — исключения приложения;
- NullReferenceException — обращение к null-объекту;
- IndexOutOfRangeException — выход за границы массива;
- ArgumentException — некорректный аргумент метода;
- InvalidOperationException — недопустимая операция;
- FormatException — ошибка формата данных.

Директивы препроцессора для отладки

```
#if DEBUG
    Console.WriteLine("Режим отладки");
#endif

#define DEBUG_MODE
#if DEBUG_MODE
    // Код для отладки
#endif
```

Атрибуты отладки

```
[Conditional("DEBUG")]
public static void DebugMessage(string message)
{
```

```
    Console.WriteLine($"Debug: {message}");
}
```

Лучшие практики отладки

1.Используйте осмысленные имена переменных и методов.
2.Добавляйте комментарии к сложным участкам кода.
3.Проверяйте граничные условия (пустые массивы, нулевые значения).
4.Используйте assert для проверки предусловий.
5.Логируйте критические операции.
6.Тестируйте код с различными входными данными.
7.Регулярно проводите code review.

Рабочее задание: Выполните отладку программного кода в соответствии с вариантом задания. Найдите и исправьте все ошибки, используя инструменты отладки Visual Studio 2022.

Вариант 1. Калькулятор с ошибками. Дан код калькулятора с несколькими ошибками. Найдите и исправьте все ошибки, используя отладчик.

```
using System;

namespace Calculator
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Калькулятор");
            Console.Write("Введите первое число: ");
            double num1 = Convert.ToDouble(Console.ReadLine());

            Console.Write("Введите операцию (+, -, *, /): ");
            string operation = Console.ReadLine();

            Console.Write("Введите второе число: ");
            double num2 = Convert.ToDouble(Console.ReadLine());

            double result = Calculate(num1, num2, operation);
            Console.WriteLine($"Результат: {result}");
        }

        static double Calculate(double a, double b, string op)
```

```
        {
            double result = 0;

            if (op == "+")
                result = a + b;
            else if (op == "-")
                result = a - b;
            else if (op == "*")
                result = a * b;
            else if (op == "/")
                result = a / b; // Ошибка: нет проверки деления на ноль

            return result;
        }
    }
}
```

Задачи:

1.Добавьте обработку исключений для некорректного ввода.
2.Добавьте проверку деления на ноль.
3.Добавьте обработку неизвестной операции.
4.Используйте точки останова для проверки значений переменных.
5.Протестируйте программу с различными входными данными.

Вариант 2. Работа с массивами. Дан код для работы с массивом. Найдите и исправьте ошибки.

```
using System;

namespace ArrayProcessing
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] numbers = { 5, 12, 8, 3, 15, 7, 20 };

            Console.WriteLine("Массив чисел:");
            PrintArray(numbers);

            int max = FindMax(numbers);
            Console.WriteLine($"Максимальное значение: {max}");

            double avg = CalculateAverage(numbers);
            Console.WriteLine($"Среднее значение: {avg}");
```

```
    int index = FindElement(numbers, 15);
    Console.WriteLine($"Индекс элемента 15: {index}");
}

static void PrintArray(int[] arr)
{
    for (int i = 0; i <= arr.Length; i++) // Ошибка: выход за границы
    {
        Console.Write(arr[i] + " ");
    }
    Console.WriteLine();
}

static int FindMax(int[] arr)
{
    int max = 0; // Ошибка: неправильная инициализация
    for (int i = 0; i < arr.Length; i++)
    {
        if (arr[i] > max)
            max = arr[i];
    }
    return max;
}

static double CalculateAverage(int[] arr)
{
    int sum = 0;
    for (int i = 0; i < arr.Length; i++)
    {
        sum += arr[i];
    }
    return sum / arr.Length; // Ошибка: целочисленное деление
}

static int FindElement(int[] arr, int value)
{
    for (int i = 0; i < arr.Length; i++)
    {
        if (arr[i] == value)
            return i;
    }
    return 0; // Ошибка: неправильное возвращаемое значение
}
```

```
    }
}
```

Задачи:

1.Исправьте ошибку выхода за границы массива.
2.Исправьте инициализацию максимального значения.
3.Исправьте целочисленное деление.
4.Исправьте возвращаемое значение при отсутствии элемента.
5.Добавьте проверку на пустой массив.

Вариант 3. Класс "Студент". Дан код класса Student с ошибками. Найдите и исправьте их.

```
using System;
using System.Collections.Generic;

namespace StudentManagement
{
    class Student
    {
        public string Name { get; set; }
        public int Age { get; set; }
        public List<int> Grades { get; set; }

        public Student(string name, int age)
        {
            Name = name;
            Age = age;
        }

        public void AddGrade(int grade)
        {
            Grades.Add(grade); // Ошибка: Grades не инициализирован
        }

        public double GetAverageGrade()
        {
            int sum = 0;
            foreach (int grade in Grades)
            {
                sum += grade;
            }
            return sum / Grades.Count; // Ошибка: целочисленное деление
        }

        public void PrintInfo()
```

```
        {
            Console.WriteLine($"Имя: {Name}");
            Console.WriteLine($"Возраст: {Age}");
            Console.WriteLine($"Средний балл: {GetAverageGrade()}");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Student student = new Student("Иван Петров", 19);

            student.AddGrade(5);
            student.AddGrade(4);
            student.AddGrade(5);
            student.AddGrade(3);

            student.PrintInfo();

            Student student2 = new Student("Мария Сидорова", 20);
            student2.PrintInfo(); // Ошибка: оценки не добавлены
        }
    }
}
```

Задачи:

1.Инициализируйте список оценок в конструкторе.
2.Исправьте целочисленное деление.
3.Добавьте проверку на пустой список оценок.
4.Добавьте валидацию оценок (от 2 до 5).
5.Добавьте обработку исключений.

Вариант 4. Банковский счет. Дан код класса BankAccount с логическими ошибками.

```
using System;

namespace BankSystem
{
    class BankAccount
    {
        public string AccountNumber { get; set; }
        public string Owner { get; set; }
        public decimal Balance { get; private set; }
```

```
        public BankAccount(string accountNumber, string owner, decimal initial-
Balance)
        {
            AccountNumber = accountNumber;
            Owner = owner;
            Balance = initialBalance;
        }

        public void Deposit(decimal amount)
        {
            Balance += amount;
            Console.WriteLine($"Пополнение на {amount}. Баланс: {Balance}");
        }

        public void Withdraw(decimal amount)
        {
            Balance -= amount; // Ошибка: нет проверки достаточности средств
            Console.WriteLine($"Снятие {amount}. Баланс: {Balance}");
        }

        public void Transfer(BankAccount destination, decimal amount)
        {
            this.Withdraw(amount);
            destination.Deposit(amount);
            Console.WriteLine($"Перевод {amount} на счет {destina-
tion.AccountNumber}");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            BankAccount account1 = new BankAccount("001", "Иванов И.И.",
10000);
            BankAccount account2 = new BankAccount("002", "Петров П.П.", 5000);

            account1.Deposit(2000);
            account1.Withdraw(15000); // Ошибка: недостаточно средств

            account1.Transfer(account2, 8000);

            Console.WriteLine($"\nИтоговый баланс счета {ac-
count1.AccountNumber}: {account1.Balance}");
```

```
            Console.WriteLine($"Итоговый баланс счета {ac-
count2.AccountNumber}: {account2.Balance}");
        }
    }
}
```

Задачи:

1.Добавьте проверку достаточности средств при снятии.
2.Добавьте валидацию суммы (должна быть положительной).
3.Обработайте ситуацию с null-счетом при переводе.
4.Добавьте исключения для некорректных операций.
5.Протестируйте все сценарии с помощью отладчика.

Вариант 5. Работа со строками. Дан код для обработки строк с ошибками.

```
using System;

namespace StringProcessing
{
    class Program
    {
        static void Main(string[] args)
        {
            string text = "Программирование на C#";

            Console.WriteLine($"Исходная строка: {text}");
            Console.WriteLine($"Количество слов: {CountWords(text)}");
            Console.WriteLine($"Количество гласных: {CountVowels(text)}");
            Console.WriteLine($"Перевернутая строка: {ReverseString(text)}");
            Console.WriteLine($"Палиндром: {IsPalindrome(text)}");
        }

        static int CountWords(string text)
        {
            string[] words = text.Split(' ');
            return words.Length;
        }

        static int CountVowels(string text)
        {
            int count = 0;
            string vowels = "аеёиоуыэюяАЕЁИОУЫЭЮЯ";

            foreach (char c in text)
            {
```

```
            if (vowels.Contains(c))
                count++;
        }
        return count;
    }

    static string ReverseString(string text)
    {
        char[] chars = text.ToCharArray();
        Array.Reverse(chars);
        return new string(chars);
    }

    static bool IsPalindrome(string text)
    {
        string reversed = ReverseString(text);
        return text == reversed; // Ошибка: не учитывает регистр и пробелы
    }
  }
}
```

Задачи:

1.Исправьте проверку на палиндром (игнорировать регистр и пробелы).

2.Добавьте проверку на null и пустую строку.

3.Исправьте подсчет слов (учесть множественные пробелы).

4.Добавьте обработку исключений.

5.Протестируйте с различными входными данными.

Вариант 6. Сортировка массива. Дан код сортировки с ошибками.

```
using System;

namespace Sorting
{
  class Program
  {
    static void Main(string[] args)
    {
      int[] numbers = { 64, 34, 25, 12, 22, 11, 90 };

      Console.WriteLine("Исходный массив:");
      PrintArray(numbers);

      BubbleSort(numbers);
```

```
        Console.WriteLine("\nОтсортированный массив:");
        PrintArray(numbers);

        int searchValue = 25;
        int index = BinarySearch(numbers, searchValue);
        Console.WriteLine($"\nЭлемент {searchValue} найден на позиции: {in-
dex}");
    }

    static void BubbleSort(int[] arr)
    {
        int n = arr.Length;
        for (int i = 0; i < n; i++) // Ошибка: лишние итерации
        {
            for (int j = 0; j < n - 1; j++) // Ошибка: выход за границы
            {
                if (arr[j] > arr[j + 1])
                {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    static int BinarySearch(int[] arr, int value)
    {
        int left = 0;
        int right = arr.Length;

        while (left <= right)
        {
            int mid = (left + right) / 2;

            if (arr[mid] == value)
                return mid;
            else if (arr[mid] < value)
                left = mid + 1;
            else
                right = mid - 1;
        }
```

```
            return -1;
        }

        static void PrintArray(int[] arr)
        {
            foreach (int num in arr)
            {
                Console.Write(num + " ");
            }
            Console.WriteLine();
        }
    }
}
```

Задачи:

1.Оптимизируйте сортировку пузырьком.
2.Исправьте ошибку в бинарном поиске (right = arr.Length - 1).
3.Добавьте проверку на пустой массив.
4.Используйте отладчик для пошагового прохода по алгоритму.
5.Протестируйте с различными массивами.

Вариант 7. Работа с файлами. Дан код для работы с файлами с ошибками.

```
using System;
using System.IO;

namespace FileProcessing
{
    class Program
    {
        static void Main(string[] args)
        {
            string filePath = "data.txt";

            WriteToFile(filePath, "Первая строка");
            WriteToFile(filePath, "Вторая строка");
            WriteToFile(filePath, "Третья строка");

            string content = ReadFromFile(filePath);
            Console.WriteLine("Содержимое файла:");
            Console.WriteLine(content);

            int lineCount = CountLines(filePath);
            Console.WriteLine($"\nКоличество строк: {lineCount}");
        }
```

```
    static void WriteToFile(string path, string text)
    {
        File.WriteAllText(path, text); // Ошибка: перезаписывает файл
    }

    static string ReadFromFile(string path)
    {
        string content = File.ReadAllText(path);
        return content;
    }

    static int CountLines(string path)
    {
        string[] lines = File.ReadAllLines(path);
        return lines.Length;
    }
  }
}
```

Задачи:

1.Исправьте метод записи (использовать AppendAllText).
2.Добавьте проверку существования файла.
3.Добавьте обработку исключений при работе с файлами.
4.Добавьте метод для очистки файла.
5.Протестируйте все методы с использованием отладчика.

Вариант 8. Класс "Прямоугольник". Дан код класса Rectangle с ошибками.

```
using System;

namespace Geometry
{
  class Rectangle
  {
    public double Width { get; set; }
    public double Height { get; set; }

    public Rectangle(double width, double height)
    {
      Width = width;
      Height = height;
    }

    public double CalculateArea()
```

```
    {
        return Width * Height;
    }

    public double CalculatePerimeter()
    {
        return Width + Height; // Ошибка: неправильная формула
    }

    public double CalculateDiagonal()
    {
        return Math.Sqrt(Width + Height); // Ошибка: неправильная формула
    }

    public bool IsSquare()
    {
        return Width == Height;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Rectangle rect = new Rectangle(5, 10);

        Console.WriteLine($"Ширина: {rect.Width}");
        Console.WriteLine($"Высота: {rect.Height}");
        Console.WriteLine($"Площадь: {rect.CalculateArea()}");
        Console.WriteLine($"Периметр: {rect.CalculatePerimeter()}");
        Console.WriteLine($"Диагональ: {rect.CalculateDiagonal()}");
        Console.WriteLine($"Это квадрат: {rect.IsSquare()}");

        Rectangle rect2 = new Rectangle(-5, 10); // Ошибка: отрицательные зна-
чения
        Console.WriteLine($"\nПлощадь второго прямоугольника:
{rect2.CalculateArea()}");
    }
}
}
```

Задачи:

1.Исправьте формулу периметра (2 * (Width + Height)).

2.Исправьте формулу диагонали ($\sqrt{(Width^2 + Height^2)}$).

3.Добавьте валидацию разме- ров (должны быть положи-

тельными).
4.Добавьте обработку исключений.
5.Используйте точки останова для проверки вычислений.
Вариант 9. Список задач (To-Do List). Дан код приложения для управления задачами с ошибками.

```
using System;
using System.Collections.Generic;

namespace TodoList
{
    class Task
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public bool IsCompleted { get; set; }

        public Task(int id, string title)
        {
            Id = id;
            Title = title;
            IsCompleted = false;
        }
    }

    class TodoManager
    {
        private List<Task> tasks;
        private int nextId;

        public TodoManager()
        {
            tasks = new List<Task>();
            nextId = 1;
        }

        public void AddTask(string title)
        {
            Task task = new Task(nextId++, title);
            tasks.Add(task);
            Console.WriteLine($"Задача добавлена: {title}");
        }

        public void CompleteTask(int id)
```

```
    {
        Task task = tasks.Find(t => t.Id == id);
        task.IsCompleted = true; // Ошибка: не проверяет null
        Console.WriteLine($"Задача {id} выполнена");
    }

    public void RemoveTask(int id)
    {
        tasks.RemoveAt(id); // Ошибка: удаляет по индексу, а не по Id
        Console.WriteLine($"Задача {id} удалена");
    }

    public void PrintTasks()
    {
        Console.WriteLine("\nСписок задач:");
        foreach (Task task in tasks)
        {
            string status = task.IsCompleted ? "[✓]" : "[ ]";
            Console.WriteLine($"{task.Id}. {status} {task.Title}");
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        TodoManager manager = new TodoManager();

        manager.AddTask("Изучить C#");
        manager.AddTask("Выполнить практическую работу");
        manager.AddTask("Подготовиться к экзамену");

        manager.PrintTasks();

        manager.CompleteTask(1);
        manager.CompleteTask(5); // Ошибка: несуществующая задача

        manager.RemoveTask(2);

        manager.PrintTasks();
    }
}
}
```

Задачи:

1.Добавьте проверку на null при выполнении задачи.
2.Исправьте метод удаления (искать по Id, а не по индексу).
3.Добавьте обработку несуществующих задач.
4.Добавьте проверку на пустое название задачи.
5.Используйте отладчик для проверки операций со списком.

Вариант 10. Конвертер температур. Дан код конвертера температур с ошибками.

```
using System;

namespace TemperatureConverter
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Конвертер температур");
         Console.WriteLine("1. Цельсий -> Фаренгейт");
         Console.WriteLine("2. Фаренгейт -> Цельсий");
         Console.WriteLine("3. Цельсий -> Кельвин");
         Console.WriteLine("4. Кельвин -> Цельсий");

         Console.Write("\nВыберите операцию: ");
         int choice = int.Parse(Console.ReadLine()); // Ошибка: нет обработки
исключений

         Console.Write("Введите температуру: ");
         double temp = double.Parse(Console.ReadLine());

         double result = ConvertTemperature(choice, temp);
         Console.WriteLine($"Результат: {result}");
      }

      static double ConvertTemperature(int operation, double temp)
      {
         double result = 0;

         switch (operation)
         {
            case 1: // Цельсий -> Фаренгейт
               result = (temp * 9/5) + 32;
               break;
            case 2: // Фаренгейт -> Цельсий
```

```
                result = (temp - 32) * 5/9;
                break;
            case 3: // Цельсий -> Кельвин
                result = temp + 273.15;
                break;
            case 4: // Кельвин -> Цельсий
                result = temp - 273.15;
                break;
        }

        return result;
    }
  }
}
```

Задачи:

1.Добавьте обработку исключений для некорректного ввода.
2.Добавьте валидацию температуры (не может быть ниже абсолютного нуля).
3.Добавьте обработку неизвестной операции.
4.Исправьте целочисленное деление в формулах (9.0/5.0).
5.Добавьте возможность повторного выполнения операции.

