Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИТ

Отчёт

по лабораторной работе№ 6

«Особенности тестирования ПО»

Выполнил:

ст. гр. ИС/б-41-о

Малиновский А.А.

Проверил:

Лагуткина Т. В.

Севастополь

2015

1. ЦЕЛЬ РАБОТЫ

Изучить особенности тестирования программного обеспечения; рассмотреть виды тестирования.

1. ВАРИАНТ ЗАДАНИЯ

Задача 1. Для целочисленной прямоугольной матрицы определить максимальное из чисел, встречающихся в заданной матрице один раз.

Задача 2. Дана строка. Подсчитать, сколько раз среди данных символов встречается буква а.

Задача 3. Программа, которая считывает из текстового файла пять предложений и выводит их в обратном порядке.

1. ВЫПОЛНЕНИЕ
   1. Этап 1 – Формирование областей эквивалентности и построения тестовых последовательностей

По варианту задаются требования к программам. Для каждой из них необходимо:

1. Написать программу, выполняющую заданные действия.
2. Определить области эквивалентности входных данных.
3. Составить примеры тестовых последовательностей.

ОБЛАСТИ ЭКВИВАЛЕНТНОСТИ ВХОДНЫХ ДАННЫХ

Определим области эквивалентности:

Задача 1.

1. По размеру матрицы:
   1. Пустая матрица
   2. Матрица состоит из одного элемента;
   3. Матрица состоит более чем из одного элемента.
2. По наличию максимального элемента в матрице
   1. Все элементы равны
   2. Существует один максимальный элемент
   3. Существует более чем один максимальный элемент

Задача 2.

Определим области эквивалентности:

1. По размеру строки:
   1. Пустая строка
   2. Строка, состоящая из одного символа
   3. Строка, состоящая из более чем одного символа
2. По наличию буквы ‘а’
   1. В строке нет буквы ‘а’
   2. В строке присутствует одна буква ‘а’
   3. В строке присутствует более чем одна буква ‘а’

Задача 3.

1. По размеру текста
   1. Пустой текст
   2. В тексте одно предложение
   3. В тексте более одного предложения
2. По наличию знаков препинания в предложении
   1. Знаки препинания отсутствуют
   2. Один знак препинания
   3. Более одного знака препинания

ТЕКСТ ПРОГРАММЫ

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

/\*

11)Задача 1. Для целочисленной прямоугольной матрицы определить максимальное из чисел, встречающихся в заданной матрице один раз.

Задача 2. Дана строка. Подсчитать, сколько раз среди данных символов встречается буква а.

Задача 3. Программа, которая считывает из текстового файла пять предложений и выводит их в обратном порядке.

\*/

namespace ConsoleApp\_TPO1

{

class Program\_TPO1

{

static void Main(string[] args)

{

#region #1

try

{

Console.WriteLine("Задача 1");

Console.WriteLine("Для целочисленной прямоугольной матрицы определить максимальное из чисел, встречающихся в заданной матрице один раз.");

Console.WriteLine("Введите имя файла с матрицей");

string file\_name\_mtx = Console.ReadLine();

StreamWriter stream\_writer\_M = new StreamWriter(file\_name\_mtx + "\_answer.txt");

int MaxElemOnceTime = MaxOnceElement(ReadTheMatrix(file\_name\_mtx+".txt"));

if (MaxElemOnceTime == 0)

{

stream\_writer\_M.WriteLine("Програмист таки кривой лах!");

Console.WriteLine("Програмист таки кривой лах!");

}

else

{

Console.Write("Таким числом будет -> ");

Console.WriteLine(MaxElemOnceTime);

stream\_writer\_M.Write("Таким числом будет -> ");

stream\_writer\_M.WriteLine(MaxElemOnceTime);

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

stream\_writer\_M.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

stream\_writer\_M.Close();

}

}catch (Exception e)

{

Console.WriteLine(e.Message);

Console.WriteLine(e.Data);

}

#endregion

#region#2

try

{

Console.WriteLine("Задача 2");

