АКТ(ф) СПбГУТ

**ОТЧЕТ**

по лабораторным работам

Поддержка и тестирование программных модулей

Студент ИСПП-21 19.09.2024 С.А.Миклякова

Преподаватель 19.09.2024 Р.В.Садовский

Архангельск 2024

# Лабораторная работа №1

## Анализ и тестирование требований к ПО

1. **Цель работы**
   1. Изучить процесс анализа и тестирования требований к программному обеспечению,
   2. Приобрести навыки выявления, формулирования и проверки требований к ПО на примере разработки простой системы управления задачами.
2. **Ход работы**
   1. Функциональные и нефункциональные требования соответствуют свойствам качественных требований к ПО, так как сформулированы четко, а также достижимы и прослеживаемы.

Бизнес требования не соответствуют свойствам качественных требований к ПО, так как не являются проверяемыми.

Во-первых, первое бизнес-требование никак не относится к работе программиста.

Во-вторых, его невозможно измерить и точно сказать, к чему относятся изменения в эффективности, к внедрению ПО или влиянию других факторов.

Пользовательские требования не соответствуют свойствам качественных требований к ПО, так как второе требование сформулировано неполно. Формулировка «интуитивно понятный интерфейс» не имеет четкого значения.

Системные требования не соответствуют свойствам качественных требований к ПО, так как являются неполными, и их очень мало для разработки качественного ПО

* 1. Тест кейсы для функциональных требований:

**Тест кейс: Создание новой задачи**

1. Пользователь открывает приложение.
2. Пользователь выбирает опцию "Создать новую задачу".
3. Пользователь вводит название задачи, описание и сроки выполнения.
4. Пользователь нажимает кнопку "Сохранить".

Ожидаемый результат: Новая задача успешно сохраняется в системе.

**Тест кейс: Установка статуса задачи**

1. Пользователь открывает приложение и выбирает задачу.
2. Пользователь находит опцию для установки статуса задачи.
3. Пользователь выбирает статус задачи.
4. Пользователь сохраняет изменения.

Ожидаемый результат: Статус задачи успешно изменен на выбранный.

**Тест кейс: Сортировка задач по дате создания**

1. Пользователь открывает приложение
2. Пользователь выбирает опцию "Сортировка задач"
3. Пользователь выбирает опцию "По дате создания"

Ожидаемый результат: Задачи отображаются в порядке возрастания даты создания.

**Тест кейс: Сортировка задач по сроку выполнения**

1. Пользователь открывает приложение
2. Пользователь выбирает опцию "Сортировка задач"
3. Пользователь выбирает опцию "По сроку выполнения"

Ожидаемый результат: Задачи отображаются в порядке возрастания срока выполнения.

2.3 Тест-кейсы для нефункциональных требований

**Тест кейс для проверки времени отклика системы при создании новой задачи:**

1. Зайти в систему под учетной записью пользователя.
2. Нажать на кнопку "Создать новую задачу".
3. Заполнить все необходимые поля формы.
4. Нажать на кнопку "Создать задачу".

Ожидаемый результат: Время отклика системы при создании новой задачи не превышает 2 секунд.

**Тест кейс для проверки поддержки системы одновременной работы не менее 1000 пользователей:**

1. Запустить 1001 сессию входа пользователя в систему.
2. Попробовать одновременно создать новые задачи с разных аккаунтов.

Ожидаемый результат: Система должна успешно обрабатывать запросы от всех 1000 пользователей и не должна падать или тормозить при этом.

**Тест кейс для проверки адаптации интерфейса системы для работы на мобильных устройствах:**

1. Зайти на сайт системы с мобильного устройства.
2. Проверить отображение всех элементов интерфейса (меню, кнопки, поля ввода и т.д.).
3. Попробовать создать задачу с мобильного устройства.

Ожидаемый результат: Интерфейс системы должен быть адаптирован для удобной работы на мобильных устройствах, все элементы должны отображаться корректно и быть доступными для использования.

1. **Ответы на контрольные вопросы**
   1. Требования делятся на:
      * бизнес-требования,
      * пользовательские требования,
      * системные требования,
      * функциональные требования,
      * нефункциональные требования.
   2. Для того чтобы определить, является ли требование тестируемым, оно должно быть измеримым, однозначным, достижимым и проверяемым. Тестируемые требования должны быть валидными, полными и согласованными с другими требованиями проекта
   3. Ключевые характеристики, делающие требование качественным, включают в себя:
   * однозначность: требование должно быть понятным и не допускать двусмысленностей,
   * проверяемость: требование должно быть измеримым для проверки его выполнения,
   * полнота: требование должно содержать достаточно информации для его реализации и проверки,
   * достижимость: требование должно быть реализуемо в рамках проекта с учетом доступных ресурсов,
   * консистентность: требования не должны противоречить друг другу.
2. **Вывод**
   1. В ходе проделанной лабораторной работы был изучен процесс анализа и тестирования требований к программному обеспечению,
   2. В ходе проделанной лабораторной работе были приобретены навыки выявления, формулирования и проверки требований к ПО на примере разработки простой системы управления задачами.

# Лабораторная работа №2

## Usability тестирование тестовой версии ПО по набору раннее

## определенных показателей

1. **Цель работы**
   1. Проведение тестирования удобства использования (usability) тестовой версии программного обеспечения (ПО) с целью оценки его качества.
2. **Ход работы**
   1. Определили набор показателей, которые будут использоваться для оценки usability ПО.
   2. Сформулировали список типовых задач (сценариев), которые необходимо выполнить для тестирования.
   3. Подготовили протокол для записи времени выполнения задач, ошибок и субъективных оценок.
   4. Запустили тестируемое ПО и последовательно выполнить все задачи согласно подготовленным сценариям.
   5. Во время выполнения каждой задачи фиксировали время, затраченное на ее выполнение, а также количество и тип ошибок.
   6. По завершении каждой задачи оценили, насколько она была удобной для выполнения, и записали свои субъективные впечатления в протокол.
   7. Проанализировали собранные данные, сравнив их с ожидаемыми показателями. Оценили, насколько ПО соответствует критериям удобства использования.
   8. Составили отчет по проделанной работе.
3. **Ответы на контрольные вопросы**
   1. При оценке удобства использование ПО учитывается:

* эффективность,
* удовлетворенность,
* эффективность использования,
* надежность,
* производительность,
* сопровождаемость,
* безопасность.
  1. Когнитивная нагрузка (Cognitive Load) — это мера умственных усилий, которые пользователь должен приложить для выполнения задачи.

Влияние когнитивной нагрузки на usability ПО:

1. удовлетворенность пользователя: высокая когнитивная нагрузка может привести к разочарованию и усталости пользователя, что снижает удовлетворенность использованием ПО,
2. эффективность использования: если пользователь тратит слишком много умственных усилий на понимание интерфейса и выполнение задач, это снижает эффективность его работы с ПО,
3. учитываемость: высокая когнитивная нагрузка затрудняет обучение новым пользователям, так как им сложнее запомнить и понять, как работает программа,
4. производительность: пользователи, сталкивающиеся с высокой когнитивной нагрузкой, могут совершать больше ошибок и тратить больше времени на выполнение задач, что снижает общую производительность.
   1. Преимущества и недостатки у различных методов сбора данных (наблюдение, опросы, интервью) в usability тестировании:

**Наблюдение (Observation)**

Преимущества:

* **естественное поведение**: наблюдение позволяет увидеть, как пользователи взаимодействуют с интерфейсом в естественных условиях, без искажений, вызванных опросами или интервью,
* **детальная информация**: можно заметить мелкие детали и нюансы поведения пользователей, которые могут быть пропущены в других методах,
* **неинвазивность**: пользователи не знают, что их наблюдают, что позволяет получить более естественные и непринужденные реакции.

Недостатки:

* **субъективность**: результаты наблюдения могут быть субъективными и зависеть от навыков наблюдателя,
* **невозможность получить глубокую информацию**: наблюдение не позволяет получить глубокие объяснения мотивов и мыслей пользователей,
* **времязатратность**: требует много времени для анализа и интерпретации данных.

**Опросы (Surveys)**

Преимущества:

* **быстрое получение данных**: опросы позволяют быстро собрать данные от большого количества пользователей,
* **количественная информация**: можно получить количественные данные, которые легко анализировать и сравнивать,
* **удобство**: опросы можно проводить удаленно, что удобно для пользователей и исследователей.

Недостатки:

* **неглубокая информация**: опросы не позволяют получить глубокие и детальные объяснения поведения пользователей,
* **смещение ответов**: пользователи могут давать социально желательные ответы, что искажает результаты,
* **невозможность увидеть поведение**: опросы не позволяют увидеть, как пользователи взаимодействуют с интерфейсом.