Требования к выполнению задания

1.Изучите предоставленный код: внимательно прочитайте код варианта; определите назначение каждого метода; выявите потенциальные проблемные места.

2.Выполните отладку с использованием инструментов Visual Studio: установите точки останова на критических участках кода; используйте пошаговое выполнение (F10, F11); анализируйте значения переменных в окнах Locals и Watch; изучите стек вызовов при возникновении ошибок.

3.Найдите и исправьте все ошибки: синтаксические ошибки; логические ошибки; ошибки времени выполнения; потенциальные исключения.

4.Добавьте обработку исключений: используйте конструкции try-catch-finally; обработайте специфические типы исключений; добавьте информативные сообщения об ошибках.

5.Добавьте валидацию входных данных: проверяйте корректность ввода пользователя; проверяйте граничные условия; обрабатывайте некорректные значения.

6.Протестируйте исправленный код: проверьте работу с корректными данными; проверьте работу с некорректными данными; проверьте граничные случаи; убедитесь в отсутствии необработанных исключений.
7.Оформите отчет, содержащий: описание найденных ошибок (тип ошибки, причина, последствия); скриншоты процесса отладки (точки останова, значения переменных); исправленный код с комментариями; результаты тестирования;

Контрольные вопросы

1.Что такое отладка программного обеспечения? Какие основные цели преследует процесс отладки?

2.Перечислите основные типы ошибок в программном коде. Приведите примеры каждого типа ошибок на языке C#.

3.Что такое точка останова (breakpoint)? Какие типы точек останова поддерживает Visual Studio 2022?

4.В чем разница между командами Step Over (F10) и Step Into (F11) при пошаговой отладке?

5.Какие окна отладки доступны в Visual Studio 2022? Опишите назначение окон Locals, Watch и Call Stack.

6.Для чего используется конструкция try-catch-finally? Приведите пример обработки исключений в C#.

7.Перечислите основные классы исключений в C#. Какой класс является базовым для всех исключений?

8.Что такое стек вызовов (Call Stack) и как он помогает в процессе отладки?

9.Какие методы отладки вы знаете? Опишите принцип "разделяй и властвуй" в контексте отладки.

10.Перечислите лучшие практики отладки программного кода. Почему важно использовать осмысленные имена переменных и добавлять комментарии к коду?

Время выполнения: 4 ч

Практическая работа № 10 «Инспекция кода модулей проекта»

Цель работы: Изучить методы и инструменты инспекции кода программных модулей, освоить практические навыки статического анализа кода, выявления потенциальных ошибок, проверки соответствия стандартам кодирования и оценки качества программного кода на языке C# с использованием среды разработки Visual Studio 2022.

Теоретическая часть

Понятие инспекции кода. **Инспекция кода** (Code Inspection) — это процесс систематического анализа исходного кода программы с целью выявления ошибок, несоответствий стандартам, потенциальных уязвимостей и возможностей для улучшения качества кода. Инспекция является важной составляющей процесса интеграции программных модулей и обеспечения качества программного обеспечения.

Цели инспекции кода

1.Выявление ошибок на ранних этапах — обнаружение дефектов до начала тестирования

2.Повышение качества кода — улучшение читаемости, сопровождаемости и производительности

3.Соблюдение стандартов кодирования — проверка соответствия принятым соглашениям

4.Снижение технического долга — предотвращение накопления проблемных участков кода

5.Обеспечение безопасности — выявление потенциальных уязвимостей

6.Обучение разработчиков — обмен знаниями и лучшими практиками в команде

Виды инспекции кода

1. Статический анализ. Анализ исходного кода без его выполнения. Выполняется автоматизированными инструментами или вручную. Преимущества: не требует запуска программы; позволяет проверить все возможные пути выполнения; выявляет проблемы на ранних этапах

2. Динамический анализ. Анализ программы во время ее выполнения с помощью специальных инструментов. Преимущества: обнаруживает проблемы, проявляющиеся только во время выполнения; позволяет выявить утечки памяти и проблемы производительности

3. Ручная инспекция (Code Review). Проверка кода коллегами-разработчиками. Преимущества: выявление логических ошибок; обмен опытом в команде; улучшение архитектуры решений

Методы инспекции кода

1.Парное программирование — два разработчика работают за одним компьютером

2.Формальная инспекция — структурированный процесс с четкими ролями и этапами

3.Неформальный просмотр — быстрая проверка кода коллегой

4.Инспекция с помощью инструментов — автоматизированный анализ специальными программами

Стандарты кодирования C#. При разработке на языке C# рекомендуется следовать соглашениям Microsoft:

Соглашения об именовании:

- **PascalCase** для имен классов, методов, свойств: CustomerRepository, GetCustomerById()
- **camelCase** для локальных переменных и параметров: customerName, orderId
- **_camelCase** для приватных полей: _connectionString, _userId
- **UPPER_CASE** для констант: MAX_CONNECTIONS, DEFAULT_TIMEOUT

Правила форматирования:

- Использование фигурных скобок даже для однострочных блоков
- Отступы: 4 пробела (не табуляция)
- Максимальная длина строки: 120-130 символов
- Пустая строка между методами и логическими блоками

Принципы написания кода:

- **SOLID принципы** — основные принципы объектно-

ориентированного проектирования
- **DRY (Don't Repeat Yourself)** — избегание дублирования кода
- **KISS (Keep It Simple, Stupid)** — простота решений
- **YAGNI (You Aren't Gonna Need It)** — не писать код "на будущее"

Инструменты инспекции кода в Visual Studio 2022

1. Встроенный анализатор кода (Code Analysis). Visual Studio включает анализаторы Roslyn, которые проверяют код в реальном времени. **Подключение:**
- Правый клик на проекте → Properties → Code Analysis
- Установка уровня анализа: Default, All, Recommended

2. Code Cleanup. Инструмент автоматического форматирования и исправления кода. **Использование:**
- Правый клик в редакторе → Code Cleanup → Run Code Cleanup
- Настройка профилей: Tools → Options → Text Editor → Code Cleanup

3. Code Metrics. Вычисление метрик качества кода. **Метрики:**
- **Индекс сопровождаемости** (Maintainability Index) — от 0 до 100, чем выше, тем лучше
- **Цикломатическая сложность** (Cyclomatic Complexity) — количество независимых путей выполнения
- **Глубина наследования** (Depth of Inheritance)
- **Связанность классов** (Class Coupling)
- **Строки кода** (Lines of Code)

Запуск: Analyze → Calculate Code Metrics

4. StyleCop Analyzers. NuGet-пакет для проверки соответствия стандартам стиля кодирования. **Установка:** Install-Package StyleCop.Analyzers

5. SonarLint. Расширение для обнаружения ошибок и уязвимостей в коде. **Установка:** Extensions → Manage Extensions → поиск "SonarLint"

Типичные проблемы, выявляемые при инспекции

1. Потенциальные исключения: обращение к null-объектам (NullReferenceException); деление на ноль; выход за границы массива; некорректная работа с файлами и ресурсами

2. Проблемы производительности: неоптимальные алгоритмы; избыточные запросы к базе данных; утечки памяти; неправильное использование коллекций

3. Нарушения безопасности: SQL-инъекции; незащищенное хранение паролей; недостаточная валидация входных данных; уязвимости межсайтового скриптинга (XSS)

4. Проблемы сопровождаемости: дублирование кода; слишком сложные методы; отсутствие или неточные комментарии; несоответствие стандартам именования

5. Архитектурные проблемы: нарушение принципов SOLID; сильная связанность компонентов; отсутствие обработки ошибок; смешивание уровней абстракции

Контрольный список инспекции кода (Code Review Checklist)

Общее качество:

- ☐ Код соответствует требованиям задачи
- ☐ Логика работы понятна и прозрачна
- ☐ Отсутствует дублирование кода
- ☐ Код следует принципам SOLID

Именование:

- ☐ Имена переменных, методов, классов осмысленны
- ☐ Соблюдаются соглашения об именовании C#
- ☐ Константы выделены и названы правильно

Обработка ошибок:

- ☐ Исключения обрабатываются корректно
- ☐ Используются appropriate типы исключений
- ☐ Ресурсы освобождаются (using statement)

Производительность:

- ☐ Нет очевидных проблем производительности
- ☐ Используются подходящие коллекции
- ☐ Запросы к БД оптимизированы

Безопасность:

- ☐ Входные данные валидируются
- ☐ Нет SQL-инъекций
- ☐ Конфиденциальные данные защищены

Тестируемость:

- ☐ Код поддается модульному тестированию
- ☐ Зависимости можно подменить (DI)
- ☐ Методы имеют единственную ответственность

Рабочее задание: Выполните инспекцию кода программного модуля в соответствии с вариантом задания. Используйте встроенные инструменты Visual Studio 2022 и руководство по инспекции кода.

Вариант 1. Модуль управления пользователями. Проведите инспекцию следующего кода модуля управления пользователями:

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;

namespace UserManagement
{
    public class UserManager
    {
        private string connStr = "Server=localhost;Database=Users;User
Id=sa;Password=123;";
```

```
    public List<User> users = new List<User>();

    public void AddUser(string name, string password, string email)
    {
        User u = new User();
        u.Name = name;
        u.Password = password;
        u.Email = email;
        u.Id = users.Count + 1;
        users.Add(u);

        SqlConnection conn = new SqlConnection(connStr);
        conn.Open();
        string query = "INSERT INTO Users VALUES ('" + u.Id + "', '" + name +
"', '" + password + "', '" + email + "')";
        SqlCommand cmd = new SqlCommand(query, conn);
        cmd.ExecuteNonQuery();
        conn.Close();
    }

    public User GetUser(int id)
    {
        for(int i = 0; i < users.Count; i++)
        {
            if(users[i].Id == id)
                return users[i];
        }
        return null;
    }

    public bool CheckPassword(string name, string password)
    {
        foreach(var user in users)
        {
            if(user.Name == name && user.Password == password)
                return true;
        }
        return false;
    }

    public void DeleteUser(int id)
    {
        User user = GetUser(id);
```

```
            users.Remove(user);
        }
    }

    public class User
    {
        public int Id;
        public string Name;
        public string Password;
        public string Email;
    }
}
```

Вариант 2. Модуль обработки заказов. Проведите инспекцию следующего кода модуля обработки заказов:

```
using System;
using System.Collections.Generic;

namespace OrderProcessing
{
    public class OrderProcessor
    {
        public double CalculateTotal(List<OrderItem> items, string customerType,
string promoCode)
        {
            double total = 0;

            for(int i = 0; i < items.Count; i++)
            {
                total = total + items[i].Price * items[i].Quantity;
            }

            if(customerType == "VIP")
            {
                total = total - total * 0.15;
            }
            else if(customerType == "Regular")
            {
                total = total - total * 0.05;
            }

            if(promoCode == "SUMMER2024")
            {
                total = total - 100;
            }
```

```
        else if(promoCode == "WELCOME")
        {
            total = total - total * 0.1;
        }

        if(total > 5000)
        {
            total = total - total * 0.03;
        }

        return total;
    }

    public void ProcessOrder(Order order)
    {
        order.Status = "Processing";
        order.Total = CalculateTotal(order.Items, order.Customer.Type, or-
der.PromoCode);

        if(order.Total > 10000)
        {
            Console.WriteLine("Large order! Needs approval.");
        }

        order.Status = "Completed";
        SendEmail(order.Customer.Email, "Your order is completed");
    }

    public void SendEmail(string email, string message)
    {
        Console.WriteLine("Email sent to: " + email);
    }
}

public class Order
{
    public int Id;
    public List<OrderItem> Items;
    public Customer Customer;
    public string PromoCode;
    public double Total;
    public string Status;
}
```

```
    public class OrderItem
    {
        public string ProductName;
        public double Price;
        public int Quantity;
    }

    public class Customer
    {
        public string Name;
        public string Email;
        public string Type;
    }
}
```

Вариант 3. Модуль работы с файлами. Проведите инспекцию следующего кода модуля работы с файлами:

```
using System;
using System.IO;
using System.Collections.Generic;

namespace FileManager
{
    public class FileProcessor
    {
        public string ReadFile(string path)
        {
            StreamReader reader = new StreamReader(path);
            string content = reader.ReadToEnd();
            return content;
        }

        public void WriteFile(string path, string content)
        {
            File.WriteAllText(path, content);
        }

        public List<string> ParseCSV(string filePath)
        {
            List<string> result = new List<string>();
            string content = ReadFile(filePath);
            string[] lines = content.Split('\n');

            for(int i = 1; i < lines.Length; i++)
            {
```

```
            result.Add(lines[i]);
        }

        return result;
    }

    public void ProcessFiles(string directory)
    {
        string[] files = Directory.GetFiles(directory);

        foreach(string file in files)
        {
            string content = ReadFile(file);
            string newContent = content.ToUpper();
            string newFile = file.Replace(".txt", "_processed.txt");
            WriteFile(newFile, newContent);
        }
    }

    public int CountLines(string filePath)
    {
        string content = File.ReadAllText(filePath);
        string[] lines = content.Split('\n');
        return lines.Length;
    }
  }
}
```

Вариант 4. Модуль валидации данных. Проведите инспекцию следующего кода модуля валидации данных:

```
using System;
using System.Text.RegularExpressions;

namespace DataValidation
{
  public class Validator
  {
    public bool ValidateEmail(string email)
    {
      if(email.Contains("@") && email.Contains("."))
        return true;
      return false;
    }

    public bool ValidatePhone(string phone)
```

```
    {
        if(phone.Length == 11)
            return true;
        else
            return false;
    }

    public bool ValidatePassword(string password)
    {
        bool hasDigit = false;
        bool hasUpper = false;
        bool hasLower = false;

        for(int i = 0; i < password.Length; i++)
        {
            if(Char.IsDigit(password[i]))
                hasDigit = true;
            if(Char.IsUpper(password[i]))
                hasUpper = true;
            if(Char.IsLower(password[i]))
                hasLower = true;
        }

        if(password.Length >= 8 && hasDigit && hasUpper && hasLower)
            return true;

        return false;
    }

    public bool ValidateAge(int age)
    {
        if(age > 0 && age < 150)
            return true;
        return false;
    }

    public string ValidateForm(string name, string email, string phone, string
password, int age)
    {
        string errors = "";

        if(name.Length < 2)
            errors = errors + "Invalid name. ";
        if(!ValidateEmail(email))
```

```
            errors = errors + "Invalid email. ";
        if(!ValidatePhone(phone))
            errors = errors + "Invalid phone. ";
        if(!ValidatePassword(password))
            errors = errors + "Invalid password. ";
        if(!ValidateAge(age))
            errors = errors + "Invalid age. ";

        return errors;
    }
  }
}
```

Вариант 5. Модуль расчета скидок. Проведите инспекцию следующего кода модуля расчета скидок:

```
using System;

namespace DiscountCalculator
{
  public class DiscountService
  {
    public double discount = 0;

    public double CalculateDiscount(double amount, int customerYears, bool is-
VIP)
    {
      if(isVIP)
      {
        discount = 0.2;
      }
      else
      {
        if(customerYears >= 5)
          discount = 0.15;
        else if(customerYears >= 3)
          discount = 0.1;
        else if(customerYears >= 1)
          discount = 0.05;
      }

      if(amount > 10000)
        discount = discount + 0.05;

      return amount * (1 - discount);
    }
```

```
        public double ApplySeasonalDiscount(double amount, string season)
        {
            double seasonalDiscount = 0;

            if(season == "winter")
                seasonalDiscount = 0.1;
            if(season == "spring")
                seasonalDiscount = 0.05;
            if(season == "summer")
                seasonalDiscount = 0.15;
            if(season == "autumn")
                seasonalDiscount = 0.08;

            return amount * (1 - seasonalDiscount);
        }

        public double GetFinalPrice(double basePrice, int years, bool vip, string sea-
son)
        {
            double price1 = CalculateDiscount(basePrice, years, vip);
            double price2 = ApplySeasonalDiscount(price1, season);
            return price2;
        }
    }
}
```

Вариант 6. Модуль управления задачами. Проведите инспекцию следующего кода модуля управления задачами:

```
using System;
using System.Collections.Generic;

namespace TaskManagement
{
    public class TaskManager
    {
        public List<Task> tasks = new List<Task>();

        public void AddTask(string title, string description, string priority)
        {
            Task t = new Task();
            t.Id = tasks.Count + 1;
            t.Title = title;
            t.Description = description;
            t.Priority = priority;
```