Console.WriteLine("Дана строка. Подсчитать, сколько раз среди данных символов встречается буква а.");

Console.WriteLine("Введи имя файла со строкой");

string file\_name\_str = Console.ReadLine();

int SymbolCount = GettingSymbolsACount(ReadingStrings(file\_name\_str+".txt"));

StreamWriter stream\_writer\_str = new StreamWriter(file\_name\_str + "\_answer.txt");

if (SymbolCount == 0)

{

Console.WriteLine("Символы отсутствуют");

stream\_writer\_str.WriteLine("Символы отсутствуют");

}

else

{

Console.Write("буква 'a' встречается в тексте: ");

Console.Write(SymbolCount);

Console.WriteLine(" раз");

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

stream\_writer\_str.Write("буква 'a' встречается в тексте: ");

stream\_writer\_str.Write(SymbolCount);

stream\_writer\_str.WriteLine(" раз");

stream\_writer\_str.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

stream\_writer\_str.Close();

}

}catch (Exception e)

{

Console.WriteLine(e.Message);

}

#endregion

#region #3

try

{

Console.WriteLine("Задача 3");

Console.WriteLine("Программа, которая считывает из текстового файла пять предложений и выводит их в обратном порядке.");

Console.WriteLine("Введите имя файла с текстом");

string file\_name\_txt = Console.ReadLine();

ReadTheText(file\_name\_txt+".txt");

}

#endregion

catch (Exception e)

{

Console.WriteLine(e.Message);

}

Console.ReadLine();

}

public static int[][] ReadTheMatrix(string namefile)

{

FileStream file = new FileStream(namefile, FileMode.Open);

StreamReader reader = new StreamReader(file);

List<string[]> matrix = new List<string[]>();

if (reader.EndOfStream == true)

{

throw (new Exception("Файл пуст. Пожалуйста, заполните файл матрицей с целочисленными элемментами"));

}

else

{

string readed\_str = "";

int M = 0;

int N = 0;

string[] space = new string[1];

space[0] = " ";

while (reader.EndOfStream == false)

{

readed\_str = reader.ReadLine();

matrix.Add(readed\_str.Split(space, StringSplitOptions.RemoveEmptyEntries));

for (int i = 0; i < matrix[N].Length; i++)

{

int res = 0;

if (int.TryParse(matrix[N][i], out res) == false)

{

throw (new Exception("Ошибка в чтении матрицы. Ошибка в чтении элемента"));

}

}

if (N != 0)

{

if (matrix[N].Length != matrix[N - 1].Length)

{

throw (new Exception("Ошибка в чтении матрицы. Строки в матрицы разной длины"));

}

}

N++;

}

M = matrix[0].Length;

if (N == 0 || M == 0)

{

throw (new Exception("Ошибка в чтении матрицы N=" + N + " M=" + M));

}

reader.Close();

int[][] final\_matrix = new int[N][];

for (int i = 0; i < matrix.Count; i++)

{

final\_matrix[i] = new int[M];

for (int j = 0; j < matrix[i].Length; j++)

{

final\_matrix[i][j] = int.Parse(matrix[i][j]);

}

}

return final\_matrix;

}

}

public static int MaxOnceElement(int[][] Matrix)

{

int max\_elem = 0;

SortedList<int, int> MatrixCount = new SortedList<int, int>();

//1

if (Matrix.Length == 0)

{

//2

throw (new Exception("Возникла ошибка чтения файла, матрица отсутсвует!"));

}

//3

else if (Matrix.Length == 1 && Matrix.Length == 1)

{

//4

if (Matrix[0][0] == 0)

{

//5

Console.WriteLine("В матрице один элемент ->");

return max\_elem = Matrix[0][0];

}

}

//6

else

{

max\_elem = Matrix[0][0];

//7

for (int i = 0; i < Matrix.Length; i++)

{

//8

for (int j = 0; j < Matrix[i].Length; j++)

{

//9

if ((max\_elem < Matrix[i][j]) && (max\_elem != Matrix[i][j]))