**Интервью (Interviews)**

Преимущества:

* **глубокая информация**: интервью позволяют получить глубокие и детальные объяснения мотивов и мыслей пользователей,
* **гибкость**: можно адаптировать вопросы в зависимости от ответов пользователей, чтобы получить более полную картину,
* **личностный контакт**: интервью позволяют установить личный контакт с пользователями, что может помочь получить более честные и откровенные ответы.

Недостатки:

* **времязатратность**: интервью требуют много времени для проведения и анализа,
* **субъективность**: результаты интервью могут быть субъективными и зависеть от навыков интервьюера,
* **небольшой охват**: интервью сложно проводить с большим количеством пользователей, что ограничивает объем получаемых данных.

1. **Вывод**
   1. В ходе данной лабораторной работы было проведено тестирование удобства использования (usability) тестовой версии программного обеспечения (ПО) с целью оценки его качества.

# Лабораторная работа №3

## Исследование и классификация программных ошибок

1. **Цель работы**
   1. Исследование программного кода приложения на наличие ошибок,
   2. Классификация обнаруженных ошибок по типам,
   3. Исправление обнаруженных ошибок.
2. **Ход работы**
   1. Таблица 1 – Документация и классификация ошибок

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Описание ошибки | Тип ошибки | Местоположение ошибки | Некорректный код | Предложения по исправлению |
| Отсутствие знака « , » | Синтаксическая  ошибка | Метод DisplayUsers() | return | return, |
| Исключение не обработано | Функциональная ошибка | Метод Authorize() | if (userCredentials[username] == password) | if (userCredentials.TryGetValue(username, out string storedPassword) && storedPassword == password) |
| Отсутствие return | Функциональная ошибка | Метод Register() | if (string.IsNullOrEmpty(username))  {  Console.WriteLine("Имя пользователя не может быть пустым."),  } | if (string.IsNullOrEmpty(username))  {  Console.WriteLine("Имя пользователя не может быть пустым."),  return,  } |
| Отсутствие return | Функциональная ошибка | Метод Register() | if (userCredentials.ContainsKey(username))  { Console.WriteLine("Пользователь с таким именем уже существует."),  } | if (userCredentials.ContainsKey(username))  { Console.WriteLine("Пользователь с таким именем уже существует."),  return,  } |
| Отсутствие return | Функциональная ошибка | Метод Register() | if (password.Length < 8)  { Console.WriteLine("Пароль слишком короткий. Минимум 8 символов."),  } | if (password.Length < 8)  { Console.WriteLine("Пароль слишком короткий. Минимум 8 символов."),  return,  } |
| Неверное условие | Функциональная ошибка | Метод AddUser() | if (age < 0) | if (age <= 0) |
| Неверное условие | Логическая ошибка | Метод AddUser() | Console.WriteLine("Возраст не может быть отрицательным."), | Console.WriteLine("Возраст должен быть больше 0."), |
| Отсутствие return | Функциональная ошибка | Метод AddUser() | if (age <= 0)  { Console.WriteLine("Возраст должен быть больше 0."),  } | if (age <= 0)  { Console.WriteLine("Возраст должен быть больше 0."),  return,  } |
| Отсутствие знака « $ » | Синтаксическая  ошибка | Метод FindUser() | if (userFound != null)  { Console.WriteLine("Найден пользователь: {userFound.Name}, возраст {userFound.age}"),  } | if (userFound != null)  { Console.WriteLine($"Найден пользователь: {userFound.Name}, возраст {userFound.age}"),  } |
| Неправильный регистр | Синтаксическая  ошибка | Метод FindUser() | {userFound.age} | {userFound.Age} |
| Отсутствие методов доступа | Функциональная ошибка | Класс User | public string Name, | public string Name { get, set, } |
| Отсутствие методов доступа | Функциональная ошибка | Класс User | public int Age, | public int Age { get, set, } |
| Неправильное написание слова | Грамматическая ошибка | Метод Main(string[] args) | Console.WriteLine("1. Добавить пользователяя"), | Console.WriteLine("1. Добавить пользователя"), |
| Неправильная нумерация | Синтаксическая  ошибка | Метод Main(string[] args) | Console.WriteLine("2. Удалить пользователя"),  Console.WriteLine("2. Найти пользователя по имени"), | Console.WriteLine("2. Удалить пользователя"),  Console.WriteLine("3. Найти пользователя по имени"), |
| Неверный знак неравенства | Функциональная ошибка | Метод DisplayUsers() | for (int i = 0, i <= users.Count, i++) | for (int i = 0, i < users.Count, i++) |
| Отсутствие оператора else | Функциональная ошибка | Метод Register() | if (password.Length < 8)  {  Console.WriteLine("Пароль слишком короткий. Минимум 8 символов."),  return,  } userCredentials.Add(usernamе, password), Console.WriteLine("Пользователь успешно зарегистрирован."), | if (password.Length < 8)  {  Console.WriteLine("Пароль слишком короткий. Минимум 8 символов."),  return,  }  else  { userCredentials.Add(username, password),  Console.WriteLine("Пользователь успешно зарегистрирован."),  } |
| Исключение не обработано | Функциональная ошибка | Метод AddUser() | int age = Convert.ToInt32(Console.ReadLine()),  if (age <= 0) | int age,  while (!int.TryParse(Console.ReadLine(), out age)|| age < 0) |

* 1. Исправленный код программы

namespace ErrorsApp

{

using System,

using System.Collections.Generic,

internal class Program

{

static Dictionary<string, string> userCredentials = new Dictionary<string, string>(),

static List<User> users = new List<User>(),

static void Main(string[] args)

{

bool exit = false,

bool isAuthenticated = false,

while (!exit)

{

Console.WriteLine("\nМеню:"),

if (!isAuthenticated)

{

Console.WriteLine("1. Авторизоваться"),

Console.WriteLine("2. Зарегистрироваться"),

Console.WriteLine("3. Выйти из программы"),

}

else

{

Console.WriteLine("1. Добавить пользователя"),

Console.WriteLine("2. Удалить пользователя"),

Console.WriteLine("3. Найти пользователя по имени"),

Console.WriteLine("4. Вывести всех пользователей"),

Console.WriteLine("5. Выйти из учетной записи"),

Console.WriteLine("6. Выйти из программы"),

}

Console.Write("Выберите опцию: "),

string choice = Console.ReadLine(),

if (!isAuthenticated)

{

switch (choice)

{

case "1":

isAuthenticated = Authorize(),

break,

case "2":

Register(),

break,

case "3":

exit = true,

break,

default:

Console.WriteLine("Неверный выбор. Попробуйте снова."),

break,

}

}

else

{

switch (choice)

{

case "1":

AddUser(),

break,

case "2":

RemoveUser(),

break,

case "3":

FindUser(),

break,

case "4":

DisplayUsers(),

break,

case "5":

isAuthenticated = false,

Console.WriteLine("Вы вышли из учетной записи."),

break,

case "6":

exit = true,

break,

default:

Console.WriteLine("Неверный выбор. Попробуйте снова."),

break,

}

}

}

}

static bool Authorize()

{

Console.WriteLine("Введите имя пользователя:"),

string username = Console.ReadLine(),

Console.WriteLine("Введите пароль:"),

string password = Console.ReadLine(),

if (userCredentials.TryGetValue(username, out string storedPassword) && storedPassword == password)

{

Console.WriteLine("Успешная авторизация!"),

return true,

}

else

{

Console.WriteLine("Неверное имя пользователя или пароль."),

return false,

}

}

static void Register()

{

Console.WriteLine("Введите имя пользователя для регистрации:"),

string username = Console.ReadLine(),

if (string.IsNullOrEmpty(username))

{

Console.WriteLine("Имя пользователя не может быть пустым."),

return,

}

if (userCredentials.ContainsKey(username))

{

Console.WriteLine("Пользователь с таким именем уже существует."),

return,

}

Console.WriteLine("Введите пароль:"),

string password = Console.ReadLine(),

if (password.Length < 8)

{

Console.WriteLine("Пароль слишком короткий. Минимум 8 символов."),

return,

}

else

{

userCredentials.Add(username, password),

Console.WriteLine("Пользователь успешно зарегистрирован."),

}

}

static void AddUser()

{

Console.WriteLine("Введите имя пользователя:"),

string name = Console.ReadLine(),

Console.WriteLine("Введите возраст пользователя:"),

int age,

while (!int.TryParse(Console.ReadLine(), out age)|| age < 0) {

Console.WriteLine("Возраст должен быть больше 0."),

return,

}

users.Add(new User(name, age)),

Console.WriteLine("Пользователь добавлен."),

}

static void RemoveUser()