```
        t.Status = "New";
        t.CreatedDate = DateTime.Now;
        tasks.Add(t);
    }

    public void CompleteTask(int id)
    {
        for(int i = 0; i < tasks.Count; i++)
        {
            if(tasks[i].Id == id)
            {
                tasks[i].Status = "Completed";
                tasks[i].CompletedDate = DateTime.Now;
                break;
            }
        }
    }

    public List<Task> GetHighPriorityTasks()
    {
        List<Task> result = new List<Task>();
        foreach(var task in tasks)
        {
            if(task.Priority == "High" || task.Priority == "high" || task.Priority ==
"HIGH")
                result.Add(task);
        }
        return result;
    }

    public void DeleteOldTasks(int days)
    {
        DateTime threshold = DateTime.Now.AddDays(-days);

        for(int i = tasks.Count - 1; i >= 0; i--)
        {
            if(tasks[i].CompletedDate < threshold)
                tasks.Remove(tasks[i]);
        }
    }

    public int GetTaskCount()
    {
        return tasks.Count;
```

```
        }
    }

    public class Task
    {
        public int Id;
        public string Title;
        public string Description;
        public string Priority;
        public string Status;
        public DateTime CreatedDate;
        public DateTime CompletedDate;
    }
}
```

Вариант 7. Модуль работы с инвентарем. Проведите инспекцию следующего кода модуля работы с инвентарем:

```
using System;
using System.Collections.Generic;

namespace InventoryManagement
{
    public class InventoryManager
    {
        private List<Product> products = new List<Product>();

        public void AddProduct(string name, int quantity, double price)
        {
            Product p = new Product();
            p.Id = products.Count + 1;
            p.Name = name;
            p.Quantity = quantity;
            p.Price = price;
            products.Add(p);
        }

        public void UpdateQuantity(int productId, int newQuantity)
        {
            Product product = null;
            for(int i = 0; i < products.Count; i++)
            {
                if(products[i].Id == productId)
                {
                    product = products[i];
                    break;
```

```
            }
        }
        product.Quantity = newQuantity;
    }

    public double CalculateInventoryValue()
    {
        double total = 0;
        foreach(var product in products)
        {
            total = total + product.Price * product.Quantity;
        }
        return total;
    }

    public List<Product> GetLowStockProducts()
    {
        List<Product> lowStock = new List<Product>();
        foreach(var product in products)
        {
            if(product.Quantity < 10)
                lowStock.Add(product);
        }
        return lowStock;
    }

    public void RemoveProduct(int id)
    {
        for(int i = 0; i < products.Count; i++)
        {
            if(products[i].Id == id)
            {
                products.RemoveAt(i);
                return;
            }
        }
    }
}

public class Product
{
    public int Id;
    public string Name;
    public int Quantity;
```

```
        public double Price;
    }
}
```

Вариант 8. Модуль логирования. Проведите инспекцию следующего кода модуля логирования:

```
using System;
using System.IO;

namespace LoggingModule
{
    public class Logger
    {
        private string logFile = "log.txt";

        public void Log(string message)
        {
            StreamWriter writer = new StreamWriter(logFile, true);
            writer.WriteLine(DateTime.Now + ": " + message);
            writer.Close();
        }

        public void LogError(string error)
        {
            StreamWriter writer = new StreamWriter(logFile, true);
            writer.WriteLine(DateTime.Now + " ERROR: " + error);
            writer.Close();
        }

        public void LogWarning(string warning)
        {
            StreamWriter writer = new StreamWriter(logFile, true);
            writer.WriteLine(DateTime.Now + " WARNING: " + warning);
            writer.Close();
        }

        public void ClearLog()
        {
            File.Delete(logFile);
        }

        public string ReadLog()
        {
            if(File.Exists(logFile))
            {
```

```
            StreamReader reader = new StreamReader(logFile);
            string content = reader.ReadToEnd();
            reader.Close();
            return content;
        }
        return "";
    }
  }
}
```

Вариант 9. Модуль конвертации данных. Проведите инспекцию следующего кода модуля конвертации данных:

```
using System;
using System.Collections.Generic;

namespace DataConverter
{
  public class Converter
  {
    public string ConvertToJSON(Dictionary<string, string> data)
    {
      string json = "{";
      int count = 0;
      foreach(var item in data)
      {
        count++;
        json = json + "\"" + item.Key + "\":\"" + item.Value + "\"";
        if(count < data.Count)
          json = json + ",";
      }
      json = json + "}";
      return json;
    }

    public Dictionary<string, string> ParseJSON(string json)
    {
      Dictionary<string, string> result = new Dictionary<string, string>();
      json = json.Replace("{", "").Replace("}", "");
      string[] pairs = json.Split(',');

      foreach(string pair in pairs)
      {
        string[] keyValue = pair.Split(':');
        string key = keyValue[0].Replace("\"", "").Trim();
        string value = keyValue[1].Replace("\"", "").Trim();
```

```
            result.Add(key, value);
        }

        return result;
    }

    public string ConvertToXML(List<string> items)
    {
        string xml = "<items>";
        for(int i = 0; i < items.Count; i++)
        {
            xml = xml + "<item>" + items[i] + "</item>";
        }
        xml = xml + "</items>";
        return xml;
    }

    public string ConvertToCSV(List<Dictionary<string, string>> data)
    {
        string csv = "";

        if(data.Count > 0)
        {
            foreach(var key in data[0].Keys)
            {
                csv = csv + key + ",";
            }
            csv = csv.TrimEnd(',') + "\n";

            foreach(var row in data)
            {
                foreach(var value in row.Values)
                {
                    csv = csv + value + ",";
                }
                csv = csv.TrimEnd(',') + "\n";
            }
        }

        return csv;
    }
  }
}
```

Вариант 10. Модуль кэширования. Проведите инспекцию сле-

дующего кода модуля кэширования:

```
using System;
using System.Collections.Generic;

namespace CachingModule
{
   public class Cache
   {
      private Dictionary<string, CacheItem> cache = new Dictionary<string, Cache-
Item>();
      private int maxSize = 100;

      public void Add(string key, object value, int expirationMinutes)
      {
         if(cache.Count >= maxSize)
         {
            ClearExpired();
         }

         CacheItem item = new CacheItem();
         item.Value = value;
         item.ExpirationTime = DateTime.Now.AddMinutes(expirationMinutes);

         if(cache.ContainsKey(key))
            cache[key] = item;
         else
            cache.Add(key, item);
      }

      public object Get(string key)
      {
         if(cache.ContainsKey(key))
         {
            CacheItem item = cache[key];
            if(item.ExpirationTime > DateTime.Now)
            {
               return item.Value;
            }
            else
            {
               cache.Remove(key);
               return null;
            }
         }
```

```
            return null;
        }

        public void ClearExpired()
        {
            List<string> keysToRemove = new List<string>();

            foreach(var item in cache)
            {
                if(item.Value.ExpirationTime <= DateTime.Now)
                {
                    keysToRemove.Add(item.Key);
                }
            }

            foreach(var key in keysToRemove)
            {
                cache.Remove(key);
            }
        }

        public void Clear()
        {
            cache.Clear();
        }
    }

    public class CacheItem
    {
        public object Value;
        public DateTime ExpirationTime;
    }
}
```

Требования к выполнению задания

1. Создание проекта в Visual Studio 2022: Создайте новый проект Console Application (.NET 6.0 или выше); Скопируйте код из вашего варианта; Убедитесь, что проект компилируется

2. Автоматический анализ кода

2.1. Включите Code Analysis: Правый клик на проекте → Properties; Перейдите в раздел Code Analysis; Установите Analysis level: All; Выполните сборку проекта; Зафиксируйте все предупреждения и ошибки анализатора

2.2. Рассчитайте метрики кода: Меню Analyze → Calculate Code Metrics (for Solution); Зафиксируйте значения метрик:
– Maintainability Index (индекс сопровождаемости)
– Cyclomatic Complexity (цикломатическая сложность)
– Depth of Inheritance (глубина наследования)
– Class Coupling (связанность классов)
– Lines of Code (строки кода)

2.3. Выполните Code Cleanup: Настройте профиль Code Clean-up (Tools → Options → Text Editor → Code Cleanup); Примените Code Cleanup к коду (Ctrl+K, Ctrl+E); Зафиксируйте изменения

3. Ручная инспекция кода. Проведите детальный анализ кода по следующим категориям:

3.1. Соответствие стандартам кодирования: Проверьте соблюдение соглашений об именовании; Оцените качество форматирования; Проверьте наличие и качество комментариев

3.2. Потенциальные ошибки: Обращение к null-объектам; Отсутствие обработки исключений; Утечки ресурсов (незакрытые соединения, файлы); Логические ошибки

3.3. Проблемы безопасности: SQL-инъекции; Хранение паролей в открытом виде; Отсутствие валидации входных данных; Раскрытие конфиденциальной информации

3.4. Проблемы производительности: Неэффективные алгоритмы; Избыточные операции; Неоптимальное использование коллекций; Дублирование вычислений

3.5. Нарушения принципов проектирования: Нарушение принципов SOLID; Дублирование кода (DRY); Высокая связанность; Низкая связность

4. Заполнение таблицы выявленных проблем

Для каждой выявленной проблемы заполните таблицу:

№	Строка кода	Категория проблемы	Описание проблемы	Уровень критичности	Рекомендация по исправлению
1					
2					

Уровни критичности:

– Критический — может привести к сбою или потере данных

– Высокий — существенно влияет на качество или безопасность
– Средний — ухудшает сопровождаемость или производительность
– Низкий — незначительные отклонения от стандартов
5. Исправление кода: Создайте улучшенную версию кода, устранив выявленные проблемы; Примените рефакторинг для улучшения структуры; Добавьте обработку исключений; Реализуйте валидацию входных данных; Улучшите именование и комментарии
6. Проверка улучшенного кода: Повторно выполните Code Analysis; Пересчитайте метрики кода; Сравните метрики до и после рефакторинга; Убедитесь, что все критические проблемы устранены

Оформление отчета. Отчет должен содержать:
1. Исходный код: Полный листинг кода из варианта задания
2. Результаты автоматического анализа: Скриншот окна Code Analysis с предупреждениями; Таблица метрик кода (до исправления):

Метрика	Значение	Оценка
Maintainability Index		
Cyclomatic Complexity		
Depth of Inheritance		
Class Coupling		
Lines of Code		

3. Результаты ручной инспекции: Заполненная таблица выявленных проблем (не менее 10 проблем); Подробное описание наиболее критичных проблем
4. Исправленный код: Полный листинг улучшенного кода; Комментарии к основным изменениям
5. Сравнительный анализ: Таблица метрик кода (после исправления); Сравнение метрик до и после; Список устраненных предупреждений Code Analysis
6. Выводы Какие типы проблем были выявлены; Какие инструменты оказались наиболее полезными; Как изменились метрики качества кода; Личные выводы о важности инспекции кода

Контрольные вопросы

1.Что такое инспекция кода и какие цели она преследует в процессе разработки программного обеспечения?

2.Перечислите основные виды инспекции кода. В чем различие между статическим и динамическим анализом?

3.Какие встроенные инструменты для инспекции кода предоставляет Visual Studio 2022? Опишите назначение каждого инструмента.

4.Что такое метрики кода? Расшифруйте следующие метрики: Maintainability Index, Cyclomatic Complexity, Class Coupling.

5.Какие значения индекса сопровождаемости (Maintainability Index) считаются приемлемыми? Что означают низкие значения этой метрики?

6.Перечислите основные соглашения об именовании в C#. Когда используется PascalCase, а когда camelCase?

7.Что такое принципы SOLID? Приведите примеры нарушения этих принципов.

8.Какие типичные проблемы безопасности можно выявить при инспекции кода? Приведите примеры.

9.Что такое SQL-инъекция? Как можно предотвратить эту уязвимость в коде?

10.Почему важно использовать конструкцию using при работе с ресурсами (файлами, соединениями с БД)? Что произойдет, если не закрыть ресурс?

11.Что такое цикломатическая сложность метода? Какие значения считаются приемлемыми?

12.Перечислите признаки некачественного кода, которые можно выявить при ручной инспекции.

13.Что такое Code Review? Какие преимущества дает командная проверка кода?

14.Как дублирование кода влияет на сопровождаемость программы? Какие техники рефакторинга помогают устранить дублирование?

15.Почему инспекция кода особенно важна при интеграции программных модулей?

Время выполнения: 2 ч

Практическая работа № 11 «Тестирование интерфейса пользователя средствами инструментальной среды разработки»

Цель работы: Освоить методы и средства тестирования интерфейса пользователя в инструментальной среде Visual Studio 2022, изучить принципы автоматизированного тестирования UI-элементов приложений на C#, научиться создавать и выполнять тесты пользовательского интерфейса для обеспечения качества программных модулей.

Теоретическая часть

Понятие тестирования пользовательского интерфейса. Тестирование пользовательского интерфейса (UI Testing) — это процесс проверки графических элементов программного обеспечения на соответствие требованиям и ожидаемому поведению. UI-тестирование позволяет убедиться, что все элементы интерфейса корректно отображаются, реагируют на действия пользователя и обеспечивают правильное взаимодействие с функциональной частью приложения.

Цели тестирования интерфейса пользователя

1.Проверка функциональности — убедиться, что элементы интерфейса выполняют предусмотренные функции

2.Проверка корректности отображения — проверить правильность расположения элементов, их размеры, цвета, шрифты

3.Проверка навигации — убедиться в корректности переходов между формами и элементами

4.Проверка обработки ошибок — протестировать отображение сообщений об ошибках и валидацию данных

5.Проверка доступности — убедиться, что интерфейс доступен для различных категорий пользователей

6.Проверка производительности UI — оценить скорость отклика интерфейса на действия пользователя

Виды тестирования пользовательского интерфейса

Ручное тестирование — тестировщик вручную взаимодействует с интерфейсом, проверяя его работу согласно тест-кейсам.

Автоматизированное тестирование — используются специализированные инструменты и фреймворки для автоматического выполнения тестов UI.

Методы тестирования UI в Visual Studio 2022

1. Coded UI Test (устаревший метод). Ранее использовался в Visual Studio для записи и воспроизведения действий пользователя. С версии Visual Studio 2019 объявлен устаревшим.

2. UI Automation с использованием библиотеки Windows UI Automation. Технология Microsoft для автоматизации тестирования приложений Windows Forms и WPF.

3. Unit-тестирование логики UI. Тестирование бизнес-логики, связанной с элементами интерфейса, без визуального отображения.

4. Использование сторонних фреймворков

- **WinAppDriver** — для тестирования Windows-приложений
- **Selenium** — для веб-приложений
- **Appium** — для мобильных приложений
- **FlaUI** — современная библиотека для UI Automation

Основные компоненты UI-тестирования в C#

Класс AutomationElement. Представляет элемент пользовательского интерфейса в дереве UI Automation.

```
using System.Windows.Automation;
AutomationElement element = AutomationElement.RootElement;
```

Поиск элементов интерфейса. Элементы можно находить по различным свойствам:

- **AutomationId** — уникальный идентификатор
- **Name** — имя элемента
- **ClassName** — класс элемента
- **ControlType** — тип элемента управления

Взаимодействие с элементами. После нахождения элемента можно выполнять различные действия: Клик по кнопкам; Ввод текста в текстовые поля; Выбор элементов в списках; Проверка состояния чекбоксов

Структура UI-теста. Типичный UI-тест состоит из следующих этапов:

1.Arrange (Подготовка) — запуск приложения, поиск необходимых элементов

2.Act (Действие) — выполнение действий над элементами интерфейса

3.Assert (Проверка) — проверка результатов действий

Использование MSTest для UI-тестирования. MSTest — это фреймворк для модульного тестирования от Microsoft, встроенный в Visual Studio.

Основные атрибуты:

- [TestClass] — помечает класс как содержащий тесты
- [TestMethod] — помечает метод как тестовый
- [TestInitialize] — метод выполняется перед каждым тестом
- [TestCleanup] — метод выполняется после каждого теста
- [ClassInitialize] — выполняется один раз перед всеми тестами класса
- [ClassCleanup] — выполняется один раз после всех тестов класса

Методы проверки (Assert):

- Assert.AreEqual(expected, actual) — проверка равенства
- Assert.IsTrue(condition) — проверка истинности условия
- Assert.IsFalse(condition) — проверка ложности условия
- Assert.IsNotNull(object) — проверка на ненулевое значение
- Assert.IsNull(object) — проверка на нулевое значение

Пример простого UI-теста для Windows Forms

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Windows.Automation;
using System.Diagnostics;

[TestClass]
public class CalculatorUITests
{
    private Process appProcess;
    private AutomationElement mainWindow;

    [TestInitialize]
    public void TestSetup()
    {
        // Запуск приложения
        appProcess = Process.Start("Calculator.exe");
        appProcess.WaitForInputIdle();
```

```
    // Получение главного окна
    mainWindow = AutomationElement.RootElement.FindFirst(
        TreeScope.Children,
        new PropertyCondition(AutomationElement.ProcessIdProperty,
                              appProcess.Id));
  }

  [TestMethod]
  public void TestButtonClick()
  {
    // Поиск кнопки
    var button = mainWindow.FindFirst(TreeScope.Descendants,
        new PropertyCondition(AutomationElement.AutomationIdProperty,
                              "btnCalculate"));

    // Клик по кнопке
    var invokePattern = button.GetCurrentPattern(InvokePattern.Pattern)
                        as InvokePattern;
    invokePattern.Invoke();

    // Проверка результата
    var resultLabel = mainWindow.FindFirst(TreeScope.Descendants,
        new PropertyCondition(AutomationElement.AutomationIdProperty,
                              "lblResult"));

    var textPattern = resultLabel.GetCurrentPattern(TextPattern.Pattern)
                      as TextPattern;
    Assert.AreEqual("Ожидаемый результат",
                    textPattern.DocumentRange.GetText(-1));
  }

  [TestCleanup]
  public void TestTearDown()
  {
    // Закрытие приложения
    if (appProcess != null && !appProcess.HasExited)
    {
      appProcess.Kill();
      appProcess.WaitForExit();
    }
  }
}
```