//10

{ max\_elem = Matrix[i][j]; }

}

}

}

//11

return max\_elem;

}

public static string ReadingStrings(string namefile)

{

FileStream file = new FileStream(namefile, FileMode.Open);

StreamReader reader = new StreamReader(file);

if (reader.EndOfStream == true)

{

throw (new Exception("Файл пуст. Пожалуйста, заполните файл строкой символов с хотя-бы одной буквой 'а'"));

}

else

{

string String = reader.ReadToEnd();

reader.Close();

return String;

}

}

public static int GettingSymbolsACount(string AllSymbols)

{

int FoundTheSymbolA = 0;

int i = 0;

while (i != AllSymbols.Length)

{

if ((AllSymbols[i] == 'a') || (AllSymbols[i] == 'а') || (AllSymbols[i] == 'А') || (AllSymbols[i] == 'A'))

{

FoundTheSymbolA++;

}

i++;

}

return FoundTheSymbolA;

}

public static void ReadTheText(string namefile)

{

FileStream read\_text\_file = new FileStream(namefile, FileMode.Open);

StreamReader reader = new StreamReader(read\_text\_file);

StreamWriter stream\_writer\_txt = new StreamWriter("\_answer"+namefile);

if (reader.EndOfStream == true)

{

throw (new Exception("Файл пуст. Пожалуйста, заполните файл текстом с предложениями(-ем)"));

stream\_writer\_txt.Write("Файл пуст. Пожалуйста, заполните файл текстом с предложениями(-ем)");

}

else

{

string Text = reader.ReadToEnd();

int i = Text.Length-1;

string reversed\_text = "";

while (i !=-1)

{

reversed\_text += Text[i];

i--;

}

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

Console.WriteLine(Text);

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

Console.WriteLine(reversed\_text);

stream\_writer\_txt.WriteLine(reversed\_text);

stream\_writer\_txt.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

stream\_writer\_txt.Close();