{

Console.WriteLine("Введите имя пользователя для удаления:"),

string name = Console.ReadLine(),

User userToRemove = users.Find(u => u.Name == name),

if (userToRemove != null)

{

users.Remove(userToRemove),

Console.WriteLine("Пользователь удален."),

}

else

{

Console.WriteLine("Пользователь не найден."),

}

}

static void FindUser()

{

Console.WriteLine("Введите имя пользователя для поиска:"),

string name = Console.ReadLine(),

User userFound = users.Find(u => u.Name == name),

if (userFound != null)

{

Console.WriteLine($"Найден пользователь: {userFound.Name}, возраст {userFound.Age}"),

}

else

{

Console.WriteLine("Пользователь не найден."),

}

}

static void DisplayUsers()

{

if (users.Count == 0)

{

Console.WriteLine("Список пользователей пуст."),

return,

}

for (int i = 0, i < users.Count, i++)

{

Console.WriteLine($"Имя: {users[i].Name}, Возраст: {users[i].Age}"),

}

}

}

class User

{

public string Name { get, set, }

public int Age { get, set, }

public User(string name, int age)

{

Name = name,

Age = age,

}

}

1. **Контрольные вопросы**
   1. Примеры синтаксических ошибок в ПО:

* отсутствие точки с запятой в конце строки кода,
* использование неправильного синтаксиса для объявления переменной,
* не закрытая скобка или кавычка,
* попытка доступа к несуществующему индексу массива.
  1. Ошибки времени выполнения (Runtime Errors) – это ошибки, которые возникают в процессе выполнения программы, когда программа не смогла правильно обработать какое-либо действие или запрос. Это может быть вызвано неверными операциями с данными, несуществующими ресурсами, переполнением стека и другими причинами.
  2. Методы для поиска ошибок ПО:
* тестирование: проведение различных видов тестов (юнит-тестирование, интеграционное тестирование, функциональное тестирование) для выявления ошибок в программе,
* отладка: использование отладочных инструментов для поиска и исправления ошибок в программном коде,
* ревью кода: проверка кода другими программистами для выявления потенциальных ошибок и улучшения качества кода,
* мониторинг и логирование: отслеживание работы программы в реальном времени и анализ журналов событий для выявления возможных проблем и ошибок.

1. **Вывод**
   1. В ходе данной работы был исследован программный код приложения на наличие ошибок,
   2. В ходе проделанной работы были классифицированы обнаруженные ошибки по типам,
   3. В ходе лабораторной работы были исправлены обнаруженные ошибки.

# Лабораторная работа №4

## Тестирование ПО методом «стеклянного (белого) ящика»

1. **Цель работы**
   1. Тестирование программного кода по методологии белого ящика,
   2. Исправление обнаруженных ошибок.
2. **Ход работы**
   1. Таблица 1 – Документация и классификация ошибок

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Описание ошибки | Тип ошибки | Местоположение ошибки | Некорректный код | Предложения по исправлению |
| Отсутствие return | Функциональная ошибка | Метод Register() | if (password.Length < 8)  { Console.WriteLine("Пароль слишком короткий. Минимум 8 символов."),  } | if (password.Length < 8)  { Console.WriteLine("Пароль слишком короткий. Минимум 8 символов."),  return,  } |
| Неверное условие | Функциональная ошибка | Метод AddUser() | if (age < 0) | if (age <= 0) |
| Неверное условие | Логическая ошибка | Метод AddUser() | Console.WriteLine("Возраст не может быть отрицательным."), | Console.WriteLine("Возраст должен быть больше 0."), |
| Возраст может быть меньше 0, но добавление не прерывается | Логическая ошибка | Метод AddUser() | if (age <= 0)  { Console.WriteLine("Возраст должен быть больше 0."),  } | if (age <= 0)  { Console.WriteLine("Возраст должен быть больше 0."),  return,  } |
| Неверная интерполяция | Ошибка в выводе | Метод FindUser() | if (userFound != null)  { Console.WriteLine("Найден пользователь: {userFound.Name}, возраст {userFound.age}"),  } | if (userFound != null)  { Console.WriteLine($"Найден пользователь: {userFound.Name}, возраст {userFound.age}"),  } |
| Неправильное обращение к полям объекта | Ошибка доступа к полям | Метод FindUser() | {userFound.age} | {userFound.Age} |
| Поля класса сделаны public вместо private | Ошибка проектирования | Класс User | public string Name, | private string Name, |
| Поля класса сделаны public вместо private | Ошибка проектирования | Класс User | public int Age, | private int Age, |
| Опечатка в пользовательском интерфейсе | Грамматическая ошибка | Метод Main(string[] args) | Console.WriteLine("1. Добавить пользователяя"), | Console.WriteLine("1. Добавить пользователя"), |
| Опечатка в пользовательском интерфейсе | Синтаксическая  ошибка | Метод Main(string[] args) | Console.WriteLine("2. Удалить пользователя"),  Console.WriteLine("2. Найти пользователя по имени"), | Console.WriteLine("2. Удалить пользователя"),  Console.WriteLine("3. Найти пользователя по имени"), |
| Цикл выполняется на одну итерацию больше | Логическая ошибка | Метод DisplayUsers() | for (int i = 0, i <= users.Count, i++) | for (int i = 0, i < users.Count, i++) |
| Отсутствие оператора else | Функциональная ошибка | Метод Register() | if (password.Length < 8)  {  Console.WriteLine("Пароль слишком короткий. Минимум 8 символов."),  return,  } userCredentials.Add(usernamе, password), Console.WriteLine("Пользователь успешно зарегистрирован."), | if (password.Length < 8)  {  Console.WriteLine("Пароль слишком короткий. Минимум 8 символов."),  return,  }  else  { userCredentials.Add(username, password),  Console.WriteLine("Пользователь успешно зарегистрирован."),  } |
| Неправильный ввод числа | Ошибка времени выполнения | Метод AddUser() | int age = Convert.ToInt32(Console.ReadLine()),  if (age <= 0) | int age,  while (!int.TryParse(Console.ReadLine(), out age)|| age < 0) |
| Пароли не хешируются | Ошибка безопасности | Метод Register() |  | byte[] passwordBytes = Encoding.UTF8.GetBytes(password),  byte[] saltBytes = GenerateSalt(),  byte[] hashedPasswordBytes = HashPassword(passwordBytes, saltBytes),  string hashedPassword = Convert.ToBase64String(hashedPasswordBytes), userCredentials.Add(username, hashedPassword + ":" + Convert.ToBase64String(saltBytes)), |

* 1. Исправленный код программы

namespace ErrorsApp

{

using System,

using System.Collections.Generic,

internal class Program

{

static Dictionary<string, string> userCredentials = new Dictionary<string, string>(),

static List<User> users = new List<User>(),

static void Main(string[] args)

{

bool exit = false,

bool isAuthenticated = false,

while (!exit)

{

Console.WriteLine("\nМеню:"),

if (!isAuthenticated)

{

Console.WriteLine("1. Авторизоваться"),

Console.WriteLine("2. Зарегистрироваться"),

Console.WriteLine("3. Выйти из программы"),

}

else

{

Console.WriteLine("1. Добавить пользователя"),

Console.WriteLine("2. Удалить пользователя"),

Console.WriteLine("3. Найти пользователя по имени"),

Console.WriteLine("4. Вывести всех пользователей"),

Console.WriteLine("5. Выйти из учетной записи"),

Console.WriteLine("6. Выйти из программы"),

}

Console.Write("Выберите опцию: "),

string choice = Console.ReadLine(),

if (!isAuthenticated)

{

switch (choice)

{

case "1":

isAuthenticated = Authorize(),

break,

case "2":

Register(),

break,

case "3":

exit = true,

break,

default:

Console.WriteLine("Неверный выбор. Попробуйте снова."),

break,

}

}

else

{

switch (choice)

{

case "1":

AddUser(),

break,

case "2":

RemoveUser(),

break,

case "3":

FindUser(),

break,

case "4":

DisplayUsers(),

break,

case "5":

isAuthenticated = false,

Console.WriteLine("Вы вышли из учетной записи."),

break,

case "6":

exit = true,

break,

default:

Console.WriteLine("Неверный выбор. Попробуйте снова."),

break,

}

}

}

}

static bool Authorize()

{

Console.WriteLine("Введите имя пользователя:"),

string username = Console.ReadLine(),

Console.WriteLine("Введите пароль:"),

string password = Console.ReadLine(),

if (userCredentials.TryGetValue(username, out string storedPassword) && storedPassword == password)

{

Console.WriteLine("Успешная авторизация!"),

return true,

}

else

{

Console.WriteLine("Неверное имя пользователя или пароль."),

return false,

}

}

static void Register()

{

Console.WriteLine("Введите имя пользователя для регистрации:"),

string username = Console.ReadLine(),

if (string.IsNullOrEmpty(username))