Тестирование элементов управления Windows Forms

Установка AutomationId для элементов. Для облегчения поиска элементов в тестах необходимо устанавливать свойство Name элементов управления в конструкторе форм или программно:

```
textBoxName.Name = "txtName";
buttonSave.Name = "btnSave";
labelResult.Name = "lblResult";
```

Получение и установка значений текстовых полей

```
// Поиск текстового поля
var textBox = mainWindow.FindFirst(TreeScope.Descendants,
    new PropertyCondition(AutomationElement.AutomationIdProperty, "txt-
Name"));

// Получение текста
var valuePattern = textBox.GetCurrentPattern(ValuePattern.Pattern)
                as ValuePattern;
string currentValue = valuePattern.Current.Value;

// Установка текста
valuePattern.SetValue("Новое значение");
```

Работа с кнопками

```
var button = mainWindow.FindFirst(TreeScope.Descendants,
    new PropertyCondition(AutomationElement.AutomationIdProperty, "btnSave"));

var invokePattern = button.GetCurrentPattern(InvokePattern.Pattern)
                as InvokePattern;
invokePattern.Invoke();
```

Работа с чекбоксами

```
var checkBox = mainWindow.FindFirst(TreeScope.Descendants,
    new PropertyCondition(AutomationElement.AutomationIdProperty,
                    "chkAgree"));

var togglePattern = checkBox.GetCurrentPattern(TogglePattern.Pattern)
                as TogglePattern;

// Установка состояния
togglePattern.Toggle();

// Проверка состояния
ToggleState state = togglePattern.Current.ToggleState;
Assert.AreEqual(ToggleState.On, state);
```

Лучшие практики UI-тестирования

1.Используйте уникальные идентификаторы — устанав-

ливайте AutomationId (свойство Name) для всех тестируемых элементов
2.Делайте тесты независимыми — каждый тест должен работать независимо от других
3.Используйте явные ожидания — добавляйте паузы или ожидания для загрузки элементов
4.Тестируйте реальные сценарии — создавайте тесты, имитирующие действия пользователей
5.Документируйте тесты — добавляйте комментарии, объясняющие логику теста
6.Обрабатывайте исключения — предусматривайте обработку возможных ошибок
7.Очищайте ресурсы — всегда закрывайте приложения и освобождайте ресурсы после тестов
Отчет о тестировании. Результаты тестирования UI должны включать: Общее количество тестов; Количество пройденных тестов; Количество проваленных тестов; Описание ошибок для проваленных тестов; Время выполнения тестов; Скриншоты (при необходимости)
Visual Studio автоматически генерирует отчет о результатах тестирования в окне Test Explorer.

Рабочее задание: Выполните тестирование интерфейса пользователя приложения в соответствии с вариантом задания. Создайте Windows Forms приложение на C#, реализуйте функциональность согласно варианту, разработайте набор автоматизированных UI-тестов для проверки корректности работы интерфейса.
Вариант 1. Калькулятор простых операций: Создать приложение-калькулятор для выполнения четырех арифметических операций (+, -, *, /). Требования к интерфейсу:

- Два текстовых поля для ввода чисел (txtNumber1, txtNumber2)
- Четыре кнопки для операций (btnAdd, btnSubtract, btnMultiply, btnDivide)
- Метка для вывода результата (lblResult)
- Кнопка очистки (btnClear)

Тесты, которые необходимо реализовать:
1.Тест сложения двух положительных чисел; 2. Тест вычитания чисел; 3.Тест умножения чисел; 4.Тест деления чисел; 5.Тест деления на ноль (должно выводиться сообщение об ошибке); 6.Тест функции очистки; 7.Тест ввода некорректных данных (буквы вместо чисел)

Вариант 2. Конвертер температуры: Создать приложение для конвертации температуры между шкалами Цельсия, Фаренгейта и Кельвина. Требования к интерфейсу:

- Текстовое поле для ввода температуры (txtTemperature)
- Три радиокнопки для выбора исходной шкалы (rbCelsius, rbFahrenheit, rbKelvin)
- Три радиокнопки для выбора целевой шкалы (rbToCelsius, rbToFahrenheit, rbToKelvin)
- Кнопка конвертации (btnConvert)
- Метка для вывода результата (lblResult)

Тесты, которые необходимо реализовать:
1.Тест конвертации из Цельсия в Фаренгейт; 2. Тест конвертации из Фаренгейта в Цельсий; 3.Тест конвертации из Цельсия в Кельвин; 4.Тест конвертации из Кельвина в Цельсий; 5.Тест конвертации одинаковых шкал; 6.Тест ввода некорректных данных; 7.Тест граничных значений (абсолютный ноль)

Вариант 3. Регистрационная форма: Создать форму регистрации пользователя с валидацией данных. Требования к интерфейсу:

- Текстовое поле для имени (txtName)
- Текстовое поле для email (txtEmail)
- Текстовое поле для пароля (txtPassword)
- Текстовое поле для подтверждения пароля (txtConfirmPassword)
- Чекбокс согласия с условиями (chkAgree)
- Кнопка регистрации (btnRegister)
- Метка для вывода статуса (lblStatus)

Тесты, которые необходимо реализовать:
1.Тест успешной регистрации с корректными данными; 2.

Тест валидации пустых полей; 3.Тест валидации некорректного email; 4.Тест несовпадения паролей; 5.Тест регистрации без согласия с условиями; 6.Тест минимальной длины пароля (минимум 6 символов); 7.Тест активации кнопки регистрации

Вариант 4. Список задач (To-Do List): Создать приложение для управления списком задач. Требования к интерфейсу:

- Текстовое поле для ввода задачи (txtTask)
- Кнопка добавления задачи (btnAdd)
- Список задач ListBox (lstTasks)
- Кнопка удаления выбранной задачи (btnDelete)
- Кнопка очистки всех задач (btnClearAll)
- Метка с количеством задач (lblCount)

Тесты, которые необходимо реализовать:
1.Тест добавления новой задачи; 2. Тест добавления нескольких задач; 3. Тест удаления выбранной задачи; 4. Тест очистки всего списка; 5.Тест добавления пустой задачи (не должно добавляться); 6.Тест корректности счетчика задач; 7Тест удаления без выбора задачи

Вариант 5. Калькулятор скидки: Создать приложение для расчета стоимости товара с учетом скидки. Требования к интерфейсу:

- Текстовое поле для ввода цены (txtPrice)
- Текстовое поле для ввода процента скидки (txtDiscount)
- Кнопка расчета (btnCalculate)
- Метка для отображения суммы скидки (lblDiscountAmount)
- Метка для отображения итоговой цены (lblFinalPrice)
- Кнопка очистки (btnClear)

Тесты, которые необходимо реализовать:
1.Тест расчета скидки 10%; 2. Тест расчета скидки 50%; 3.Тест расчета без скидки (0%); 4.Тест расчета со скидкой 100%; 5.Тест ввода отрицательной цены; 6.Тест ввода скидки больше 100%; 7.Тест функции очистки полей

Вариант 6. Конвертер валют: Создать приложение для

конвертации между тремя валютами (рубль, доллар, евро). Требования к интерфейсу:

- Текстовое поле для ввода суммы (txtAmount)
- ComboBox для выбора исходной валюты (cmbFrom)
- ComboBox для выбора целевой валюты (cmbTo)
- Кнопка конвертации (btnConvert)
- Метка для отображения результата (lblResult)
- Метка для отображения курса (lblRate)

Тесты, которые необходимо реализовать:
1.Тест конвертации из рублей в доллары; 2.Тест конвертации из долларов в евро; 3.Тест конвертации из евро в рубли; 4.Тест конвертации одинаковых валют; 5.Тест ввода нулевой суммы; 6.Тест ввода отрицательной суммы; 7.Тест отображения актуального курса

Вариант 7. Калькулятор ИМТ (индекс массы тела): Создать приложение для расчета индекса массы тела и определения категории веса. Требования к интерфейсу:

- Текстовое поле для ввода роста в см (txtHeight)
- Текстовое поле для ввода веса в кг (txtWeight)
- Кнопка расчета (btnCalculate)
- Метка для отображения ИМТ (lblBMI)
- Метка для отображения категории (lblCategory)
- Кнопка очистки (btnClear)

Тесты, которые необходимо реализовать:
1.Тест расчета ИМТ для нормального веса; 2.Тест расчета ИМТ для недостаточного веса; 3.Тест расчета ИМТ для избыточного веса; 4.Тест расчета ИМТ для ожирения; 5.Тест ввода некорректного роста (0 или отрицательное); 6.Тест ввода некорректного веса; 7.Тест правильности определения категории

Вариант 8. Генератор паролей: Создать приложение для генерации случайных паролей с настраиваемыми параметрами. Требования к интерфейсу:

- Текстовое поле для указания длины пароля (txtLength)
- Чекбокс для включения цифр (chkDigits)
- Чекбокс для включения специальных символов (chkSpecial)
- Чекбокс для включения заглавных букв (chkUppercase)

- Кнопка генерации (btnGenerate)
- Текстовое поле для отображения пароля (txtPassword)
- Кнопка копирования в буфер обмена (btnCopy)

Тесты, которые необходимо реализовать:
1.Тест генерации пароля заданной длины; 2. Тест генерации пароля только из букв; 3. Тест генерации пароля с цифрами; 4. Тест генерации пароля со спецсимволами; 5.Тест генерации пароля с заглавными буквами; 6.Тест минимальной длины пароля (4 символа); 7.Тест максимальной длины пароля (20 символов)

Вариант 9. Таймер обратного отсчета: Создать приложение-таймер с возможностью установки времени обратного отсчета. Требования к интерфейсу:

- Текстовое поле для ввода минут (txtMinutes)
- Текстовое поле для ввода секунд (txtSeconds)
- Кнопка старта (btnStart)
- Кнопка паузы (btnPause)
- Кнопка сброса (btnReset)
- Метка для отображения времени (lblTimer)
- ProgressBar для визуализации (progressBar)

Тесты, которые необходимо реализовать:
1.Тест установки времени; 2. Тест запуска таймера; 3. Тест паузы таймера; 4. Тест сброса таймера; 5.Тест завершения отсчета; 6.Тест ввода некорректного времени; 7.Тест повторного запуска после завершения

Вариант 10. Калькулятор площади фигур: Создать приложение для расчета площади различных геометрических фигур. Требования к интерфейсу:

- ComboBox для выбора фигуры (cmbShape): круг, квадрат, прямоугольник, треугольник
- Текстовые поля для параметров (txtParam1, txtParam2) — отображаются в зависимости от фигуры
- Метки для обозначения параметров (lblParam1, lblParam2)
- Кнопка расчета (btnCalculate)
- Метка для отображения результата (lblResult)

Тесты, которые необходимо реализовать:
1.Тест расчета площади круга; 2. Тест расчета площади квадрата; 3. Тест расчета площади прямоугольника; 4. Тест расчета

площади треугольника; 5.Тест изменения доступных полей при смене фигуры; 6.Тест ввода отрицательных значений; 7.Тест ввода нулевых значений

Требования к выполнению задания

1.Создайте Windows Forms приложение: Разработайте интерфейс согласно требованиям варианта; Установите свойство Name для всех элементов управления согласно указанным идентификаторам; Реализуйте функциональность приложения

2.Создайте проект модульного тестирования: Добавьте в решение новый проект типа "Unit Test Project"; Добавьте ссылку на основной проект приложения; Добавьте необходимые пространства имен:

– using System. Windows. Automation;
– using System.Diagnostics;
– using Microsoft.VisualStudio. TestTools.UnitTesting;

3.Разработайте автоматизированные UI-тесты: Создайте не менее 7 тестовых методов согласно варианту; Каждый тест должен содержать комментарии с описанием проверяемого сценария; Используйте методы Assert для проверки результатов; Реализуйте методы инициализации и очистки (TestInitialize, TestCleanup)

4.Запустите тесты и проанализируйте результаты: Выполните все тесты через Test Explorer; Убедитесь, что все тесты проходят успешно; При необходимости исправьте ошибки в коде приложения или тестов

5.Оформите отчет, содержащий: Описание разработанного приложения; Скриншоты интерфейса приложения; Код основного приложения (ключевые фрагменты); Код тестовых классов (полностью); Скриншот окна Test Explorer с результатами тестирования; Описание каждого теста и его назначение; Выводы по работе

Контрольные вопросы

1.Что такое тестирование пользовательского интерфейса и каковы его основные цели?

2.Какие виды UI-тестирования существуют? В чем преимущества автоматизированного тестирования перед ручным?

3.Что представляет собой технология UI Automation? Какие основные классы и интерфейсы она предоставляет?

4.Объясните назначение класса AutomationElement. Какие способы поиска элементов интерфейса вы знаете?

5.Какие атрибуты используются в MSTest для организации тестов? Объясните назначение атрибутов [TestInitialize] и [TestCleanup].

6.Какие методы Assert используются для проверки результатов в модульных тестах? Приведите примеры использования.

7.Как получить доступ к свойствам элементов управления (текст, состояние, видимость) через UI Automation?

8.Как выполнить клик по кнопке или ввод текста в текстовое поле с использованием UI Automation?

9.Какие паттерны взаимодействия (Patterns) предоставляет UI Automation для различных типов элементов управления?

10.Почему важно устанавливать свойство Name (AutomationId) для элементов управления при разработке приложений, которые будут тестироваться?

11.Какие лучшие практики следует соблюдать при разработке UI-тестов?

12.Как интерпретировать результаты тестирования в окне Test Explorer Visual Studio?

Время выполнения: 4 ч

Практическая работа № 12 «Разработка тестовых модулей проекта для тестирования отдельных модулей»

Цель работы: освоить методы разработки модульных тестов для проверки корректности работы отдельных программных модулей, изучить принципы использования фреймворков для модульного тестирования в среде visual studio 2022 на языке c#, научиться создавать тест-кейсы и анализировать результаты тестирования.

Теоретическая часть

Модульное тестирование. Модульное тестирование (Unit Testing) — это процесс тестирования отдельных модулей, компонентов или функций программного обеспечения изолированно от остальной системы. Основная цель модульного тестирования — проверить, что каждый модуль работает корректно и выполняет свои функции в соответствии с требованиями.

Преимущества модульного тестирования:

- Раннее обнаружение ошибок на этапе разработки
- Упрощение отладки (легче локализовать проблему)
- Документирование поведения модулей
- Упрощение рефакторинга кода
- Повышение качества кода
- Снижение стоимости исправления дефектов

Принципы модульного тестирования. **FIRST-принципы:**

Fast (Быстрые) — тесты должны выполняться быстро

Independent (Независимые) — тесты не должны зависеть друг от друга

Repeatable (Повторяемые) — тесты должны давать одинаковый результат при повторном запуске

Self-Validating (Самопроверяемые) — тест сам определяет успех или неудачу

Timely (Своевременные) — тесты пишутся до или вместе с кодом

Фреймворки для модульного тестирования в C#

MSTest — встроенный фреймворк Microsoft для модульного тестирования, интегрированный в Visual Studio.

NUnit — популярный открытый фреймворк для .NET, портированный с JUnit.

xUnit — современный фреймворк, разработанный создателями NUnit с улучшенной архитектурой.

Структура модульного теста. Типичный модульный тест состоит из трех частей (паттерн AAA):

1.Arrange (Подготовка) — создание и настройка тестовых данных и объектов

2.Act (Действие) — выполнение тестируемого метода

3.Assert (Проверка) — проверка результата

```
[TestMethod]
public void Add_TwoPositiveNumbers_ReturnsCorrectSum()
{
```

```
    // Arrange
    Calculator calc = new Calculator();
    int a = 5;
    int b = 3;

    // Act
    int result = calc.Add(a, b);

    // Assert
    Assert.AreEqual(8, result);
}
```

Основные атрибуты MSTest

[TestClass] — отмечает класс, содержащий тесты
[TestMethod] — отмечает метод как тестовый
[TestInitialize] — метод выполняется перед каждым тестом
[TestCleanup] — метод выполняется после каждого теста
[ClassInitialize] — метод выполняется один раз перед всеми тестами класса
[ClassCleanup] — метод выполняется один раз после всех тестов класса
[DataRow] — позволяет передавать параметры в тестовый метод

Методы Assert

Assert.AreEqual(expected, actual) — проверяет равенство значений
Assert.AreNotEqual(notExpected, actual) — проверяет неравенство
Assert.IsTrue(condition) — проверяет, что условие истинно
Assert.IsFalse(condition) — проверяет, что условие ложно
Assert.IsNull(object) — проверяет, что объект равен null
Assert.IsNotNull(object) — проверяет, что объект не равен null
Assert.ThrowsException() — проверяет, что выбрасывается исключение

Покрытие кода тестами. Покрытие кода (Code Coverage) — метрика, показывающая, какая часть кода выполняется при запуске тестов. Измеряется в процентах**.** Рекомендуемое покрытие для критически важного кода — не менее 80 %.