}

}

}

}

ПРИМЕРЫ ТЕСТОВЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

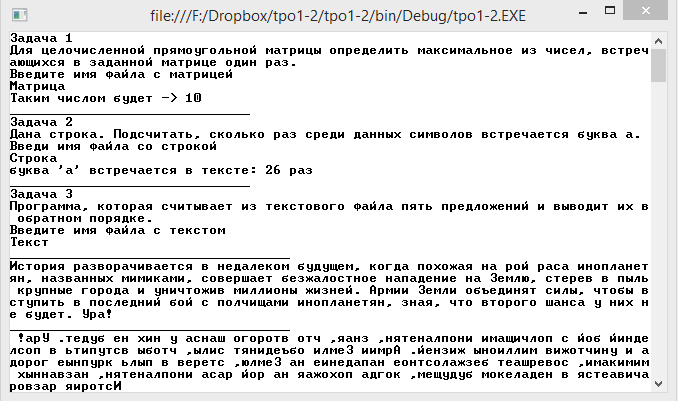


Рисунок 1 – Правильное выполнение программы

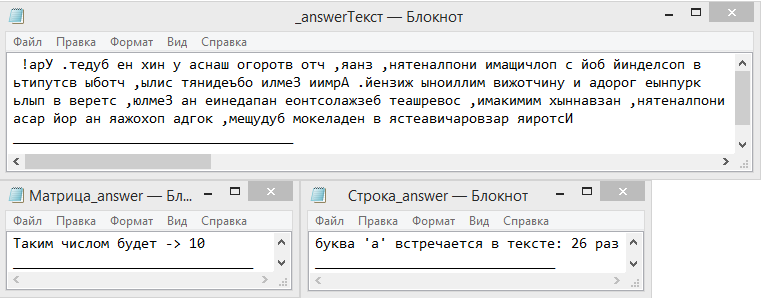


Рисунок 2 – Файлы с матрицей, строкой и текстом заполненные ответом

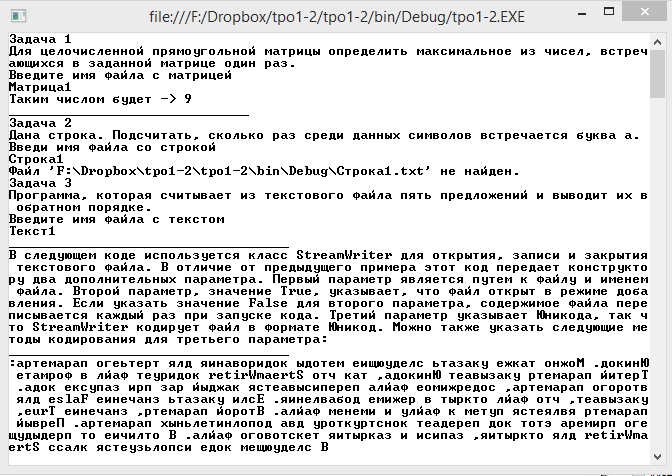


Рисунок 2 – Файлы с матрицей и текстом заполнены совершенным образом. Файл «Строка1» не существует

* 1. Этап 2 – Модульное тестирование

Вариант задания

1. Выбрать в качестве тестируемого один из классов, спроектированных в лабораторных работах №№ 1 – 4.
2. Составить спецификацию тестового случая для одного из методов выбранного класса.
3. Реализовать тестируемый класс и необходимое тестовое окружение на языке С#.
4. Выполнить тестирование с выводом результатов на экран и сохранением в *log*-файл.
5. Проанализировать результаты тестирования, сделать выводы.

Выбранный тестируемый модуль AccountBase( ) и функция регистрации нового пользователя (Registration).

//------------------------------------------------------------------------------

// <auto-generated>

// Этот код создан инструментальным средством

// В случае повторного создания кода изменения, внесенные в этот файл, будут потеряны.

// </auto-generated>

//------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

public class AccountBase

{

private List<List<string>> accounts;

public virtual string Login(string login, string password)

{

for (int i = 0; i < accounts.Count; i++)

{

if (login == accounts[i][0] && password == accounts[i][1])

{

return accounts[i][2];

}

}

return "";

}

public AccountBase()

{

accounts = new List<List<string>>();

accounts.Add(new List<string>());

accounts[0].Add("user");

accounts[0].Add("123456789");

accounts[0].Add("Policyholder");

accounts.Add(new List<string>());

accounts[1].Add("admin");

accounts[1].Add("admin");

accounts[1].Add("Insurer");

accounts.Add(new List<string>());

accounts[2].Add("bank");

accounts[2].Add("54321");

accounts[2].Add("BankEmployee");

}

public virtual string GegErrorMassage(string login, string password, string role)

{

string line = "";

//Login

if (string.IsNullOrWhiteSpace(login) == true)

{

line+="Логин не может быть пустой или состоять из пробельных симфолов.\n";

}

if (login.Length < 4)

{

line += "Логин должен состоять не менее чем из 4х символом.\n";

}

for (int i = 0; i < login.Length; i++)

{

if (char.IsUpper(login[i]) == true)

{

line+="Логин должен состоять из строчных букв.\n";

break;

}

}

for (int i = 0; i < login.Length; i++)

{

if ((login[i]>='А')&&((login[i])<='ё'))

{

line += "Логин не должнен состоять из символов кирилицы.\n";

break;

}

}

for (int i = 0; i < login.Length; i++)

{

if(char.IsSeparator(login[i])==true)

{

line += "Логин не может содержать раделительные знаки.\n";

}

}

//Password

if (string.IsNullOrWhiteSpace(password) == true)

{

line += "Пароль не может быть пустой или состоять из пробельных симфолов.\n";

}

if (password.Length < 9)

line += "Пароль должен состоять не менее чем из 8х символом.\n";

for (int i = 0; i < password.Length; i++)

{

if (char.IsSeparator(password[i]) == true)

{

line += "Пароль не может содержать раделительные знаки.\n";

}

}

bool

Upper = false,

Lower = false,

Numeric = false;

for (int i = 0; i < password.Length; i++)