{

Console.WriteLine("Имя пользователя не может быть пустым."),

return,

}

if (userCredentials.ContainsKey(username))

{

Console.WriteLine("Пользователь с таким именем уже существует."),

return,

}

Console.WriteLine("Введите пароль:"),

string password = Console.ReadLine(),

if (password.Length < 8)

{

Console.WriteLine("Пароль слишком короткий. Минимум 8 символов."),

return,

}

else

{

userCredentials.Add(username, password),

Console.WriteLine("Пользователь успешно зарегистрирован."),

}

}

static void AddUser()

{

Console.WriteLine("Введите имя пользователя:"),

string name = Console.ReadLine(),

Console.WriteLine("Введите возраст пользователя:"),

int age,

while (!int.TryParse(Console.ReadLine(), out age)|| age < 0) {

Console.WriteLine("Возраст должен быть больше 0."),

return,

}

users.Add(new User(name, age)),

Console.WriteLine("Пользователь добавлен."),

}

static void RemoveUser()

{

Console.WriteLine("Введите имя пользователя для удаления:"),

string name = Console.ReadLine(),

User userToRemove = users.Find(u => u.Name == name),

if (userToRemove != null)

{

users.Remove(userToRemove),

Console.WriteLine("Пользователь удален."),

}

else

{

Console.WriteLine("Пользователь не найден."),

}

}

static void FindUser()

{

Console.WriteLine("Введите имя пользователя для поиска:"),

string name = Console.ReadLine(),

User userFound = users.Find(u => u.Name == name),

if (userFound != null)

{

Console.WriteLine($"Найден пользователь: {userFound.Name}, возраст {userFound.Age}"),

}

else

{

Console.WriteLine("Пользователь не найден."),

}

}

static void DisplayUsers()

{

if (users.Count == 0)

{

Console.WriteLine("Список пользователей пуст."),

return,

}

for (int i = 0, i < users.Count, i++)

{

Console.WriteLine($"Имя: {users[i].Name}, Возраст: {users[i].Age}"),

}

}

}

class User

{

private string Name,

private int Age,

public User(string name, int age)

{

Name = name,

Age = age,

}

}

1. **Ответы на контрольные вопросы**
   1. Метод тестирования белого ящика (или структурное тестирование) — это метод, при котором тестировщик имеет доступ к внутренней структуре, архитектуре и коду программы. Он основывается на анализе внутреннего исполнения программного обеспечения, позволяет выявить ошибки в логике, алгоритмах и потоках выполнения.

Метод черного ящика, напротив, фокусируется на тестировании функциональности приложения без знания его внутренней структуры. Тестировщик проверяет только входные и выходные данные, а также интерфейсы, не углубляясь в детали реализации.

* 1. Достоинства и недостатки тестирования

Достоинства:

1. детальный анализ: позволяет глубже понять внутреннюю логику работы приложения и выявить возможные ошибки, которые могут оставаться незамеченными при тестировании черного ящика,
2. улучшение качества кода: способствует выявлению недостатков в алгоритмах и неэффективности кода,
3. обеспечение покрытия кода: тесты могут быть спроектированы так, чтобы охватить все ветвления и условия в коде, что повышает вероятность обнаружения ошибок,
4. автоматизация: часто лучше поддается автоматизации тестирования, поскольку все внутренние условия известны.

Недостатки:

1. сложность: требует глубоких знаний в области программирования и структуры тестируемого кода,
2. временные затраты: может занять много времени для создания тестов, особенно для больших проектов,
3. исключения: может не выявлять ошибки, связанные с неправильной реализацией требований, так как фокусируется на внутреннем коде,
4. посредственная проверка пользовательского опыта: меньшее внимание уделяется взаимодействию пользователя с приложением, что может привести к упущению критически важных аспектов функциональности.
5. **Вывод**
   1. В ходе лабораторной работы было выполнено тестирование программного кода по методологии белого ящика,
   2. В ходе лабораторной работы были исправлены обнаруженные ошибки.

# Лабораторная работа №5

## Тестирование ПО методом «стеклянного ящика»

1. **Цель работы**
   1. Тестирование программного кода по методологии белого ящика.
2. **Ход работы**
   1. Тест-кейс №1
3. **Идентификатор теста**: LB-001.
4. **Название теста**: тестирование ветвления в методе вывода списка книг.
5. **Цель теста**: проверить корректность ветвления в методе ShowBooks().
6. **Предусловия**:

* метод ShowBooks() должен быть реализован,
* список книг в библиотеке должен быть заранее заполнен.

1. **Описание тестируемого метода (модуля)**: метод ShowBooks() принимает список книг, выводит нумерованный список книг, если книги в библиотеке присутствуют, в ином случае выводит ошибку «В библиотеке нет книг».

static void ShowBooks(List<Book> library)

{

if (library.Count == 0)

{

Console.WriteLine("В библиотеке нет книг."),

}

else

{

Console.WriteLine("\nСписок книг в библиотеке:"),

for (int i = 0, i < library.Count, i++)

{

Console.WriteLine($"{i + 1}. {library[i]}"),

}

}

}

1. **Шаги выполнения теста**:

* Шаг 1: передать в метод список книг.
* Шаг 2: проверить, что метод корректно выводит список книг.
* Шаг 3: не передавать список книг в метод.
* Шаг 4: проверить, что метод выводит ошибку.

1. **Ожидаемый результат**:

* если книги в библиотеке есть, то выводится список книг,
* если книг в библиотеке нет, то выводится сообщение «В библиотеке нет книг».

1. **Фактический результат**:

* если книги в библиотеке есть, то выводится список книг (тест пройден),
* если книг в библиотеке нет, то выводится сообщение «В библиотеке нет книг» (тест пройден).

1. **Примечания**: метод работает корректно, все ветвления отработали согласно ожиданиям.
2. **Статус**: пройден.
   1. Тест-кейс №2
3. **Идентификатор теста**: LB-002
4. **Название теста**: тестирование ветвления в методе удаления книг.
5. **Цель теста**: проверить корректность ветвления в методе RemoveBook().
6. **Предусловия**:
   * метод RemoveBook() должен быть реализован,
   * список книг в библиотеке должен быть заранее заполнен.
7. **Описание тестируемого метода (модуля)**: Метод RemoveBook() предназначен для удаления книги из списка библиотеки. Он принимает в качестве параметра список книг (List<Book> library).

static void RemoveBook(List<Book> library)

{

ShowBooks(library),

if (library.Count == 0) return,

Console.Write("Введите номер книги для удаления: "),

if (int.TryParse(Console.ReadLine(), out int index) && index > 0 && index <= library.Count)

{

library.RemoveAt(index - 1),

Console.WriteLine("Книга удалена."),

}

else

{

Console.WriteLine("Неверный ввод."),

}

}

1. **Шаги выполнения теста**:
   * Шаг 1: метод вызывает ShowBooks(library), который выводит на экран все книги, находящиеся в библиотеке.
   * Шаг 2: проверка наличия книг: если библиотека пуста, метод завершает выполнение, не производя никаких действий.
   * Шаг 3: запрос ввода номера книги: пользователю предлагается ввести номер книги, которую он хочет удалить.
   * Шаг 4: проверяется, является ли введенное значение целым числом, что введенный индекс больше 0 и меньше или равен количеству книг в библиотеке.
   * Шаг 5: если ввод корректен, книга удаляется из списка
   * Шаг 6: после успешного удаления выводится сообщение "Книга удалена."
   * Шаг 7: если ввод некорректен, выводится сообщение "Неверный ввод.
2. **Ожидаемый результат**:
   * если метод находит книгу по индексу, удаляет ее и выводит сообщение «Книга удалена»,
   * если метод не находит книгу по индексу, то выводит сообщение «Неверный ввод».
3. **Фактический результат**:
   * при наличии книги под введенным индексом «1» она удаляется (тест пройден),
   * при отсутствии книги под введенным индексом «10058935786656» выводится сообщение «Неверный ввод» (тест пройден).
4. **Примечания**: метод работает корректно, все ветвления отработали согласно ожиданиям.
5. **Статус**: пройден.
   1. Тест-кейс №3
6. **Идентификатор теста**: LB-003
7. **Название теста**: тестирование ветвления в методе для добавления книг.
8. **Цель теста**: проверить корректность ветвления в методе AddBook().
9. **Предусловия**:
   * Метод AddBook() должен быть реализован.
10. **Описание тестируемого метода (модуля)**: Метод AddBook() предназначен для добавления книг. Он принимает в качестве параметра список книг (List<Book> library).

static void AddBook(List<Book> library)