Создание тестового проекта в Visual Studio 2022

1.В Solution Explorer щелкните правой кнопкой мыши по решению → Add → New Project

2.Выберите шаблон "MSTest Test Project (.NET Core)" или "Unit

Test Project"

3.Назовите проект (например, "ProjectName.Tests")

4.Добавьте ссылку на тестируемый проект

Запуск тестов в Visual Studio

– **Test → Run All Tests** — запуск всех тестов

– **Test Explorer** (Test → Test Explorer) — панель управления тестами

– Зеленая галочка — тест пройден

- Красный крестик — тест не пройден

Тестирование различных типов модулей

Тестирование бизнес-логики:

```
[TestMethod]
public void CalculateDiscount_Premium_Returns20Percent()
{
    var service = new DiscountService();
    decimal result = service.CalculateDiscount(100, CustomerType.Premium);
    Assert.AreEqual(20, result);
}
```

Тестирование валидации:

```
[TestMethod]
[ExpectedException(typeof(ArgumentException))]
public void ValidateEmail_InvalidFormat_ThrowsException()
{
    var validator = new EmailValidator();
    validator.Validate("invalid-email");
}
```

Тестирование работы с данными:

```
[TestMethod]
public void GetUserById_ExistingId_ReturnsUser()
{
    var repository = new UserRepository();
    var user = repository.GetById(1);
    Assert.IsNotNull(user);
    Assert.AreEqual("John", user.FirstName);
}
```

Именование тестовых методов Рекомендуемый формат:

MethodName_Scenario_ExpectedBehavior

Примеры: Add_TwoPositiveNumbers_ReturnsSum; Divide_ByZero_ThrowsException; GetUser_InvalidId_ReturnsNull

Рабочее задание: Разработайте набор модульных тестов для тестирования программных модулей в соответствии с вариантом задания**.**

Вариант 1. Система управления библиотекой. Разработать класс **BookManager** для управления книгами в библиотеке со следующими методами:
– AddBook(Book book) — добавление книги
– RemoveBook(int id) — удаление книги по ID
– FindBookByTitle(string title) — поиск книги по названию
– GetAvailableBooks() — получение списка доступных книг
– IssueBook(int bookId, int userId) — выдача книги пользователю
– ReturnBook(int bookId) — возврат книги
Создать тестовый проект и написать не менее 10 тестовых методов, покрывающих различные сценарии работы (позитивные, негативные, граничные условия).

Вариант 2. Калькулятор арифметических операций. Разработать класс **Calculator** с методами:
– Add(double a, double b) — сложение
– Subtract(double a, double b) — вычитание
– Multiply(double a, double b) — умножение
– Divide(double a, double b) — деление
– Power(double x, double y) — возведение в степень
– SquareRoot(double x) — квадратный корень
Создать тестовый проект и написать не менее 12 тестовых методов, включая тесты на исключения (деление на ноль, корень из отрицательного числа).

Вариант 3. Валидатор пользовательских данных. Разработать класс **UserValidator** с методами:
–ValidateEmail(string email) — проверка корректности email
–ValidatePassword(string password) — проверка надежности пароля (минимум 8 символов, буквы и цифры)
–ValidatePhone(string phone) — проверка формата телефона
–ValidateAge(int age) — проверка возраста (от 18 до 100)
–ValidateUsername(string username) — проверка имени пользователя (3-20 символов, буквы и цифры)
Создать тестовый проект и написать не менее 15 тестовых методов для различных валидных и невалидных входных данных.

Вариант 4. Система скидок для интернет-магазина. Разработать класс **DiscountCalculator** с методами:
– CalculateDiscount(decimal amount, CustomerType type) — расчет скидки по типу клиента
– ApplyPromoCode(decimal amount, string promoCode) — применение промокода
– CalculateFinalPrice(decimal amount, decimal discountPercent) — расчет итоговой цены
– IsEligibleForDiscount(Customer customer) — проверка права на скидку
– GetSeasonalDiscount(DateTime date) — получение сезонной скидки
Создать тестовый проект и написать не менее 12 тестовых методов для различных комбинаций скидок и типов клиентов.

Вариант 5. Банковский счет. Разработать класс **BankAccount** с методами:
– Deposit(decimal amount) — пополнение счета
– Withdraw(decimal amount) — снятие средств
– GetBalance() — получение баланса
– Transfer(BankAccount target, decimal amount) — перевод на другой счет
– CalculateInterest(int days) — расчет процентов за указанное количество дней
– Close() — закрытие счета
Создать тестовый проект и написать не менее 13 тестовых методов, включая проверку на недостаточность средств, отрицательные суммы.

Вариант 6. Система расписания занятий. Разработать класс **ScheduleManager** с методами:
– AddLesson(Lesson lesson) — добавление занятия
– RemoveLesson(int lessonId) — удаление занятия
– GetLessonsByGroup(string groupName) — получение расписания группы
– GetLessonsByTeacher(string teacherName) — получение расписания преподавателя
– CheckRoomAvailability(string room, DateTime time) — проверка доступности аудитории

– HasConflict(Lesson lesson) — проверка конфликтов в расписании
Создать тестовый проект и написать не менее 11 тестовых методов для различных сценариев работы с расписанием.

Вариант 7. Конвертер единиц измерения. Разработать класс **UnitConverter** с методами:
– KilometersToMiles(double km) — км в мили
– MilesToKilometers(double miles) — мили в км
– CelsiusToFahrenheit(double celsius) — Цельсий в Фаренгейт
– FahrenheitToCelsius(double fahrenheit) — Фаренгейт в Цельсий
– KilogramsToOunces(double kg) — кг в унции
– LitersToGallons(double liters) — литры в галлоны
Создать тестовый проект и написать не менее 10 тестовых методов с проверкой точности конвертации.

Вариант 8. Менеджер задач (To-Do List). Разработать класс **TaskManager** с методами:
– AddTask(Task task) — добавление задачи
– RemoveTask(int taskId) — удаление задачи
– CompleteTask(int taskId) — отметка задачи как выполненной
– GetPendingTasks() — получение невыполненных задач
– GetTasksByPriority(Priority priority) — получение задач по приоритету
– GetOverdueTasks() — получение просроченных задач
Создать тестовый проект и написать не менее 12 тестовых методов для различных операций с задачами.

Вариант 9. Система учета товаров на складе. Разработать класс **InventoryManager** с методами:
– AddProduct(Product product, int quantity) — добавление товара
– RemoveProduct(int productId, int quantity) — списание товара
– GetProductQuantity(int productId) — получение количества товара
– IsInStock(int productId) — проверка наличия товара
– GetLowStockProducts(int threshold) — получение товаров с низким остатком
– CalculateTotalValue() — расчет общей стоимости товаров на складе
Создать тестовый проект и написать не менее 11 тестовых мето-

дов, включая граничные случаи.

Вариант 10. Анализатор текста. Разработать класс **TextAnalyzer** с методами:
– CountWords(string text) — подсчет количества слов
– CountCharacters(string text) — подсчет количества символов
– CountVowels(string text) — подсчет гласных букв
– CountConsonants(string text) — подсчет согласных букв
– IsPalindrome(string text) — проверка, является ли текст палиндромом
– GetMostFrequentWord(string text) — поиск самого частого слова
Создать тестовый проект и написать не менее 13 тестовых методов для различных типов текста.

Требования к выполнению задания

1.Создайте основной проект: Создайте консольное приложение или библиотеку классов; Реализуйте класс и методы согласно варианту задания; Убедитесь, что код компилируется без ошибок
2.Создайте тестовый проект: Добавьте в решение тестовый проект MSTest; Добавьте ссылку на основной проект; Создайте тестовый класс с атрибутом [TestClass]
3.Напишите модульные тесты: Создайте не менее указанного количества тестовых методов; Используйте атрибут [TestMethod] для каждого теста; Следуйте паттерну AAA (Arrange-Act-Assert); Используйте осмысленные имена тестов
4.Покройте различные сценарии: Позитивные тесты (корректные входные данные); Негативные тесты (некорректные данные, исключения); Граничные условия (минимальные, максимальные значения, пустые коллекции); Тесты на исключения (используйте Assert.ThrowsException)
5.Запустите тесты: Убедитесь, что все тесты проходят успешно; Сделайте скриншоты Test Explorer с результатами
6.Оформите отчет, содержащий: Описание разработанного класса и его методов; Листинг кода основного класса; Листинг кода тестового класса; Скриншоты результатов выполнения тестов в Test Explorer; Таблицу с описанием каждого теста (имя теста, что проверяет, результат); Выводы по работе

Пример выполнения

Основной класс (StringHelper.cs):

```
namespace StringUtilities
{
    public class StringHelper
    {
        public string Reverse(string input)
        {
            if (input == null)
                throw new ArgumentNullException(nameof(input));

            char[] chars = input.ToCharArray();
            Array.Reverse(chars);
            return new string(chars);
        }

        public bool IsPalindrome(string input)
        {
            if (string.IsNullOrEmpty(input))
                return false;

            string reversed = Reverse(input);
            return input.Equals(reversed, StringComparison.OrdinalIgnoreCase);
        }

        public int CountOccurrences(string text, char character)
        {
            if (text == null)
                throw new ArgumentNullException(nameof(text));

            return text.Count(c => c == character);
        }
    }
}
```

Тестовый класс (StringHelperTests.cs):

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using StringUtilities;

namespace StringUtilities.Tests
{
    [TestClass]
    public class StringHelperTests
    {
        private StringHelper _helper;
```

```
[TestInitialize]
public void Setup()
{
    _helper = new StringHelper();
}

[TestMethod]
public void Reverse_ValidString_ReturnsReversedString()
{
    // Arrange
    string input = "hello";

    // Act
    string result = _helper.Reverse(input);

    // Assert
    Assert.AreEqual("olleh", result);
}

[TestMethod]
public void Reverse_EmptyString_ReturnsEmptyString()
{
    // Arrange
    string input = "";

    // Act
    string result = _helper.Reverse(input);

    // Assert
    Assert.AreEqual("", result);
}

[TestMethod]
[ExpectedException(typeof(ArgumentNullException))]
public void Reverse_NullString_ThrowsArgumentNullException()
{
    // Act
    _helper.Reverse(null);
}

[TestMethod]
public void IsPalindrome_ValidPalindrome_ReturnsTrue()
{
```

```
    // Arrange
    string input = "radar";

    // Act
    bool result = _helper.IsPalindrome(input);

    // Assert
    Assert.IsTrue(result);
}

[TestMethod]
public void IsPalindrome_NotPalindrome_ReturnsFalse()
{
    // Arrange
    string input = "hello";

    // Act
    bool result = _helper.IsPalindrome(input);

    // Assert
    Assert.IsFalse(result);
}

[TestMethod]
public void CountOccurrences_CharacterExists_ReturnsCorrectCount()
{
    // Arrange
    string text = "hello world";
    char character = 'l';

    // Act
    int result = _helper.CountOccurrences(text, character);

    // Assert
    Assert.AreEqual(3, result);
}

[TestMethod]
public void CountOccurrences_CharacterNotExists_ReturnsZero()
{
    // Arrange
    string text = "hello";
    char character = 'z';
```

```
            // Act
            int result = _helper.CountOccurrences(text, character);

            // Assert
            Assert.AreEqual(0, result);
        }
    }
}
```

Контрольные вопросы

1.Что такое модульное тестирование? Каковы основные цели и преимущества модульного тестирования в процессе разработки программного обеспечения?

2.Опишите структуру модульного теста по паттерну AAA. Что включает в себя каждая из трех частей?

3.Какие основные атрибуты используются в фреймворке MSTest? Для чего предназначены атрибуты [TestClass], [TestMethod], [TestInitialize]?

4.Перечислите основные методы класса Assert и опишите их назначение. Когда следует использовать Assert.AreEqual, а когда Assert.IsTrue?

5.Что такое покрытие кода тестами (Code Coverage)? Какой процент покрытия считается приемлемым для критически важных модулей?

6.В чем разница между позитивными и негативными тестами? Приведите примеры каждого типа.

7.Какие принципы FIRST должны соблюдаться при написании модульных тестов? Почему независимость тестов важна?

8.Как протестировать метод, который должен выбросить исключение? Какой атрибут или метод Assert для этого используется?

9.Для чего используются методы, помеченные атрибутами [TestInitialize] и [TestCleanup]? Приведите примеры ситуаций, когда они необходимы.

10.Почему важно использовать осмысленные имена для тестовых методов? Какой формат именования рекомендуется использовать?

Время выполнения: 2 ч

Практическая работа № 13 «Выполнение функционального тестирования»

Цель работы: Изучить методологию функционального тестирования программных модулей, освоить техники разработки тест-кейсов, научиться применять инструменты тестирования в среде Visual Studio 2022 для проверки корректности работы программных модулей на языке C#.

Теоретическая часть

Понятие функционального тестирования. Функциональное тестирование — это процесс проверки соответствия программного обеспечения функциональным требованиям и спецификациям. Основная цель — убедиться, что программа выполняет все заявленные функции корректно. Функциональное тестирование отвечает на вопрос: **«Что делает система?»**, в отличие от нефункционального тестирования, которое отвечает на вопрос: **«Как система это делает?»**.

Виды функционального тестирования

1.Модульное тестирование (Unit Testing) — тестирование отдельных программных модулей, классов, методов в изоляции от остальной системы.

2.Интеграционное тестирование (Integration Testing**)** — проверка взаимодействия между интегрированными модулями или компонентами системы.

3.Системное тестирование (System Testing) — тестирование системы в целом на соответствие требованиям.

4.Приемочное тестирование (Acceptance Testing) — проверка готовности системы к передаче заказчику.

Методы функционального тестирования. **Тестирование по принципу «черного ящика»** — тестирование без знания внутренней структуры программы, основываясь только на спецификации. Основные техники:

- Эквивалентное разбиение — разделение входных данных на классы эквивалентности
- Граничные значения — тестирование на границах допустимых диапазонов
- Таблицы принятия решений — систематическое тестиро-

вание комбинаций условий
- Попарное тестирование — проверка всех возможных пар входных параметров

Структура тест-кейса. Тест-кейс (Test Case) — это документ, описывающий набор действий для проверки определенной функциональности. Структура тест-кейса:
- **ID тест-кейса** — уникальный идентификатор
- **Название** — краткое описание проверяемой функции
- **Предусловия** — требования, которые должны быть выполнены до начала теста
- **Шаги выполнения** — последовательность действий
- **Тестовые данные** — входные данные для теста
- **Ожидаемый результат** — описание корректного поведения системы
- **Фактический результат** — реальный результат выполнения теста
- **Статус** — Passed (пройден) / Failed (провален)

Модульное тестирование в C# с использованием MSTest.
MSTest — это фреймворк модульного тестирования от Microsoft, интегрированный в Visual Studio.

Основные атрибуты MSTest:

```
[TestClass]    // Указывает, что класс содержит тесты
[TestMethod]   // Отмечает метод как тестовый
[TestInitialize]   // Выполняется перед каждым тестом
[TestCleanup]      // Выполняется после каждого теста
[ClassInitialize]  // Выполняется один раз перед всеми тестами класса
[ClassCleanup]     // Выполняется один раз после всех тестов класса
```

Методы Assert:

```
Assert.AreEqual(expected, actual);        // Проверка равенства
Assert.AreNotEqual(expected, actual);     // Проверка неравенства
Assert.IsTrue(condition);                 // Проверка истинности
Assert.IsFalse(condition);                // Проверка ложности
Assert.IsNull(object);                   // Проверка на null
Assert.IsNotNull(object);                 // Проверка на not null
Assert.ThrowsException<T>();              // Проверка выброса исключения
```

Пример создания тестового проекта в Visual Studio 2022

Шаг 1. Создание тестового проекта:
- Правой кнопкой на решение → Добавить → Создать проект

- Выбрать шаблон "Проект модульного тестирования MSTest"
- Указать имя проекта (например, "ProjectName.Tests")

Шаг 2. Добавление ссылки на тестируемый проект:

- Правой кнопкой на Dependencies тестового проекта → Добавить ссылку на проект
- Выбрать основной проект

Шаг 3. Создание тестового класса:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Calculator.Tests
{
   [TestClass]
   public class CalculatorTests
   {
      [TestMethod]
      public void Add_TwoPositiveNumbers_ReturnsCorrectSum()
      {
         // Arrange (Подготовка)
         var calculator = new Calculator();
         int a = 5;
         int b = 3;
         int expected = 8;

         // Act (Действие)
         int actual = calculator.Add(a, b);

         // Assert (Проверка)
         Assert.AreEqual(expected, actual);
      }
   }
}
```

Паттерн AAA (Arrange-Act-Assert). Рекомендуемая структура тестовых методов:

1.Arrange (Подготовка) — инициализация объектов, подготовка данных

2.Act (Действие) — выполнение тестируемого действия

3.Assert (Проверка) — проверка результата

Покрытие кода тестами. Покрытие кода (Code Coverage) — метрика, показывающая процент кода, который выполняется во время тестирования. Просмотр покрытия кода в Visual Studio:

Тест → Анализ покрытия кода для всех тестов
Рекомендуемое покрытие кода тестами: не менее 70-80% для критически важных модулей.
Тестирование граничных условий
Особое внимание следует уделять тестированию граничных значений: Минимальные и максимальные значения; Нулевые значения; Пустые коллекции и строки; Значения на границах диапазонов; Специальные символы
Тестирование исключений. Пример проверки выброса исключения:

```
[TestMethod]
[ExpectedException(typeof(DivideByZeroException))]
public void Divide_ByZero_ThrowsException()
{
    var calculator = new Calculator();
    calculator.Divide(10, 0);
}
```

Или с использованием Assert.ThrowsException:

```
[TestMethod]
public void Divide_ByZero_ThrowsException()
{
    var calculator = new Calculator();
    Assert.ThrowsException<DivideByZeroException>(() =>
        calculator.Divide(10, 0));
}
```

Рабочее задание: Выполните функциональное тестирование программного модуля в соответствии с вариантом задания. Разработайте тест-кейсы и реализуйте модульные тесты с использованием фреймворка MSTest в Visual Studio 2022.
Вариант 1. Тестирование класса "Калькулятор". Разработать класс **Calculator** с методами выполнения арифметических операций и комплект тестов для его проверки.
Требуемые методы класса:
– Add(int a, int b) — сложение
– Subtract(int a, int b) — вычитание
– Multiply(int a, int b) — умножение
– Divide(double a, double b) — деление (с проверкой деления на ноль)
– Power(double a, int n) — возведение в степень

– Factorial(int n) — вычисление факториала (для n >= 0)

Требования к тестированию: Разработать не менее 15 тест-кейсов; Протестировать все методы класса; Включить проверку граничных условий; Проверить обработку исключительных ситуаций; Достичь покрытия кода не менее 90%

Вариант 2. Тестирование класса "Банковский счет". Разработать класс **BankAccount** для управления банковским счетом и комплект тестов.