{

if (char.IsUpper(password[i]) == true)

{

Upper = true;

}

if (char.IsLower(password[i]) == true)

{

Lower = true;

}

if (char.IsNumber(password[i])==true)

{

Numeric = true;

}

}

if (Upper==false)

{

line += "Пароль должен содержать символы верхнего регистра.\n";

}

if(Lower==false)

{

line += "Пароль должен содержать символы нижнего регистра.\n";

}

if(Numeric==false)

{

line += "Пароль должен содержать символы из категории цифр.\n";

}

//Role

switch (role)

{

case "Policyholder":

break;

case "Insurer":

break;

case "BankEmployee":

break;

default:

line += "Не может быть использован такой профиль. Профиль может выбран только из Policyholder/Insurer/BankEmployee.\n";

break;

}

if (line == "")

{

line += "Учетная запись успешно сформирована!";

}

return line;

}

public virtual bool Registration(string login,string password,string role)

{

//Login

if (string.IsNullOrWhiteSpace(login) == true)

{

return false;

}

if (login.Length < 5) return false;

for(int i=0;i<login.Length;i++)

{

if(char.IsUpper(login[i])==true)

{

return false;

}

if((login[i]>='А')&&((login[i])<='ё'))

{

return false;

}

if(char.IsSeparator(login[i])==true)

{

return false;

}

}

//Password

if (string.IsNullOrWhiteSpace(password) == true)

{

return false;

}

if (password.Length < 9)

{

return false;

}

bool

Upper = false,

Lower = false,

Numeric = false;

for (int i = 0; i < password.Length; i++)

{

if(char.IsSeparator(password[i])==true)

{

return false;

}

if (char.IsUpper(password[i]) == true)

{

Upper = true;

}

if (char.IsLower(password[i]) == true)

{

Lower = true;

}

if (char.IsNumber(password[i])==true)

{

Numeric = true;

}

}

if (Upper==false||Lower==false||Numeric==false)

{

return false;

}

//Role

switch(role)

{

case "Policyholder":

break;

case "Insurer":

break;

case "BankEmployee":

break;

default:

return false;

}

//

accounts.Add(new List<string>());

accounts[accounts.Count - 1].Add(login);

accounts[accounts.Count - 1].Add(password);

accounts[accounts.Count - 1].Add(role);

return true;

}

public virtual void Exit()

{

return;

}

public virtual void Select\_profile()

{

throw new System.NotImplementedException();

}

}

СПЕЦИФИКАЦИЯ ТЕСТОВОГО СЛУЧАЯ.

Спецификация тестового случая должна включать следующие пункты:

1. Название тестируемого класса: AccountBase.
2. Название тестового случая: AccountBaseTest().
3. Тест проверяет правильность работы метода ***public virtual bool Registration(string login,string password,string role)*** – изменение публичных атрибутов login, password и role. В тесте подаются значения из разных интервалов:

Таблица 1 – Тестовые последовательности

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Параметр | 1-е значение | 2-е значение | 3-е значение | 4-е значение | 5-е значение | 6-е значение | 7-е значение | 8-е значение | 9-е значение |
| Логин | user | admin | bank | Haker | Вася | Лола88 | user51 |  | Alex.xelA |
| Пароль | sheloB21 | курилко | BankS55A | 456515844 |  | pAsword11 | Hope2 | dISpйОch48 | asd../\*\* |
| Роль | Policyholder | Insurer | BankEmployee | BankEmployee | Policyholder | Калич | Admin | User |  |

ТЕКСТ ПРОГРАММЫ

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace AccountBaseTest

{

public class Log

{

//Создание лог файла

static private StreamWriter log = new StreamWriter("log.log");

static public void Add(string msg) //Добавление сообщения в лог файл

{

log.WriteLine(msg);

}

static public void Close() //Закрыть лог файл

{

log.Close();

}

}

abstract class Tester

{

protected void LogMessage(string s)

//Добавление сообщения в лог-файл

{

Log.Add(s);