{

Console.Write("Введите название книги: "),

string title = Console.ReadLine(),

Console.Write("Введите автора книги: "),

string author = Console.ReadLine(),

library.Add(new Book(title, author)),

Console.WriteLine("Книга добавлена."),

}

1. **Шаги выполнения теста**:
   * Шаг 1: метод выводит сообщение "Введите название книги: " и ожидает ввода от пользователя.
   * Введенное значение сохраняется в переменную title.
   * Запрос автора книги:
   * Шаг 2: метод выводит сообщение "Введите автора книги: " и ожидает ввода от пользователя.
   * Введенное значение сохраняется в переменную author.
   * Шаг 3: метод создает новый объект класса Book, используя введенные значения title и author.
   * Шаг 4: новый объект Book добавляется в список library с помощью метода Add.
   * Шаг 5: метод выводит сообщение "Книга добавлена.", подтверждая успешное выполнение операции.
2. **Ожидаемый результат**:
   * если пользователь вводит название и автора книги, то книга добавляется в библиотеку, и выводится сообщение «Книга добавлена»,
   * если пользователь не вводит название или автора книги, то книга не добавляется в библиотеку, и выводится сообщение «Некорректный ввод»
3. **Фактический результат**:
   * если пользователь вводит название и автора книги, то книга добавляется в библиотеку, и выводится сообщение «Книга добавлена» (тест пройден),
   * если пользователь не вводит название или автора книги, то книга не добавляется в библиотеку, и выводится сообщение «Некорректный ввод» (тест не пройден).
4. **Примечания**: метод работает некорректно, добавляет книгу с пустыми данными.
5. **Статус**: не пройден.
   1. Тест-кейс №4
6. Идентификатор теста: LB-004
7. Название теста: тестирование ветвления в методе Main.
8. Цель теста: проверить корректность ветвления в методе Main().
9. Предусловия: метод Main() должен быть реализован.
10. Описание тестируемого метода (модуля**)**: метод Main является основной точкой входа в программу, которая управляет библиотекой книг. Он предоставляет пользователю интерфейс для взаимодействия с библиотекой через консольное меню.

static void Main(string[] args)

{

List<Book> library = new List<Book>(),

bool running = true,

while (running)

{

Console.WriteLine("\nБиблиотека книг"),

Console.WriteLine("1. Добавить книгу"),

Console.WriteLine("2. Удалить книгу"),

Console.WriteLine("3. Показать все книги"),

Console.WriteLine("4. Выход"),

Console.Write("Выберите действие: "),

string choice = Console.ReadLine(),

switch (choice)

{

case "1":

AddBook(library),

break,

case "2":

RemoveBook(library),

break,

case "3":

ShowBooks(library),

break,

case "4":

running = false,

break,

default:

Console.WriteLine("Неверный выбор. Пожалуйста, попробуйте снова."),

break,

}

}

}

1. Шаги выполнения теста:
   * Шаг 1: создается новый список книг library типа List<Book>, который будет использоваться для хранения добавленных книг.
   * Шаг 2: внутри цикла выводится меню с доступными действиями.
   * Шаг 3: программа считывает выбор пользователя и сохраняет его в переменной choice.
   * Шаг 4: в зависимости от введенного значения выполняется соответствующее действие.
2. Ожидаемый результат:
   * если пользователь вводит число, которому соответствует определенное действие, то оно выполняется,
   * если пользователь вводит число, которому не соответствует ни одно из действий, то выводится сообщение "Неверный выбор. Пожалуйста, попробуйте снова.".
3. Фактический результат:
   * если пользователь вводит число, которому соответствует определенное действие, то оно выполняется (тест пройден),
   * если пользователь вводит число, которому не соответствует ни одно из действий, то выводится сообщение "Неверный выбор. Пожалуйста, попробуйте снова." (тест пройден).
4. **Примечания**: метод работает корректно, все ветвления отработали согласно ожиданиям.
5. **Статус**: пройден.
6. **Вывод**
   1. В ходе проделанной работы было проведено тестирование программного кода по методологии белого ящика.

# Лабораторная работа №6

## Модульное тестирование

1. **Цель работы**
   1. Тестирование программного кода при помощи библиотеки xUnit.
2. **Ход работы**
   1. Создали форк репозитория https://github.com/ReyRom/UnitTesting.
   2. Клонировали репозиторий на локальное устройство.
   3. Разработали набор модульных тестов и использованием xUnit для метода библиотеки классов.
   4. Разработали модульный тест для проверки одиночного набора данных с использованием атрибута [Fact].
   5. Разработали модульный тест для проверки нескольких наборов данных с использованием атрибутов [Theory] и [InlineData].
   6. Разработали модульный тест для проверки генерации исключения при помощи Assert.Throws.
3. **Ответы на контрольные вопросы**
   1. Атрибут [Fact] обозначает метод как тест, который не принимает параметров и выполняется один раз. Он используется для тестирования простых сценариев, где нет необходимости в различных входных данных.
   2. Атрибут [Theory] позволяет передавать параметры в тестовые методы, что позволяет проводить тестирование с множеством наборов данных. Это полезно для проверки одного и того же кода с различными входными значениями.
   3. Методы Assert предназначены для проверки условий в тестах:

* assertEquals(expected, actual) проверяет равенство двух значений,
  + assertTrue(condition) проверяет, что условие истинно,
  + assertFalse(condition) проверяет, что условие ложно.

1. **Вывод**
   1. В ходе данной лабораторной работы был протестирован программный код при помощи библиотеки xUnit.

# Лабораторная работа №7

## Интеграционное тестирование

1. **Цель работы**
   1. Интеграционное тестирование программного кода при помощи библиотеки xUnit и Moq.
2. **Ход работы**
   1. Создали форк репозитория https://github.com/ReyRom/UnitTesting.
   2. Клонировали репозиторий на локальное устройство.
   3. Разработали набор тестов с использованием xUnit для методов библиотеки классов.
   4. При разработке тестов использовали подмену зависимостей при помощи Moq, использовали проверку внесения изменений в репозиторий,
3. **Ответы на контрольные вопросы**
   1. Для настройки зависимостей в Moq используется метод Setup, который позволяет задать поведение подменяемого объекта.
   2. Для проверки выполнения действий в зависимостях применяется метод Verify, который подтверждает, что определенные методы были вызваны.
   3. Интеграционное тестирование – это процесс проверки взаимодействия между модулями или компонентами системы, чтобы убедиться, что они работают корректно вместе. Оно обычно проводится после модульного тестирования и фокусируется на выявлении дефектов в интерфейсах и взаимодействиях между компонентами.
4. **Вывод**
   1. В ходе данной лабораторной работы было проведено интеграционное тестирование программного кода при помощи библиотеки xUnit и Moq.

# Лабораторная работа №8

## Тестирование ПО методом «черного ящика»

1. **Цель работы**
   1. Освоить процесс тестирования методом «черного ящика»
2. **Ход работы**

|  |  |  |
| --- | --- | --- |
| **Действия** | **Ожидаемый результат** | **Полученный результат** |
| Нажатие кнопки пропустить вопрос. | Если пропустил вопрос, то в конце на них все равно нужно ответить. | Программа выдает результат, не позволяя ответить на пропущенные вопросы. |
| Добавление ответа на вопрос в админ-панели. | Программа не даст добавить ответ, если все нужные поля не заполнены. | Программа позволяет добавить ответ, когда он неправильно оформлен. |
| Переход к следующему вопросу. Примечание: после удаления всех вопросов. | Программа выдаст сообщение, что вопросов больше нет. | Вылет из программы. |
| Неконтролируемое добавление количества ответов на вопрос. | Программа не позволит добавить большое количество ответов на вопрос. | Программа добавляет большое количество ответов на вопрос. |
| Удаление вопроса. Примечание: создание нового теста, нет вопросов. | Программа сообщит, что вопросы еще не созданы. | Программа не обрабатывает данную ошибку. |
| Удаление теста. Примечание: он еще не создан. | Программа удалит тест и будет исправна. | Вылет из программы. |
| Добавление вопроса | Программа не должна позволять добавить вопрос, если он полностью не сформирован. | Программа позволяет добавить пустой вопрос. |
| Добавление вопроса. Примечание: после того, как удалили все остальные вопросы. | Программа добавит новый вопрос. | Программа не дает добавить новый вопрос. |
| Удаление ответа на вопрос. Примечание: после того, как удалили все остальные вопросы. | Программа удалит ответ на вопрос. | Вылет из программы. |
| Добавление нового теста. Примечание: до этого был написан шаблон другого теста, но не создан. | Программа предложит сохранить шаблон теста, прежде чем приступить к созданию нового теста. | Потеря прошлого шаблона. Создание нового теста. |
| Сохранение вопроса. Примечание: были добавлены ответы на вопросы, но не везде. | Программа должна предупреждать о не заполнении ячеек в некоторых ответах или удалять пустые ответы. | Программа создает вопрос с пустыми и непустыми ответами. |
| Прохождение теста. Примечание: был добавлен тест с пустыми вариантами ответов. | Программа не должна учитывать пустые ответы (их вообще быть не должно). | Программа считает неправильный ответ правильным. |
| Добавление нового теста. Примечание: название теста состоит из пробелов. | Программа выдаст предупреждение о некорректном названии теста. | Программа создает тест с пробелами. |
| Изменение названия у лекции. Примечание: ввели большое количество символов в названии. | В программе предусмотрено ограничение количества символов для названия лекции. | Из-за большого количества символов «плывет» интерфейс. |
| Удаление всех лекций. | Программа удалит все лекции и будет исправна. | Вылет из программы. |

1. **Ответы на контрольные вопросы**
   1. Тестирование чёрного ящика — стратегия (метод) тестирования функционального поведения объекта (программы, системы) с точки зрения внешнего мира, при котором не используется знание о внутреннем устройстве (коде) тестируемого объекта.
   2. Преимущества и недостатки у тестирования черного ящика:

Преимущества: тестирование черного ящика может проверить дополнительные аспекты программного обеспечения, помимо функций и функциональности.