Требуемые методы класса:

– BankAccount(string owner, decimal initialBalance) — конструктор

– Deposit(decimal amount) — пополнение счета

– Withdraw(decimal amount) — снятие средств

– GetBalance() — получение баланса

– Transfer(BankAccount target, decimal amount) — перевод средств

– AddInterest(decimal rate) — начисление процентов

Бизнес-правила:

- Начальный баланс должен быть >= 0
- Нельзя снять больше, чем есть на счете
- Сумма операции должна быть положительной
- Процентная ставка должна быть от 0 до 100

Требования к тестированию: Разработать не менее 15 тест-кейсов; Проверить все бизнес-правила; Протестировать граничные значения; Проверить выброс исключений при нарушении правил

Вариант 3. Тестирование класса "Валидатор строк". Разработать класс **StringValidator** для проверки строк и комплект тестов.

Требуемые методы класса:

- IsValidEmail(string email) — проверка email
- IsValidPhone(string phone) — проверка телефона (формат +7XXXXXXXXXX)
- IsValidPassword(string password) — проверка пароля (не менее 8 символов, буквы и цифры)
- IsValidINN(string inn) — проверка ИНН (10 или 12 цифр)
- ContainsOnlyDigits(string input) — содержит только цифры
- ContainsOnlyLetters(string input) — содержит только буквы

Требования к тестированию: Разработать не менее 20 тест-кейсов; Проверить корректные и некорректные входные данные; Протестировать граничные случаи (пустые строки, null, специальные символы); Проверить различные комбинации символов

Вариант 4. Тестирование класса "Корзина покупок" Разработать класс **ShoppingCart** для управления корзиной покупок и комплект тестов.

Требуемые методы класса:

- AddItem(string name, decimal price, int quantity) — добавить товар
- RemoveItem(string name) — удалить товар
- UpdateQuantity(string name, int newQuantity) — изменить количество
- GetTotalPrice() — получить общую стоимость
- GetItemCount() — получить количество позиций
- Clear() — очистить корзину
- ApplyDiscount(decimal percent) — применить скидку

Бизнес-правила:

- Цена товара должна быть > 0
- Количество товара должно быть > 0
- Скидка должна быть от 0 до 100%
- Нельзя удалить несуществующий товар

Требования к тестированию: Разработать не менее 15 тест-кейсов; Проверить добавление, удаление, обновление товаров; Протестировать расчет общей стоимости; Проверить применение скидок

Вариант 5. Тестирование класса "Работа с датами". Разработать класс **DateHelper** для работы с датами и комплект тестов.

Требуемые методы класса:

- GetAge(DateTime birthDate) — вычислить возраст
- IsLeapYear(int year) — проверка високосного года
- GetDaysDifference(DateTime date1, DateTime date2) — разница в днях
- GetWeekday(DateTime date) — получить день недели
- AddWorkingDays(DateTime date, int days) — добавить рабочие дни (исключая выходные)
- IsWorkingDay(DateTime date) — проверка рабочего дня

Требования к тестированию: Разработать не менее 15 тест-кейсов; Проверить корректность вычислений для различных дат; Протестировать граничные случаи (високосные годы, переход года); Проверить обработку некорректных дат

Вариант 6. Тестирование класса "Оценки студента". Разработать класс **StudentGrades** для работы с оценками студента и комплект тестов.

Требуемые методы класса:

- AddGrade(string subject, int grade) — добавить оценку (от 2 до 5)
- GetAverageGrade() — средний балл
- GetAverageBySubject(string subject) — средний балл по предмету
- GetGradeCount() — количество оценок
- HasExcellentGrades() — есть ли оценки "5"
- HasFailedGrades() — есть ли оценки "2"
- GetGradeDistribution() — распределение оценок (сколько каких оценок)

Бизнес-правила:

- Оценка должна быть от 2 до 5
- Название предмета не должно быть пустым
- Средний балл округляется до 2 знаков

Требования к тестированию: Разработать не менее 15 тест-кейсов; Проверить добавление оценок и расчет среднего балла; Протестировать граничные значения оценок; Проверить работу с несуществующими предметами

Вариант 7. Тестирование класса "Геометрические фигуры".

Разработать классы геометрических фигур и комплект тестов.

Требуемые классы и методы:

- Класс **Circle**: GetArea(), GetPerimeter(), GetDiameter()
- Класс **Rectangle**: GetArea(), GetPerimeter(), IsSquare()
- Класс **Triangle**: GetArea(), GetPerimeter(), IsValid() (проверка существования треугольника)

Бизнес-правила:

- Все размеры должны быть > 0
- Для треугольника: сумма двух сторон должна быть больше третьей
- Формулы: площадь круга = πr^2, периметр круга = $2\pi r$

Требования к тестированию: Разработать не менее 18 тест-кейсов (по 6 на каждую фигуру); Проверить корректность вычислений; Протестировать граничные значения; Проверить невалидные фигуры

Вариант 8. Тестирование класса "Конвертер величин".

Разработать класс **UnitConverter** для конвертации различных величин и комплект тестов.

Требуемые методы класса:

- CelsiusToFahrenheit(double celsius) — температура
- FahrenheitToCelsius(double fahrenheit) — температура
- KilometersToMiles(double km) — расстояние

- MilesToKilometers(double miles) — расстояние
- KilogramsToLbs(double kg) — вес
- LbsToKilograms(double lbs) — вес

Формулы:

- F = C × 1.8 + 32
- Miles = Km × 0.621371
- Lbs = Kg × 2.20462

Требования к тестированию: Разработать не менее 15 тест-кейсов; Проверить точность конвертации (погрешность не более 0.01); Протестировать граничные значения (0, отрицательные числа); Проверить обратную конвертацию

Вариант 9. Тестирование класса "Парсер текста". Разработать класс **TextParser** для анализа текста и комплект тестов.

Требуемые методы класса:

- CountWords(string text) — подсчет слов
- CountSentences(string text) — подсчет предложений
- CountVowels(string text) — подсчет гласных букв
- CountConsonants(string text) — подсчет согласных букв
- GetMostFrequentWord(string text) — самое частое слово
- IsPalindrome(string text) — проверка палиндрома (игнорируя пробелы и регистр)

Требования к тестированию: Разработать не менее 18 тест-кейсов; Проверить работу с различными текстами; Протестировать граничные случаи (пустые строки, знаки препинания); Проверить работу с кириллицей и латиницей

Вариант 10. Тестирование класса "Библиотечная книга". Разработать класс **LibraryBook** для управления книгой в библиотеке и комплект тестов**.**

Требуемые методы класса:

- LibraryBook(string title, string author, string isbn) — конструктор
- Borrow(string borrower, DateTime date) — выдать книгу
- Return(DateTime returnDate) — вернуть книгу
- IsAvailable() — проверка доступности
- CalculateFine(DateTime returnDate) — расчет штрафа (10 руб/день просрочки)
- GetBorrowHistory() — история выдач

Бизнес-правила:

- ISBN должен быть валидным (13 цифр)
- Нельзя выдать уже выданную книгу

- Срок выдачи — 14 дней
- Штраф начисляется за каждый день просрочки

Требования к тестированию: Разработать не менее 15 тест-кейсов; Проверить логику выдачи и возврата; Протестировать расчет штрафов; Проверить валидацию ISBN

Требования к выполнению задания

1. Разработка тест-плана. Составьте тест-план, включающий**:** Цели тестирования; Объект тестирования (тестируемый класс); Методы, подлежащие тестированию; Типы применяемых тестов; Критерии успешного прохождения

2. Разработка тест-кейсов. Создайте таблицу тест-кейсов со следующими полями: ID тест-кейса (TC-001, TC-002, и т.д.); Название теста; Тестируемый метод; Предусловия; Входные данные; Шаги выполнения; Ожидаемый результат; Фактический результат; Статус (Passed/Failed)

Пример таблицы:

ID	Название	Метод	Входные данные	Ожидаемый результат	Статус
TC-001	Сложение положительных чисел	Add	a=5, b=3	8	Passed
TC-002	Деление на ноль	Divide	a=10, b=0	DivideByZeroException	Passed

3. Реализация класса. Создайте основной проект в Visual Studio 2022:

- Тип проекта: Console App (.NET 6.0 или выше)
- Реализуйте класс согласно варианту задания
- Добавьте необходимую валидацию и обработку исключений

4. Создание тестового проекта. Добавьте в решение тестовый проект:

- Правой кнопкой на решение → Добавить → Создать проект
- Выберите шаблон "Проект модульного тестирования MSTest"
- Добавьте ссылку на основной проект
- Создайте тестовый класс

5. Реализация тестов. Реализуйте все разработанные тест-кейсы в виде тестовых методов: Используйте атрибут [TestMethod];

Применяйте паттерн AAA (Arrange-Act-Assert); Используйте осмысленные имена методов; (например, Add_TwoPositiveNumbers_ReturnsCorrectSum); Добавьте комментарии к сложным тестам

6. Выполнение тестов. Запустите тесты:

- Тест → Запустить все тесты
- Проанализируйте результаты в окне Test Explorer
- Исправьте ошибки при их наличии

7. Анализ покрытия кода. Проверьте покрытие кода тестами: Тест → Анализ покрытия кода для всех тестов; Убедитесь, что покрытие составляет не менее 85%; Добавьте тесты для непокрытых участков кода

8. Документирование результатов. Заполните таблицу тест-кейсов фактическими результатами и статусами.

Содержание отчета. Отчет должен содержать разделы:

1. Тест-план: Описание объекта тестирования; Цели тестирования; Перечень тестируемых методов; Применяемые техники тестирования

2. Тест-кейсы: Полная таблица тест-кейсов с результатами выполнения

3. Исходный код: Код тестируемого класса; Код тестового класса с комментариями

4. Результаты тестирования: Скриншот окна Test Explorer с результатами выполнения тестов; Скриншот отчета о покрытии кода; Анализ результатов

5. Выявленные дефекты: Описание найденных ошибок (если были); Способы их исправления

6. Выводы: Какие навыки приобретены; Какие сложности возникли; Практическая ценность функционального тестирования

Контрольные вопросы

1.Что такое функциональное тестирование? Чем оно отличается от нефункционального тестирования?

2.Перечислите основные виды функционального тестирования. В чем заключается разница между модульным и интеграционным тестированием?

3.Что такое тест-кейс? Опишите обязательные элементы тест-кейса.

4.Объясните паттерн AAA (Arrange-Act-Assert). Почему рекомендуется использовать эту структуру в модульных тестах?

5.Какие методы класса Assert используются в MSTest? Приведите примеры использования.

6.Что такое покрытие кода тестами (Code Coverage)? Какой процент покрытия считается приемлемым?

7.Какие техники тестирования по принципу «черного ящика» вы знаете? Опишите технику граничных значений.

8.Как в MSTest проверить, что метод выбрасывает исключение? Приведите два способа.

9.Зачем нужны атрибуты [TestInitialize] и [TestCleanup]? Приведите пример их использования.

10.Почему важно тестировать граничные условия и исключительные ситуации? К чему может привести отсутствие таких тестов в реальном проекте?

Время выполнения: 4 ч

Практическая работа № 14 «Тестирование интеграции»

Цель работы: Изучить методы и инструменты тестирования интеграции программных модулей, освоить практические навыки разработки интеграционных тестов в среде Visual Studio 2022 с использованием языка C# и фреймворков для модульного тестирования**.**

Теоретическая часть

Понятие тестирования интеграции. Интеграционное тестирование — это этап тестирования программного обеспечения, на котором отдельные программные модули объединяются и тестируются как группа. Цель интеграционного тестирования — выявить дефекты во взаимодействии между интегрированными компонентами или модулями.

Интеграционное тестирование проводится после модульного тестирования (unit testing) и перед системным тестированием. Оно позволяет обнаружить ошибки в интерфейсах между модулями,

проблемы с передачей данных, нарушения протоколов взаимодействия.

Уровни интеграционного тестирования

1. Интеграция компонентов — тестирование взаимодействия между классами и методами внутри одного модуля
2. Интеграция подсистем — тестирование взаимодействия между различными модулями приложения
3. Интеграция систем — тестирование взаимодействия с внешними системами, API, базами данных, сервисами

Стратегии интеграционного тестирования

1. Нисходящая интеграция (Top-Down). Тестирование начинается с модулей верхнего уровня. Модули нижнего уровня заменяются заглушками (stubs), которые имитируют их поведение.

Преимущества: Раннее обнаружение проблем в архитектуре верхнего уровня; Возможность демонстрации базового функционала на ранних этапах

Недостатки: Необходимость создания заглушек; Проблемы нижнего уровня обнаруживаются поздно

2. Восходящая интеграция (Bottom-Up). Тестирование начинается с модулей нижнего уровня. Для вызова модулей нижнего уровня используются драйверы (drivers).

Преимущества: Не требуются заглушки для нижних модулей; Раннее тестирование критичных утилит

Недостатки: Необходимость создания драйверов; Общая структура программы тестируется в последнюю очередь

3. Сэндвич-интеграция (Sandwich/Hybrid). Комбинация нисходящей и восходящей стратегий. Одновременное тестирование с верхнего и нижнего уровней с последующей встречей посередине.

4. Интеграция «большого взрыва» (Big Bang). Все или большинство модулей интегрируются одновременно и тестируются как единое целое.

Преимущества: Простота подхода; Не требуется создание заглушек и драйверов

Недостатки: Сложность локализации ошибок; Позднее обнаружение критических проблем

Инструменты для интеграционного тестирования в C#

1. MSTest. Встроенный в Visual Studio фреймворк для тестирования от Microsoft.

```
[TestClass]
public class IntegrationTests
{
    [TestMethod]
    public void TestDataIntegration()
    {
        // Arrange
        var repository = new UserRepository();
        var service = new UserService(repository);

        // Act
        var result = service.GetUser(1);

        // Assert
        Assert.IsNotNull(result);
    }
}
```

2. NUnit. Популярный фреймворк для тестирования с широкими возможностями.

```
[TestFixture]
public class IntegrationTests
{
    [Test]
    public void TestModuleIntegration()
    {
        // Тестовый код
    }
}
```

3. xUnit. Современный фреймворк для тестирования, используемый в .NET Core.

```
public class IntegrationTests
{
    [Fact]
    public void TestServiceIntegration()
    {
        // Тестовый код
    }
}
```

Техники интеграционного тестирования

1. Тестирование с использованием Mock-объектов

Mock-объекты имитируют поведение реальных объектов для изоляции тестируемого кода.