}

}

class TCommandTester : Tester // Тестовый драйвер

{

AccountBase OUT;

public TCommandTester()

{

OUT = new AccountBase();

Run();

}

private void Run()

{

LoginTest();

}

private void LoginTest()

{

Random random = new Random();

string[] login = { "user" ,"admin", "bank" ,"Haker" ,"Вася" , "Лола88" ,"user51", " " ,"Alex.xelA" };

string[] password = { "sheloB21","курилко","BankS55A","456515844"," ", "pAsword11","Hope2" , "dISpйОch48","asd../\*\*"};

string[] role = { "Policyholder", "Insurer", "BankEmployee", "BankEmployee", "Policyholder", "Калич", "Admin", "User", " " };

for (int i = 0; i < 9; i++)

{

LogMessage("Логин:" + login[i] + " Пароль:" + password[i] + " Профиль:" + role[i] + OUT.GegErrorMassage(login[i], password[i], role[i]) + OUT.ToString());

}

}

static void Main()

{

TCommandTester CommandTester = new TCommandTester();

Log.Close();

}

}

}

ТЕСТОВЫЕ ПРИМЕРЫ

На рисунке 1 представлено содержимое файла log.log после выполнения программы для тестовой последовательности, описанной в таблице 1.

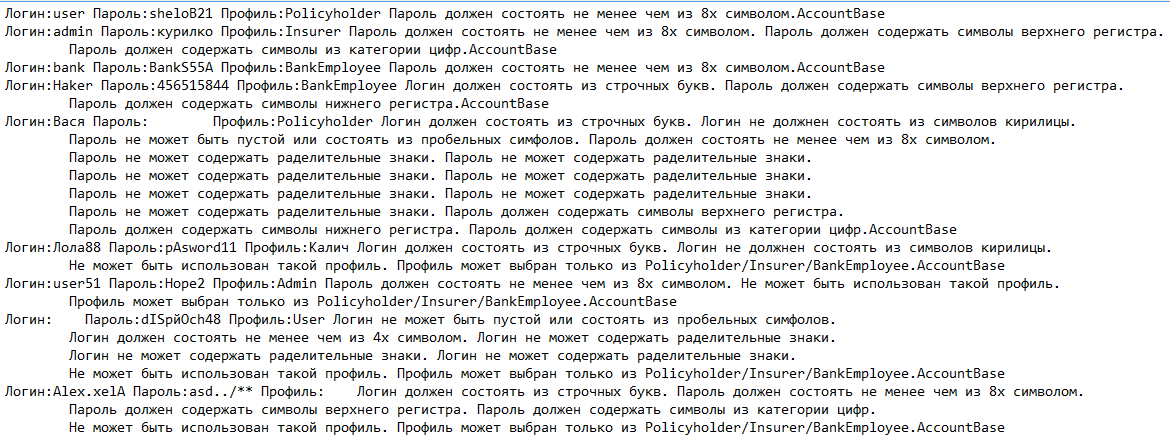


Рисунок 3 – Файл log.log после выполнения программы

* 1. Этап 3 – Интеграционное тестирование

Задача

1. Выбрать в качестве тестируемого взаимодействие двух или более классов, спроектированных в лабораторных работах №№1 – 4.
2. Составить спецификацию тестового случая.
3. Реализовать тестируемые классы и необходимое тестовое окружение на языке С#.
4. Выполнить тестирование с выводом результатов на экран и сохранением в *log*-файл.
5. Проанализировать результаты тестирования, сделать выводы.

СПЕЦИФИКАЦИЯ ТЕСТОВОГО СЛУЧАЯ.

Спецификация тестового случая должна включать следующие пункты:

1) **Названия взаимодействующих классов**: Policyholder, Bankaccount.

2) **Название теста**: ScheduleTaskTest.

3) **Описание теста**: тест проверяет возможность создания объекта типа Task и добавления его в список при вызове метода AddTaskToListTask().

4) **Начальные условия**: список задач пуст.