Недостатки: тестовые случаи крайне сложно разрабатывать без четких и кратких спецификаций. Тестирование каждого возможного входного потока невозможно, поскольку это занимает много времени, и в конечном итоге это оставит многие пути программы непроверенными. Результаты могут быть иногда переоценены.

* 1. Типы ошибок при тестировании черного ящика, связанные с функциональностью программного обеспечения:
* ошибки в функциональности,
* ошибки в интерфейсе,
* ошибка обработки вводных данных,
* ошибка в производительности,
* ошибка безопасности,
* ошибки в совместимости,
* ошибки в локализации,
* ошибки в интеграции.

1. **Вывод**
   1. В ходе данной лабораторной работы освоили процесс тестирования методом «черного ящика».

# Лабораторная работа №9

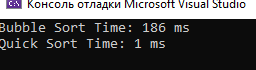
## Тестирование программного модуля по определенному сценарию

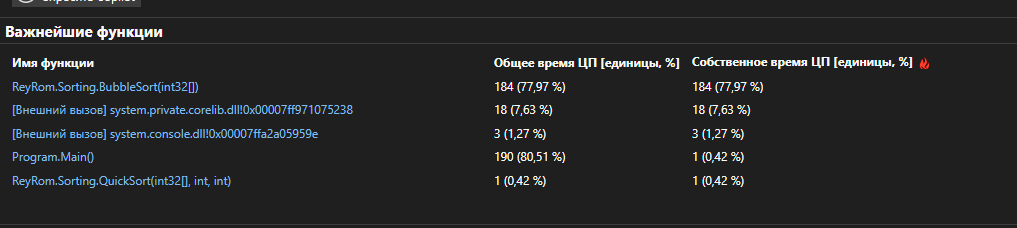
1. **Цель работы**
   1. Освоить процесс написания сценариев тестирования,
   2. Освоить процесс тестирования по определенному сценарию.
2. **Ход работы**
   1. Загрузили тестовое приложение из репозитория по ссылке https://github.com/ReyRom/EducationProjects/tree/master/AuthApp.
   2. Описали сценарии тестирования функций регистрации и авторизации.
   3. Следуя шагам сценария написать автоматизированные тесты для пользовательского интерфейса с использованием фреймворка FlaUI.
3. **Сценарии тестирования**
   1. Регистрация:
4. запустить приложение,
5. проверить наличие поля для ввода логина, пароля, подтверждения пароля и кнопки регистрации,
6. нажать кнопку «Зарегистрироваться»,
7. проверить, что система отображает сообщение об отсутствии информации в полях,
8. ввести логин в соответствующее ему поле,
9. нажать кнопку «Зарегистрироваться»,
10. проверить, что система отображает сообщение об отсутствии информации в других полях,
11. ввести пароль в соответствующее ему поле,
12. нажать кнопку «Зарегистрироваться»,
13. проверить, что система отображает сообщение об отсутствии информации в подтверждении пароля,
14. ввести несовпадающее подтверждение пароля в соответствующее ему поле,
15. нажать кнопку «Зарегистрироваться»,
16. проверить, что система отображает сообщение о несовпадении паролей,
17. ввести неправильные пароль и подтверждение пароля в соответствующие им поля,
18. нажать кнопку «Зарегистрироваться»,
19. проверить, что система отображает сообщение о исключениях в пароле,
20. ввести корректные данные в поля,
21. нажать кнопку «Зарегистрироваться»,
22. проверить, что система отображает сообщение о успешной регистрации,
23. ввести пароль и подтверждение пароля в соответствующие им поля, не указывая логин,
24. нажать кнопку «Зарегистрироваться»,
25. проверить, что система отображает сообщение об отсутствии информации в логине.
    1. Авторизация:
26. запустить приложение,
27. проверить наличие поля для ввода логина, пароля и кнопки входа в систему,
28. нажать кнопку «войти»,
29. проверить, что система отображает сообщение об отсутствии информации в полях,
30. ввести логин в соответствующее ему поле,
31. нажать кнопку «войти»,
32. проверить, что система отображает сообщение об отсутствии информации в другом поле,
33. ввести пароль в соответствующее ему поле, не указывая логин,
34. нажать кнопку «войти»,
35. проверить, что система отображает сообщение об отсутствии информации в другом поле,
36. ввести неверный логин в соответствующее ему поле,
37. нажать кнопку «войти»,
38. проверить, что система отображает сообщение о неверном логине или пароле,
39. ввести неверный пароль в соответствующее ему поле,
40. нажать кнопку «войти»,
41. проверить, что система отображает сообщение о неверном логине или пароле,
42. ввести некорректные данные в соответствующие поля,
43. нажать кнопку «войти»,
44. проверить, что система отображает сообщение о неверном логине или пароле,
45. ввести корректные данные в поля,
46. нажать кнопку «войти»,
47. проверить, что система отображает сообщение об успешной авторизации.
48. **Вывод**
    1. В ходе проделанной работы были освоен процесс написания сценариев тестирования;
    2. В ходе проделанной работы были освоен процесс тестирования по определенному сценарию.

# Лабораторная работа №10

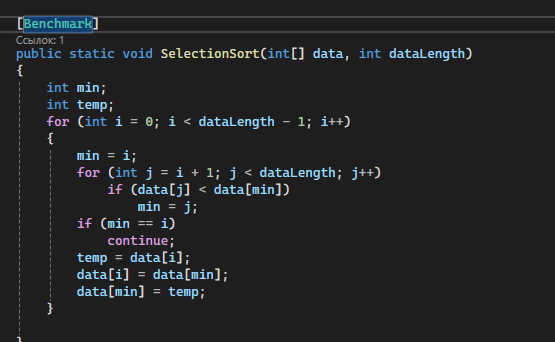
## Отладка ПО с использованием инструментальных средств.

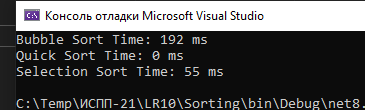
1. **Цель работы**
   1. Освоить процесс отладки ПО с использованием инструментальных средств.
2. **Ход работы**
   1. Создали форк, а затем клонировали репозиторий по ссылке https://github.com/ReyRom/Profiling.
   2. Запустить проект Sorting. С использованием профилировщика производительности и бенчмарка оценили эффективность и время, затраченное на выполнение алгоритмов сортировки.