Библиотека Moq:

```
var mockRepository = new Mock<IUserRepository>();
mockRepository.Setup(r => r.GetUser(1))
              .Returns(new User { Id = 1, Name = "Test" });

var service = new UserService(mockRepository.Object);
var user = service.GetUser(1);
```

2. Тестирование с реальной базой данных

Использование тестовой базы данных для проверки интеграции с СУБД.

```
[TestClass]
public class DatabaseIntegrationTests
{
    private DbContext _context;

    [TestInitialize]
    public void Setup()
    {
        _context = new TestDbContext();
        _context.Database.EnsureCreated();
    }

    [TestCleanup]
    public void Cleanup()
    {
        _context.Database.EnsureDeleted();
        _context.Dispose();
    }
}
```

3. Тестирование API интеграции

Проверка взаимодействия с внешними API и веб-сервисами.

```
[TestMethod]
public async Task TestApiIntegration()
{
    using var client = new HttpClient();
    var response = await client.GetAsync("https://api.example.com/users");

    Assert.AreEqual(HttpStatusCode.OK, response.StatusCode);
```

```
}
```

Паттерн AAA (Arrange-Act-Assert). Стандартная структура тестов:

- **Arrange (Подготовка)** — настройка тестового окружения, создание объектов
- **Act (Действие)** — выполнение тестируемого действия
- **Assert (Проверка)** — проверка результата

```
[TestMethod]
public void TestOrderProcessing()
{
    // Arrange
    var orderService = new OrderService();
    var order = new Order { Id = 1, Amount = 100 };

    // Act
    var result = orderService.ProcessOrder(order);

    // Assert
    Assert.IsTrue(result);
    Assert.AreEqual(OrderStatus.Processed, order.Status);
}
```

Создание тестового проекта в Visual Studio 2022

1.В Solution Explorer щелкните правой кнопкой мыши по решению
2.Выберите **Add → New Project**
3.Выберите шаблон **MSTest Test Project** или **NUnit Test Project**
4.Добавьте ссылку на тестируемый проект
5.Создайте классы тестов

Атрибуты MSTest

[TestClass] — помечает класс как содержащий тесты
[TestMethod] — помечает метод как тестовый
[TestInitialize] — метод выполняется перед каждым тестом
[TestCleanup] — метод выполняется после каждого теста
[ClassInitialize] — выполняется один раз перед всеми тестами класса
[ClassCleanup] — выполняется один раз после всех тестов класса

Методы Assert

Assert.AreEqual(expected, actual) — проверка равенства
Assert.AreNotEqual(notExpected, actual) — проверка неравенства
Assert.IsTrue(condition) — проверка истинности
Assert.IsFalse(condition) — проверка ложности

Assert.IsNull(object) — проверка на null
Assert.IsNotNull(object) — проверка на не-null
Assert.ThrowsException<T>() — проверка выброса исключения

Метрики качества интеграционного тестирования

1.Покрытие кода (Code Coverage) — процент протестированного кода
2.Покрытие интеграционных сценариев — количество проверенных взаимодействий между модулями
3.Количество обнаруженных дефектов — эффективность тестирования
4.Время выполнения тестов — производительность тестового набора

Лучшие практики интеграционного тестирования

1.Изолируйте тесты — каждый тест должен быть независимым
2.Используйте осмысленные имена — название теста должно описывать проверяемый сценарий
3.Поддерживайте тесты актуальными — обновляйте тесты при изменении кода
4.Тестируйте граничные случаи — проверяйте поведение на краях допустимых значений
5.Автоматизируйте запуск — интегрируйте тесты в процесс CI/CD
6.Минимизируйте зависимости — используйте mock-объекты для внешних зависимостей
7.Документируйте тесты — добавляйте комментарии к сложным проверкам

Рабочее задание: Выполните разработку интеграционных тестов для программной системы в соответствии с вариантом задания.

Вариант 1. Система управления библиотекой. Разработать набор интеграционных тестов для системы управления библиотекой, включающей модули:

- BookRepository — работа с базой данных книг
- UserRepository — работа с базой данных читателей
- LoanService — сервис выдачи и возврата книг
- NotificationService — сервис уведомлений о сроках воз-

врата

Требуется протестировать:

Интеграцию между LoanService и BookRepository при выдаче книги; Интеграцию между LoanService и UserRepository при регистрации выдачи; Интеграцию между LoanService и NotificationService при истечении срока; Корректность обновления статуса книги при выдаче и возврате; Обработку ошибочных ситуаций (книга уже выдана, читатель заблокирован)

Вариант 2. Система электронной коммерции. Разработать набор интеграционных тестов для интернет-магазина, включающего модули:

- ProductRepository — работа с каталогом товаров
- OrderRepository — работа с заказами
- PaymentService — сервис обработки платежей
- InventoryService — сервис управления складом
- EmailService — сервис email-уведомлений

Требуется протестировать:

Интеграцию процесса создания заказа с проверкой наличия товара; Интеграцию обработки платежа с обновлением статуса заказа; Интеграцию резервирования товара на складе при оформлении заказа; Интеграцию отправки уведомлений при смене статуса заказа; Откат транзакций при неуспешной оплате

Вариант 3. Система управления задачами (Task Management). Разработать набор интеграционных тестов для системы управления задачами, включающей модули:

- TaskRepository — работа с базой данных задач
- UserRepository — работа с пользователями
- ProjectService — сервис управления проектами
- NotificationService — сервис уведомлений
- ReportService — сервис формирования отчетов

Требуется протестировать:

Интеграцию создания задачи с назначением исполнителя; Интеграцию изменения статуса задачи с отправкой уведомлений; Интеграцию формирования отчета по проекту с агрегацией данных из задач; Проверку прав доступа при операциях с задачами; Каскадное удаление задач при удалении проекта

Вариант 4. Банковская система. Разработать набор инте-

грационных тестов для банковской системы, включающей модули:

- AccountRepository — работа со счетами
- TransactionRepository — работа с транзакциями
- TransferService — сервис переводов между счетами
- ValidationService — сервис валидации операций
- AuditService — сервис аудита операций

Требуется протестировать:
Интеграцию перевода средств с проверкой баланса; Атомарность операции перевода (списание и зачисление); Интеграцию валидации перед выполнением операции; Логирование всех финансовых операций в аудит; Обработку конкурентных операций с одним счетом

Вариант 5. Система бронирования отелей. Разработать набор интеграционных тестов для системы бронирования, включающей модули:

- RoomRepository — работа с номерами
- BookingRepository — работа с бронированиями
- CustomerRepository — работа с клиентами
- PricingService — сервис расчета стоимости
- PaymentService — сервис оплаты

Требуется протестировать:
Интеграцию проверки доступности номера при бронировании; Интеграцию расчета стоимости с учетом сезонности и типа номера; Интеграцию оплаты бронирования с обновлением статуса; Предотвращение двойного бронирования одного номера; Автоматическую отмену неоплаченных бронирований

Вариант 6. Система управления курсами (LMS). Разработать набор интеграционных тестов для системы управления обучением, включающей модули:

- CourseRepository — работа с курсами
- StudentRepository — работа со студентами
- EnrollmentService — сервис записи на курсы
- GradingService — сервис выставления оценок
- CertificateService — сервис генерации сертификатов

Требуется протестировать:
Интеграцию записи студента на курс с проверкой предваритель-

ных требований; Интеграцию выставления оценок с обновлением прогресса студента; Интеграцию автоматической генерации сертификата при завершении курса; Проверку лимита мест на курсе; Расчет итоговой оценки на основе всех заданий курса

Вариант 7. Система складского учета. Разработать набор интеграционных тестов для системы складского учета, включающей модули:

- ProductRepository — работа с товарами
- WarehouseRepository — работа со складами
- InventoryService — сервис управления запасами
- OrderService — сервис обработки заказов
- SupplierService — сервис работы с поставщиками

Требуется протестировать:
Интеграцию приемки товара от поставщика с обновлением остатков; Интеграцию резервирования товара под заказ; Интеграцию перемещения товара между складами; Автоматическое формирование заказа поставщику при достижении минимального остатка; Инвентаризацию с выявлением расхождений

Вариант 8. Социальная сеть. Разработать набор интеграционных тестов для социальной сети, включающей модули:

- UserRepository — работа с пользователями
- PostRepository — работа с публикациями
- FriendshipService — сервис управления дружбой
- FeedService — сервис формирования ленты новостей
- NotificationService — сервис уведомлений

Требуется протестировать:
Интеграцию создания публикации с обновлением ленты друзей; Интеграцию добавления в друзья с отправкой уведомления; Интеграцию формирования персонализированной ленты новостей; Фильтрацию контента на основе настроек приватности; Каскадное удаление публикаций при удалении пользователя

Вариант 9. Система медицинских записей. Разработать набор интеграционных тестов для системы медицинских записей, включающей модули:

- PatientRepository — работа с пациентами
- DoctorRepository — работа с врачами
- AppointmentRepository — работа с записями на прием

- ScheduleService — сервис управления расписанием
- MedicalRecordService — сервис медицинских карт

Требуется протестировать:
Интеграцию создания записи на прием с проверкой расписания врача; Интеграцию добавления записи в медицинскую карту пациента; Предотвращение конфликтов в расписании врача; Интеграцию отмены приема с освобождением времени в расписании; Доступ к медицинским картам только авторизованным врачам

Вариант 10. Система управления парковкой. Разработать набор интеграционных тестов для системы управления парковкой, включающей модули:

- ParkingSpaceRepository — работа с парковочными местами
- VehicleRepository — работа с транспортными средствами
- EntryExitService - сервис въезда/выезда
- PaymentService — сервис оплаты парковки
- ReservationService — сервис резервирования мест

Требуется протестировать:
Интеграцию регистрации въезда с занятием парковочного места; Интеграцию расчета стоимости парковки при выезде; Интеграцию резервирования места с блокировкой для других пользователей; Автоматическое освобождение места при выезде; Обработку оплаты и открытие шлагбаума

Требования к выполнению задания

1. Создайте основной проект приложения: Разработайте классы для основных модулей системы согласно варианту: Репозитории для работы с данными (можно использовать in-memory коллекции); Сервисы бизнес-логики; Модели данных (классы сущностей); Интерфейсы для абстракции зависимостей
2. Создайте тестовый проект: Добавьте в решение проект MSTest Test Project; Добавьте ссылку на основной проект; При необходимости установите NuGet-пакет Moq для создания mock-объектов
3. Разработайте интеграционные тесты. Создайте не менее **8 интеграционных тестов**, включающих:

- Минимум 3 теста на успешные сценарии интеграции
- Минимум 2 теста на обработку ошибочных ситуаций

- Минимум 2 теста с использованием mock-объектов
- Минимум 1 тест на проверку транзакционности/атомарности операций

4. Структура каждого теста должна соответствовать паттерну AAA:

```
[TestMethod]
public void ОписательноеИмяТеста()
{
   // Arrange - подготовка данных и объектов

   // Act - выполнение тестируемого действи

   // Assert - проверка результата
}
```

5. Используйте осмысленные имена тестов

Примеры:

- WhenBookIsAvailable_LoanService_ShouldSuccessfullyLoanBook
- WhenInsufficientFunds_TransferService_ShouldThrowException
- WhenOrderIsPaid_InventoryService_ShouldReserveProducts

6. Запустите все тесты

- Используйте Test Explorer в Visual Studio
- Убедитесь, что все тесты проходят успешно (зеленые)
- Сделайте скриншот результатов выполнения тестов

7. Проверьте покрытие кода тестами (опционально)

- Используйте встроенные инструменты Visual Studio для анализа покрытия кода
- Стремитесь к покрытию не менее 70% кода сервисов

Содержание отчета. Отчет должен содержать:

1.Описание разработанной системы: Диаграмма классов основных компонентов; Описание модулей и их взаимодействия; Перечень реализованных интерфейсов

2.Листинг кода основных классов: Модели данных; Репозитории; Сервисы

3.Листинг кода тестов: Все разработанные тестовые методы с комментариями; Объяснение логики каждого теста

4.Результаты выполнения тестов: Скриншот Test Explorer с результатами; Описание каждого теста и его результата; При наличии - отчет о покрытии кода

5.Анализ результатов: Какие интеграционные сценарии были протестированы; Какие дефекты были обнаружены в процессе разработки тестов; Выводы о качестве интеграции модулей
6.Выводы по работе

Контрольные вопросы

1.Что такое интеграционное тестирование и чем оно отличается от модульного тестирования? На каком этапе разработки оно выполняется?

2.Опишите основные стратегии интеграционного тестирования (нисходящая, восходящая, сэндвич, «большой взрыв»). В каких случаях применяется каждая из них?

3.Что такое mock-объекты и для чего они используются в интеграционном тестировании? Приведите пример использования библиотеки Moq.

4.Опишите паттерн AAA (Arrange-Act-Assert). Почему рекомендуется следовать этой структуре при написании тестов?

5.Какие атрибуты используются в MSTest для создания тестовых классов и методов? В чем разница между [TestInitialize] и [ClassInitialize]?

6.Перечислите основные методы класса Assert в MSTest. Для чего используется метод Assert.ThrowsException<T>()?

7.Что такое покрытие кода (code coverage) и как его измерить в Visual Studio? Какой процент покрытия считается достаточным для критичных компонентов?

8.Какие преимущества дает автоматизация интеграционных тестов? Как интеграционные тесты встраиваются в процесс непрерывной интеграции (CI/CD)?

9.В чем заключаются лучшие практики именования тестовых методов? Почему важно давать тестам описательные имена?

10.Объясните, что такое атомарность операций и как проверить атомарность в интеграционном тесте. Приведите пример из вашего варианта задания.

Время выполнения: 4 ч

Практическая работа № 15 «Документирование результатов тестирования»

Цель работы: Изучить методы и средства документирования результатов тестирования программных модулей, освоить структуру и правила оформления тестовой документации, получить практические навыки составления отчетов о тестировании и протоколов дефектов в среде разработки Visual Studio 2022 с использованием языка C#.

Теоретическая часть

Понятие тестирования программного обеспечения. Тестирование — это процесс исследования, испытания программного продукта, имеющий две цели**:**

1.Продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям

2.Выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации

Тестирование является неотъемлемой частью процесса интеграции программных модулей и позволяет убедиться в корректности взаимодействия компонентов системы.

Виды тестирования

Модульное (Unit) тестирование — тестирование отдельных программных модулей в изоляции от других компонентов системы.

Интеграционное тестирование — тестирование взаимодействия между интегрированными модулями или компонентами системы.

Системное тестирование — тестирование системы в целом на соответствие требованиям.

Регрессионное тестирование — повторное тестирование после внесения изменений для проверки отсутствия новых дефектов.

Документирование результатов тестирования. Документирование результатов тестирования — это процесс фиксации информации о проведенных тестах, выявленных дефектах и общем качестве программного продукта. Качественная документация позволяет:

- отслеживать прогресс тестирования
- фиксировать найденные ошибки и баги
- анализировать качество программного продукта
- предоставлять доказательства выполненной работы
- обеспечивать коммуникацию между членами команды
- принимать решения о готовности продукта к выпуску

Основные документы тестирования

1. План тестирования (Test Plan). Определяет стратегию, подход, ресурсы и график тестирования. Включает: цели и задачи тестирования; объекты тестирования; виды тестирования; критерии начала и завершения тестирования; ресурсы и ответственных; риски

2. Тест-кейс (Test Case). Детальное описание одного тестового сценария. Содержит: идентификатор тест-кейса; название; предусловия; шаги выполнения; тестовые данные; ожидаемый результат; фактический результат; статус (пройден/не пройден)

3. Отчет об ошибке (Bug Report / Defect Report). Документ, описывающий выявленный дефект. Включает: идентификатор дефекта; краткое описание; шаги воспроизведения; ожидаемый результат; фактический результат; приоритет и серьезность; статус; скриншоты/логи

4. Отчет о результатах тестирования (Test Summary Report). Итоговый документ, обобщающий результаты тестирования. Содержит: статистику выполненных тестов; количество найденных дефектов; метрики качества; общую оценку готовности продукта; рекомендации

Классификация дефектов

По серьезности (Severity):

– Критический (Critical) — полная неработоспособность системы, потеря данных

–Высокий (High) — существенное нарушение функциональности

– Средний (Medium) — частичное нарушение функциональности

– Низкий (Low) — незначительные недостатки, косметические дефекты

По приоритету (Priority):
– Срочный (Urgent) — требует немедленного исправления
– Высокий (High**)** — должен быть исправлен в ближайшее время
– Средний (Medium) — может быть исправлен в текущей версии
– Низкий (Low) — может быть отложен
По статусу:

- Новый (New**)** — дефект только что зарегистрирован
- Назначен (Assigned) — дефект назначен разработчику
- В работе (In Progress) — дефект исправляется
- Исправлен (Fixed) — дефект исправлен
- Проверен (Verified) — исправление проверено
- Закрыт (Closed) — дефект закрыт
- Отклонен (Rejected) — дефект признан некорректным

Стандарты документирования тестирования
ISO/IEC/IEEE 29119 — международный стандарт тестирования программного обеспечения, определяющий процессы, документацию и техники тестирования.
ГОСТ Р ИСО/МЭК 9126 — стандарт оценки качества программного обеспечения.
Инструменты документирования в Visual Studio 2022
MSTest / NUnit / xUnit. Фреймворки для модульного тестирования в .NET, поддерживающие создание тестов и генерацию отчетов.
Test Explorer. Встроенный инструмент Visual Studio для запуска тестов и просмотра результатов.
Azure DevOps / TFS. Системы управления жизненным циклом приложений, включающие средства для управления тестами и дефектами.
XML-комментарии в коде. Специальные комментарии для документирования кода:

```
/// <summary>
/// Выполняет сложение двух чисел
/// </summary>
/// <param name="a">Первое слагаемое</param>
/// <param name="b">Второе слагаемое</param>
/// <returns>Сумма двух чисел</returns>
public int Add(int a, int b)
{
```

```
    return a + b;
}
```

Структура тест-кейса

ID: TC-001
Название: Проверка авторизации с корректными данными
Приоритет: Высокий
Предусловия:
- Пользователь зарегистрирован в системе
- Приложение запущено
Шаги:
1. Открыть форму авторизации
2. Ввести логин: "user@example.com"
3. Ввести пароль: "Password123"
4. Нажать кнопку "Войти"
Ожидаемый результат: Пользователь успешно авторизован и перенаправлен на главную страницу

Фактический результат: (заполняется при выполнении)

Статус: (заполняется при выполнении)

Структура отчета об ошибке

ID дефекта: BUG-001
Дата: 15.12.2024
Обнаружил: Иванов И.И.
Модуль: Модуль авторизации
Серьезность: Высокая
Приоритет: Срочный
Статус: Новый
Краткое описание: Невозможно войти в систему при использовании пароля с кириллическими символами
Шаги воспроизведения:
1. Запустить приложение
2. Открыть форму авторизации
3. Ввести логин: "user@example.com"
4. Ввести пароль: "Пароль123"
5. Нажать кнопку "Войти"

Ожидаемый результат: Пользователь должен успешно войти в систему
Фактический результат: Появляется сообщение об ошибке "Невер-

ный логин или пароль"

Дополнительная информация:
- ОС: Windows 11
- Версия приложения: 1.0.0
- Скриншот: прилагается

Метрики тестирования

Test Coverage (Покрытие кода тестами) — процент кода, который выполняется при запуске тестов.
Defect Density — количество дефектов на единицу размера кода (обычно на 1000 строк кода).
Test Pass Rate — процент успешно пройденных тестов от общего количества.
Defect Detection Rate — количество дефектов, обнаруженных на определенном этапе тестирования.