5) **Ожидаемый результат**: сначала в список будет добавлены 3 задачи, а затем ещё две.

ТЕКСТ ПРОГРАММЫ

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System;

using System.IO;

namespace TestTwoClass

{

class Program

{

public class Log

{

//Создание лог файла

static private StreamWriter log = new StreamWriter("log.log");

static public void Add(string msg) //Добавление сообщения в лог файл

{

log.WriteLine(msg);

}

static public void Close() //Закрыть лог файл

{

log.Close();

}

}

abstract class Tester

{

protected void LogMessage(string s)

//Добавление сообщения в лог-файл

{

Log.Add(s);

}

}

class PaymantTester : Tester // Тестовый драйвер

{

Policyholder OUTMain;

public PaymantTester()

{

Init();

}

public void Init()

{

OUTMain = new Policyholder();

}

private void Run()

{

Policyholder\_BankAccountTest();

}

private void Policyholder\_BankAccountTest()

{

string[] pay={"sdgsdg","0","12234","12.50","124,32"};

for (int i = 0; i < 5; i++)

{

LogMessage("Выполнение финансовой операции");

LogMessage("Оплата: "+pay[i]+" Результат: "+OUTMain.Financial\_transactions(pay[i]));

}

}

static void Main()

{

PaymantTester profileTester = new PaymantTester();

profileTester.Run();

Log.Close();

}

}

}

}

ТЕСТОВЫЕ ПРИМЕРЫ

На рисунке 4 представлено содержимое файла log.log после выполнения программы.

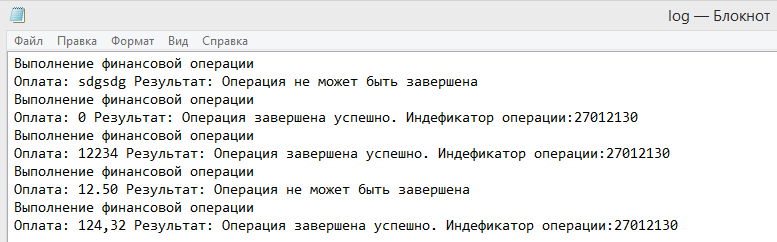


Рисунок 4 – Файл log.log после выполнения программы

* 1. Этап 4 – Модельное тестирование

Задача

1. Реализовать на языке C# один из классов, спроектированных в лабораторной работе № 1. Методы класса при этом не реализовывать.
2. Разработать для созданного класса набор модульных тестов, включающий тесты для каждого метода.
3. Запустить набор тестов, проанализировать и сохранить результаты.
4. Поочередно реализовать методы класса, выполняя тестирование при каждом изменении программного кода.
5. После того, как весь набор тестов будет выполняться успешно, реализацию классов можно считать завершенной.
6. СПЕЦИФИКАЦИЯ ТЕСТОВОГО СЛУЧАЯ.

Спецификация тестового случая должна включать следующие пункты:

1. Название тестируемого класса: AccountBase.
2. Название тестовых случаев: TestRegistration(), TestRegError1(), TestLogin(), TestRegError2(), TestException();
3. Тест проверяет правильность работы методов:

* Registration (str\_login, str\_password, str\_profile)- изменение атрибутов login, password,profile;
* GetErrMessage (str\_login, str\_password, str\_profile) – возвращения текста ошибки для ввода того или иного значения для атрибутов login, password, profile;

В тесте подаются значения (Таблица 2):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Тест | Логин | Пароль | Профиль | Результат | Ошибка |
| TestRegistration() | Padme | Wakaa5564 | Policyholder | false |  |
| TestRegError1() | Adam | Petra88zf | Policyholder | false | Логин должен состоять из строчных букв.\n |
| TestLogin() | admin | admin | ожид/Insurer | true |  |
| TestRegError2() | admin1 | 1admin1 | Insurer | false | Пароль должен состоять не менее чем из 8х символом.\nПароль должен содержать символы верхнего регистра.\n |
| Тест | Тип ожидаемого исключения | | | Результат | Ошибки |
| TestException() | NotImplementedException | | | true |  |