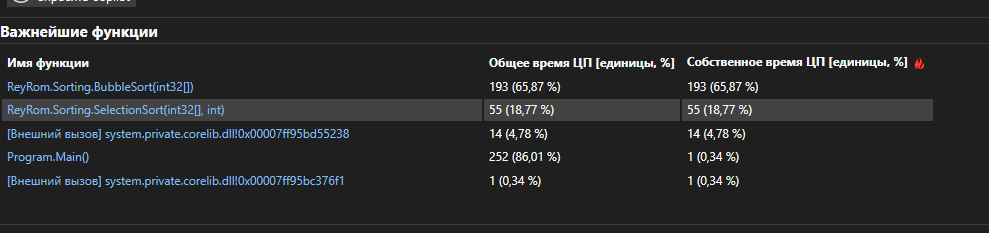
****

****

* 1. Разработали метод, выполняющий сортировку способом, не представленным в программе и повторить профилирование.
  2. Сортировка методом выборки:

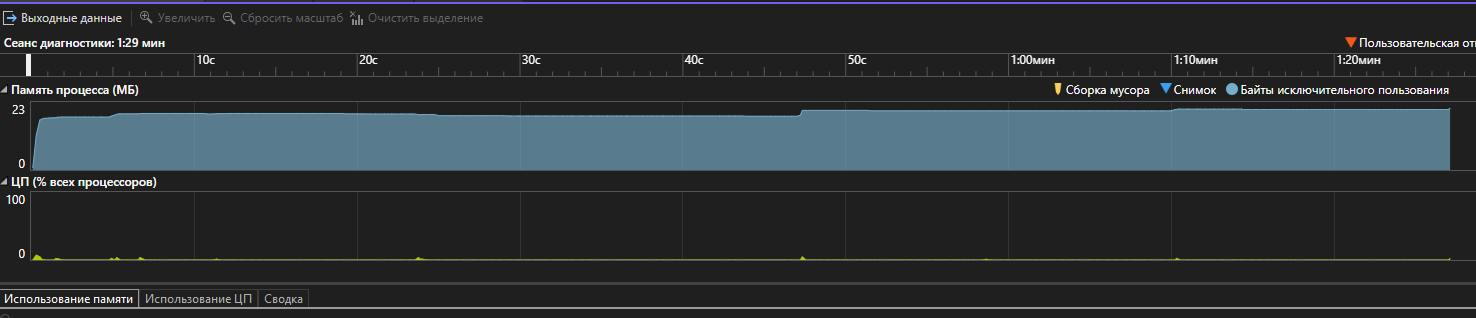


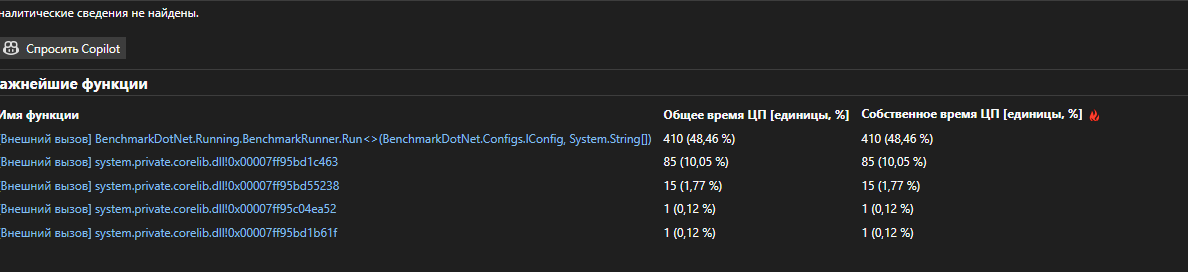
****

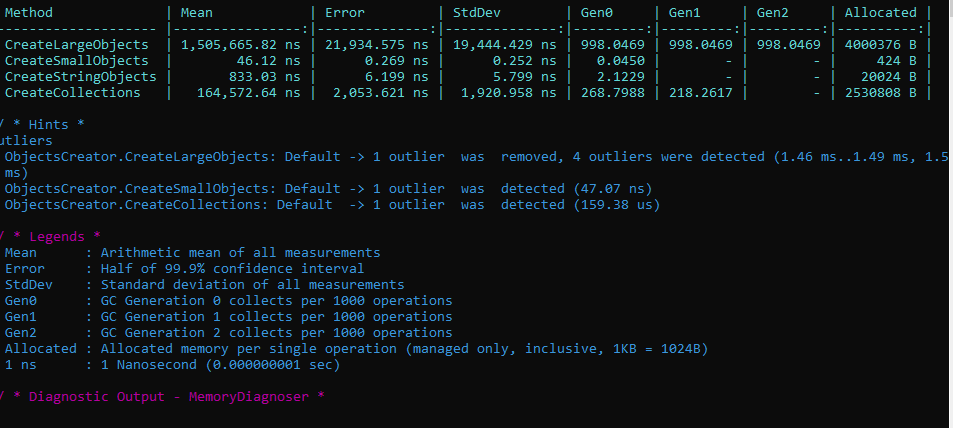
****

Исходя из полученных данных, можно сказать, что пузырьковая медленнее сортировка тратит больше ресурсов на выполнение, чем другие методы сортировок.

* 1. Запустили проект Objects. С использованием профилировщика производительности и бенчмарка оценили время, затраченное на создание различных объектов.

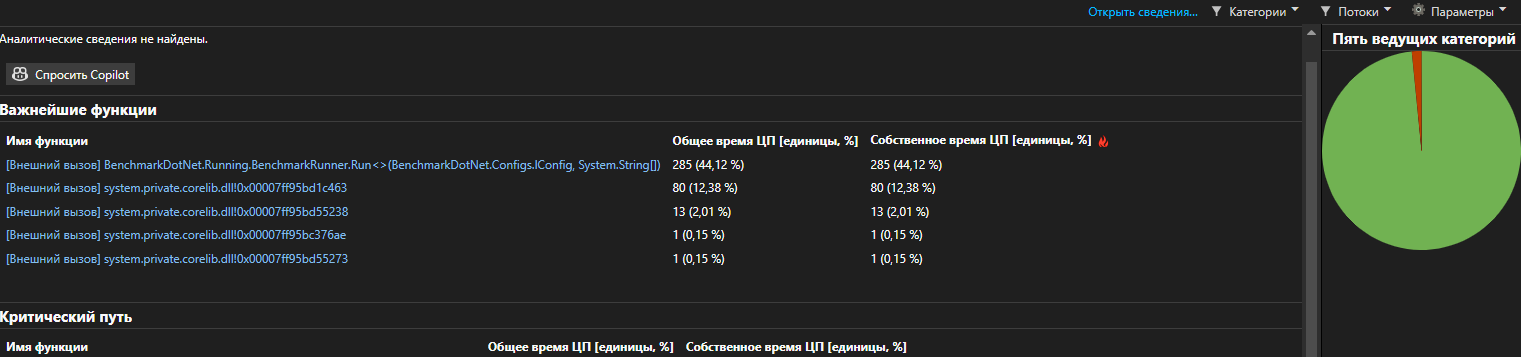


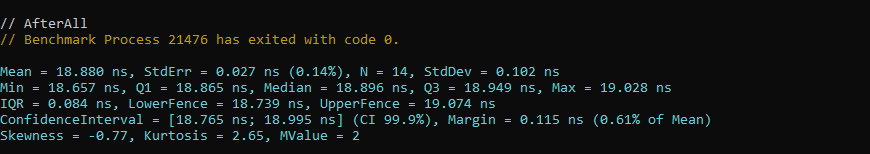
****



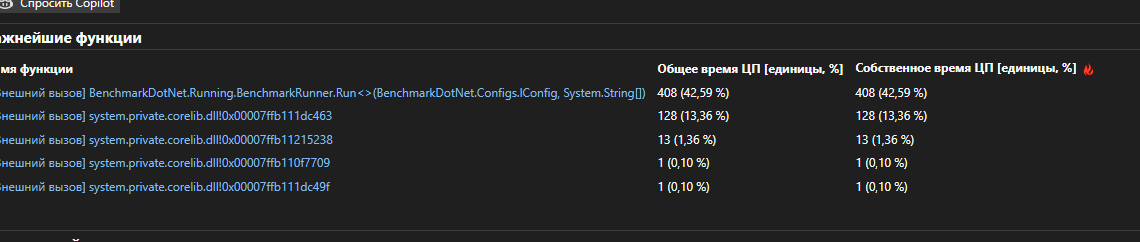
Исходя из этих данных, можно сказать, что для создания больших объектов требуется больше времени.

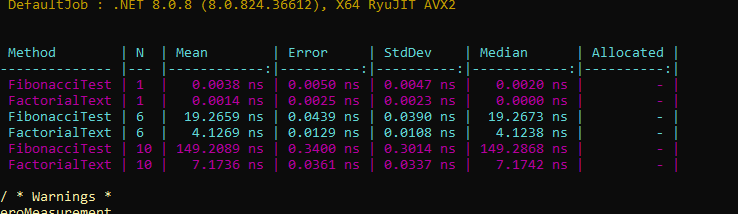
* 1. Запустили проект Recursion. С использованием профилировщика производительности и бенчмарка оценили время, затраченное на выполнение рекурсивной функции для больших значений.