Лучшие практики документирования

– Однозначность — описания должны быть четкими и недвусмысленными
– Полнота — вся необходимая информация должна быть зафиксирована
– Актуальность — документация должна своевременно обновляться
– Структурированность — использование шаблонов и единого формата
– Прослеживаемость — связь между требованиями, тестами и дефектами
– Доступность — документация должна быть доступна всем участникам проекта

Рабочее задание: Выполните тестирование программного модуля в соответствии с вариантом задания и составьте комплект документации по результатам тестирования.

Вариант 1. Модуль работы с математическими операциями. Протестировать модуль калькулятора, выполняющий базовые арифметические операции (сложение, вычитание, умножение, деление). Модуль должен корректно обрабатывать различные типы данных и граничные случаи (деление на ноль,

переполнение).

Требуется протестировать:

Операцию сложения целых и вещественных чисел; Операцию вычитания с отрицательными числами; Операцию умножения, включая умножение на ноль; Операцию деления с обработкой деления на ноль; Работу с граничными значениями (максимальные/минимальные значения типов данных)

Вариант 2. Модуль валидации пользовательского ввода. Протестировать модуль валидации данных, проверяющий корректность ввода email, телефона, пароля. Модуль должен возвращать информацию о корректности данных и список ошибок валидации.

Требуется протестировать:

Валидацию email-адресов (корректные и некорректные форматы); Валидацию номеров телефонов (различные форматы); Валидацию паролей (требования к длине, сложности); Обработку пустых и null значений; Обработку специальных символов

Вариант 3. Модуль работы со строками. Протестировать модуль обработки строк, выполняющий операции поиска, замены, разбиения строк, изменения регистра. Модуль должен корректно работать с различными кодировками и специальными символами.

Требуется протестировать:

Поиск подстроки в строке (с учетом и без учета регистра); Замену подстроки; Разбиение строки по разделителю; Изменение регистра (верхний, нижний, каждое слово с заглавной); Работу с пустыми строками и null

Вариант 4. Модуль работы с коллекциями данных. Протестировать модуль для работы с коллекциями (List, Dictionary), выполняющий операции добавления, удаления, поиска, сортировки элементов. Модуль должен обрабатывать различные типы данных.

Требуется протестировать:

Добавление элементов в коллекцию; Удаление элементов из коллекции; Поиск элементов в коллекции; Сортировку коллекции (по возрастанию/убыванию); Работу с пустыми коллекциями

Вариант 5. Модуль работы с датами и временем. Проте-

стировать модуль для работы с датами, выполняющий операции сравнения дат, вычисления разницы между датами, форматирования дат. Модуль должен учитывать часовые пояса и високосные годы.

Требуется протестировать:

Сравнение дат (раньше, позже, равны); Вычисление разницы между датами (в днях, часах); Форматирование дат в различные форматы; Обработку високосных годов; Работу с граничными датами (минимальная/максимальная дата)

Вариант 6. Модуль файловых операций. Протестировать модуль для работы с файлами, выполняющий операции чтения, записи, копирования, удаления файлов. Модуль должен корректно обрабатывать различные форматы файлов и ошибки доступа.

Требуется протестировать:

Чтение содержимого файла; Запись данных в файл; Копирование файла; Удаление файла; Обработку ошибок (файл не найден, нет прав доступа)

Вариант 7. Модуль конвертации данных. Протестировать модуль конвертации данных, выполняющий преобразование между различными форматами (JSON, XML, CSV). Модуль должен корректно обрабатывать различные структуры данных.

Требуется протестировать:

Преобразование объектов в JSON; Преобразование JSON в объекты; Преобразование данных в XML; Преобразование XML в объекты; Обработку некорректных данных

Вариант 8. Модуль шифрования данных. Протестировать модуль шифрования, выполняющий операции шифрования и дешифрования текстовых данных различными алгоритмами. Модуль должен обеспечивать безопасное хранение ключей.

Требуется протестировать:

Шифрование текстовых данных; Дешифрование зашифрованных данных; Корректность работы с различными длинами ключей; Обработку некорректных данных при дешифровании; Работу с пустыми строками

Вариант 9. Модуль работы с базой данных. Протестировать модуль для работы с базой данных, выполняющий операции

CRUD (Create, Read, Update, Delete). Модуль должен корректно обрабатывать подключение к БД и SQL-запросы.

Требуется протестировать:

Создание записей в базе данных; Чтение данных из базы данных; Обновление существующих записей; Удаление записей; Обработку ошибок подключения и выполнения запросов

Вариант 10. Модуль логирования событий. Протестировать модуль логирования, записывающий события приложения в файл с различными уровнями важности (Info, Warning, Error). Модуль должен поддерживать ротацию логов.

Требуется протестировать:

Запись сообщений различных уровней; Форматирование сообщений логов; Запись в файл; Обработку ошибок записи; Корректность работы при многопоточном доступе

Вариант 11. Модуль аутентификации пользователей. Протестировать модуль аутентификации, выполняющий проверку учетных данных пользователя, хеширование паролей, генерацию токенов доступа.

Требуется протестировать:

Проверку корректных учетных данных; Проверку некорректных учетных данных; Хеширование паролей; Генерацию токенов доступа; Проверку срока действия токенов

Вариант 12. Модуль отправки уведомлений. Протестировать модуль отправки уведомлений, поддерживающий отправку сообщений по email и SMS. Модуль должен формировать шаблоны сообщений и обрабатывать очередь отправки.

Требуется протестировать:

Формирование сообщений по шаблону; Валидацию адресов получателей; Добавление сообщений в очередь; Обработку ошибок отправки; Повторную отправку при сбоях

В*ариант 13. Модуль генерации отчетов.* Протестировать модуль генерации отчетов, создающий документы в форматах PDF, Excel, Word на основе данных из системы.

Требуется протестировать:

Генерацию отчета в формате PDF; Генерацию отчета в формате Excel; Применение шаблонов оформления; Корректность данных в отчете; Обработку больших объемов данных

Вариант 14. Модуль кэширования данных. Протестировать модуль кэширования, обеспечивающий хранение часто используемых данных в памяти с механизмом инвалидации устаревших записей**.**

Требуется протестировать:

Сохранение данных в кэш; Извлечение данных из кэша; Удаление данных из кэша; Автоматическое удаление устаревших данных (TTL); Работу при переполнении кэша

Вариант 15. Модуль поиска и фильтрации данных. Протестировать модуль поиска, выполняющий полнотекстовый поиск по данным с поддержкой фильтров, сортировки и пагинации результатов.

Требуется протестировать:

Поиск по ключевым словам; Применение фильтров; Сортировку результатов; Пагинацию результатов (разбивка на страницы); Поиск с пустым запросом

Требования к выполнению задания

1. Разработка программного модуля. Создайте в Visual Studio 2022 консольное приложение на языке C#, реализующее функциональность в соответствии с вашим вариантом задания. Код должен быть структурирован, содержать XML-комментарии. Пример структуры проекта:

```
ИмяПроекта/
├── Program.cs (главный класс)
├── Модуль.cs (класс с тестируемой функциональностью)
└── Tests/ (папка с тестами, если используется отдельный проект)
```

2. Создание тест-кейсов. Разработайте **не менее 10 тест-кейсов**, покрывающих различные сценарии использования модуля**:** позитивные тест-кейсы (корректные данные); негативные тест-кейсы (некорректные данные); граничные значения; исключительные ситуации

Каждый тест-кейс должен содержать: уникальный идентификатор; название; приоритет; предусловия; шаги выполнения; тестовые данные; ожидаемый результат; поле для фактического результата; поле для статуса

3. Выполнение тестирования. Выполните разработанные тест-кейсы, зафиксируйте фактические результаты и статусы (Passed/Failed). Для модульно- го тестирования можно ис-

пользовать:

- Ручное тестирование с фиксацией результатов
- Автоматизированное тестирование с использованием MSTest или NUnit (по желанию)

4. Документирование дефектов. Для каждого выявленного дефекта (несоответствия ожидаемого и фактического результата) составьте отчет об ошибке, содержащий:
– уникальный идентификатор; – дату обнаружения; – модуль; – краткое описание; - шаги воспроизведения; – ожидаемый результат; – фактический результат; - серьезность (Critical, High, Medium, Low); приоритет (Urgent, High, Medium, Low); – статус; – скриншот или лог (при необходимости)
Документируйте **не менее 3 дефектов** (реальных или смоделированных).

5. Составление отчета о результатах тестирования. Подготовьте итоговый отчет о тестировании, который должен включать:
Статистика выполнения тестов: какие виды тестирования применялись; общее количество тест-кейсов; количество пройденных тестов; количество проваленных тестов; процент успешного прохождения (Test Pass Rate)
Статистика дефектов: общее количество найденных дефектов; распределение по серьезности; распределение по приоритету; распределение по статусам
Метрики качества: покрытие функциональности тестами; плотность дефектов (если применимо)
Анализ результатов: основные проблемы; рекомендации по доработке; риски
Заключение: общая оценка качества модуля; готовность к интеграции/релизу

6. Оформление отчета по практической работе
Подготовьте отчет, содержащий:
Раздел 1. Описание разработанного модуля: назначение модуля; реализованная функциональность; листинг кода с комментариями
Раздел 2. Тест-кейсы: таблица со всеми разработанными тест-кейсами; детальное описание каждого тест-кейса

Раздел 3. Протоколы дефектов: таблица выявленных дефектов; детальное описание каждого дефекта
Раздел 4. Отчет о результатах тестирования: итоговый отчет с метриками и анализом
Раздел 5. Скриншоты: скриншоты выполнения тестов; скриншоты выявленных ошибок; скриншот структуры проекта в Visual Studio
Выводы
Пример выполнения задания
Пример кода модуля (для варианта 1)

```
using System;

namespace CalculatorModule
{
    /// <summary>
    /// Класс для выполнения математических операций
    /// </summary>
    public class Calculator
    {
        /// <summary>
        /// Выполняет сложение двух чисел
        /// </summary>
        /// <param name="a">Первое слагаемое</param>
        /// <param name="b">Второе слагаемое</param>
        /// <returns>Сумма двух чисел</returns>
        public double Add(double a, double b)
        {
            return a + b;
        }

        /// <summary>
        /// Выполняет вычитание двух чисел
        /// </summary>
        /// <param name="a">Уменьшаемое</param>
        /// <param name="b">Вычитаемое</param>
        /// <returns>Разность двух чисел</returns>
        public double Subtract(double a, double b)
        {
            return a - b;
        }

        /// <summary>
```

```
        /// Выполняет умножение двух чисел
        /// </summary>
        /// <param name="a">Первый множитель</param>
        /// <param name="b">Второй множитель</param>
        /// <returns>Произведение двух чисел</returns>
        public double Multiply(double a, double b)
        {
            return a * b;
        }

        /// <summary>
        /// Выполняет деление двух чисел
        /// </summary>
        /// <param name="a">Делимое</param>
        /// <param name="b">Делитель</param>
        /// <returns>Частное двух чисел</returns>
        /// <exception cref="DivideByZeroException">
        /// Выбрасывается при попытке деления на ноль
        /// </exception>
        public double Divide(double a, double b)
        {
            if (b == 0)
            {
                throw new DivideByZeroException(
                    "Деление на ноль невозможно");
            }
            return a / b;
        }
    }
}
```

Пример тест-кейса

ID: TC-001
Название: Проверка сложения двух положительных целых чисел
Приоритет: Высокий
Предусловия: Объект класса Calculator создан

Шаги выполнения:
1. Вызвать метод Add с параметрами a = 5, b = 3
2. Получить результат

Тестовые данные:
a = 5
b = 3

Ожидаемый результат:
Метод возвращает значение 8.0

Фактический результат:
Метод вернул значение 8.0

Статус: Passed

Пример отчета об ошибке

ID дефекта: BUG-001
Дата: 15.12.2024
Обнаружил: Петров П.П.
Модуль: Calculator.Divide
Серьезность: Критическая
Приоритет: Срочный
Статус: Новый

Краткое описание:
При делении максимального значения double на 0.0001
возвращается Infinity вместо корректного числа

Шаги воспроизведения:
1. Создать объект класса Calculator
2. Вызвать метод Divide(Double.MaxValue, 0.0001)
3. Проверить результат

Ожидаемый результат:
Метод должен вернуть корректное числовое значение
или выбросить исключение переполнения

Фактический результат:
Метод возвращает Double.PositiveInfinity

Дополнительная информация:
- Visual Studio 2022
- .NET 6.0
- Windows 11

Пример фрагмента итогового отчета

ОТЧЕТ О РЕЗУЛЬТАТАХ ТЕСТИРОВАНИЯ
Модуль: Calculator
Версия: 1.0.0
Дата тестирования: 15.12.2024
Ответственный: Петров П.П.

СТАТИСТИКА ТЕСТОВ:

Всего тест-кейсов: 10
Пройдено: 7 (70%)
Провалено: 3 (30%)
СТАТИСТИКА ДЕФЕКТОВ:
Всего дефектов: 3
Критические: 1
Высокие: 1
Средние: 1
Низкие: 0

ЗАКЛЮЧЕНИЕ:
Модуль требует доработки. Критические дефекты
должны быть исправлены перед интеграцией.

Контрольные вопросы

1.Что такое тестирование программного обеспечения и каковы его основные цели?

2.Какие виды тестирования вы знаете? В чем разница между модульным и интеграционным тестированием?

3.Какие основные документы создаются в процессе тестирования программного обеспечения? Опишите назначение каждого из них.

4.Что должен содержать тест-кейс? Приведите структуру тест-кейса и объясните назначение каждого элемента.

5.Что такое отчет об ошибке (Bug Report)? Какая информация должна быть в нем зафиксирована?

6.В чем различие между серьезностью (Severity) и приоритетом (Priority) дефекта? Приведите примеры.

7.Какие метрики используются для оценки качества тестирования? Как рассчитывается Test Pass Rate?

8.Что такое покрытие кода тестами (Test Coverage) и почему это важный показатель?

9.Какой международный стандарт регламентирует процессы тестирования программного обеспечения?

10.Какие инструменты для автоматизированного тестирования доступны в Visual Studio 2022? В чем преимущества автоматизированного тестирования перед ручным?

11.Что такое регрессионное тестирование и в каких случаях оно применяется?

12.Зачем необходимо документировать результаты тестирования? Какие проблемы могут возникнуть при отсутствии документации?

13.Какие статусы может иметь дефект в процессе его жизненного цикла? Опишите каждый статус.

14.Что должен содержать итоговый отчет о результатах тестирования (Test Summary Report)?

15.Какие лучшие практики документирования результатов тестирования вы можете назвать?

Время выполнения: 4 ч

ПЕРЕЧЕНЬ ИСПОЛЬЗОВАННЫХ ИНФОРМАЦИОННЫХ РЕСУРСОВ

1. Мейер Б. Объектно-ориентированное программирование и программная инженерия Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Эр Медиа, 2019 – 285с. http://www.iprbookshop.ru/79706.html
2. Федорова Г.Н Разработка модулей программного обеспечения для компьютерных систем: учебник для студ. СПО М.: Академия 2018, 384 с.
3. Фуфаев Д.Э. Разработка и эксплуатация автоматизированных информационных систем Разработка и эксплуатация автоматизированных информационных систем учеб. пособие для студ. СПО М.: Академия 2018. 304 с.
4.Никлаус Вирт. Алгоритмы и структуры данных/ Саратов: Профобразование Электрон.текстовые данные (электронный ресурс) http://www.iprbookshop.ru/63821.html
5. Семакин И.Г Основы алгоритмизации и программирования. Практикум учеб. пособие для сред. проф. образования М.: Академия 2018. 144 с.