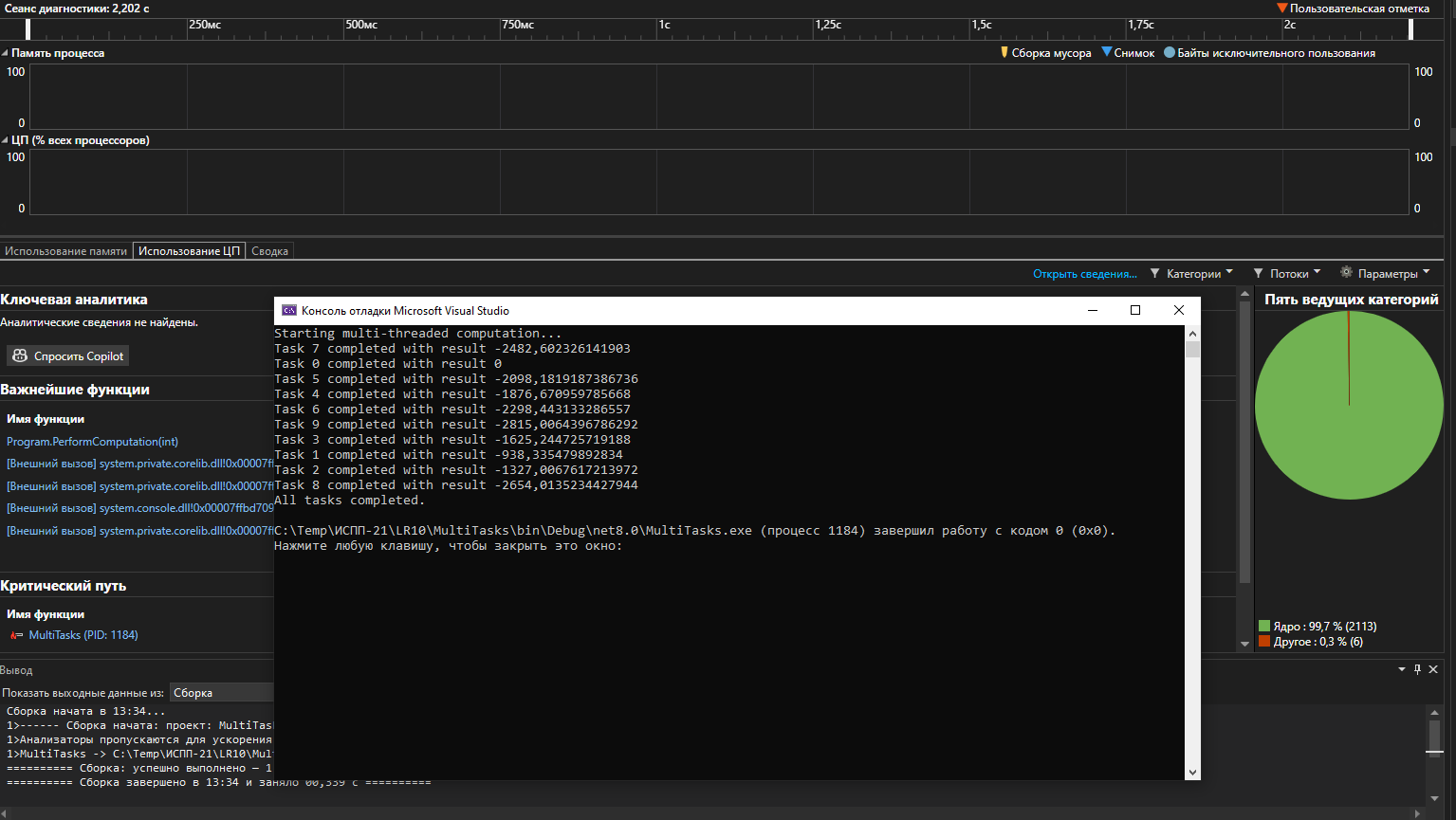


* 1. Разработали рекурсивный метод, не представленный в программе, вызвали его для большого значения параметра и повторили профилирование.

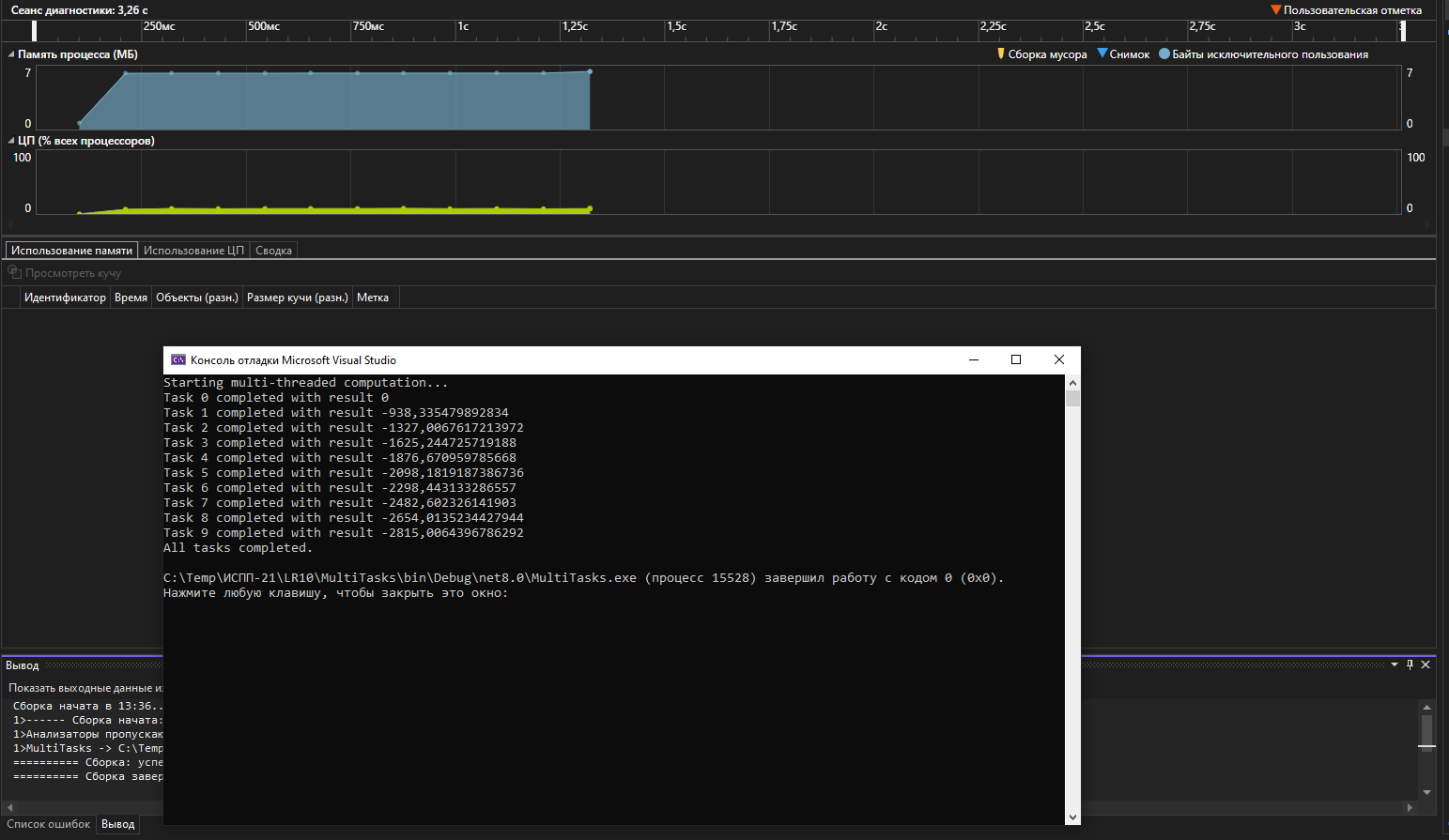
****

****

* 1. Запустили проект MultiTasks. С использованием профилировщика производительности оценили время, затраченное на выполнение длительных асинхронных операций.



* 1. Выполнить данное вычисление в синхронном режиме сравнить результаты, записать данные в отчет



1. **Ответы на контрольные вопросы**
   1. Профилировщик производительности в Visual Studio можно запустить в меню «Отладка» → «Профилировщик производительности»
   2. Использование памяти в профилировщике можно посмотреть в «Использование памяти»
   3. Чтобы просмотреть использование ЦП в профилировщике нужно выбрать «Использование ЦП»
2. **Вывод**
   1. В ходе лабораторной работы освоили процесс отладки ПО с использованием инструментальных средств.

# Лабораторная работа №13

## Испытание ПО методом «Регрессионного тестирования».

1. **Цель работы**
   1. Освоить процесс применения регрессионного тестирования.
2. **Ход работы**
   1. Клонировали репозиторий.
   2. Последовательно внесли изменения в проект, изменяя или добавляя новые регрессионные тесты по мере внесения изменений фиксируя состояние тестов на каждом этапе.
   3. Добавить проверки поочередно, проверяя как изменения повлияли на ранее работающий код
3. **Ответы на контрольные вопросы**
   1. Регрессионное тестирование – это процесс повторного выполнения тестов на программном обеспечении для обнаружения новых ошибок, возникших в результате внесения изменений или доработок.
4. **Вывод**
   1. В ходе лабораторной работы был освоен процесс применения регрессионного тестирования.