ТЕСТИРУЕМЫЙ КЛАСС

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

public class AccountBase

{

private List<List<string>> accounts;

public virtual string Login(string login, string password)

{

for (int i = 0; i < accounts.Count; i++)

{

if (login == accounts[i][0] && password == accounts[i][1])

{

return accounts[i][2];

}

}

return "";

}

public AccountBase()

{

accounts = new List<List<string>>();

accounts.Add(new List<string>());

accounts[0].Add("user");

accounts[0].Add("123456789");

accounts[0].Add("Policyholder");

accounts.Add(new List<string>());

accounts[1].Add("admin");

accounts[1].Add("admin");

accounts[1].Add("Insurer");

accounts.Add(new List<string>());

accounts[2].Add("bank");

accounts[2].Add("54321");

accounts[2].Add("BankEmployee");

}

public virtual string GetErrorMassage(string login, string password, string role)

{

string line = "";

//Login

if (string.IsNullOrWhiteSpace(login) == true)

{

line+="Логин не может быть пустой или состоять из пробельных симфолов.\n";

}

if (login.Length < 4)

{

line += "Логин должен состоять не менее чем из 4х символом.\n";

}

for (int i = 0; i < login.Length; i++)

{

if (char.IsUpper(login[i]) == true)

{

line+="Логин должен состоять из строчных букв.\n";

break;

}

}

for (int i = 0; i < login.Length; i++)

{

if ((login[i]>='А')&&((login[i])<='ё'))

{

line += "Логин не должнен состоять из символов кирилицы.\n";

break;

}

}

for (int i = 0; i < login.Length; i++)

{

if(char.IsSeparator(login[i])==true)

{

line += "Логин не может содержать раделительные знаки.\n";

}

}

//Password

if (string.IsNullOrWhiteSpace(password) == true)

{

line += "Пароль не может быть пустой или состоять из пробельных симфолов.\n";

}

if (password.Length < 9)

line += "Пароль должен состоять не менее чем из 8х символом.\n";

for (int i = 0; i < password.Length; i++)

{

if (char.IsSeparator(password[i]) == true)

{

line += "Пароль не может содержать раделительные знаки.\n";

}

}

bool

Upper = false,

Lower = false,

Numeric = false;

for (int i = 0; i < password.Length; i++)

{

if (char.IsUpper(password[i]) == true)

{

Upper = true;

}

if (char.IsLower(password[i]) == true)

{

Lower = true;

}

if (char.IsNumber(password[i])==true)

{

Numeric = true;

}

}

if (Upper==false)

{

line += "Пароль должен содержать символы верхнего регистра.\n";

}

if(Lower==false)

{

line += "Пароль должен содержать символы нижнего регистра.\n";

}

if(Numeric==false)

{

line += "Пароль должен содержать символы из категории цифр.\n";

}

//Role

switch (role)

{

case "Policyholder":

break;

case "Insurer":

break;

case "BankEmployee":

break;

default:

line += "Не может быть использован такой профиль. Профиль может выбран только из Policyholder/Insurer/BankEmployee.\n";

break;

}

if (line == "")

{

line += "Учетная запись успешно сформирована!";

}

return line;

}

public virtual bool Registration(string login,string password,string role)

{

//Login

if (string.IsNullOrWhiteSpace(login) == true)

{

return false;

}

if (login.Length < 5) return false;

for(int i=0;i<login.Length;i++)

{

if(char.IsUpper(login[i])==true)

{

return false;

}

if((login[i]>='А')&&((login[i])<='ё'))

{

return false;

}

if(char.IsSeparator(login[i])==true)

{

return false;

}

}

//Password

if (string.IsNullOrWhiteSpace(password) == true)

{

return false;

}

if (password.Length < 9)

{

return false;

}

bool

Upper = false,

Lower = false,

Numeric = false;

for (int i = 0; i < password.Length; i++)

{

if(char.IsSeparator(password[i])==true)

{

return false;

}

if (char.IsUpper(password[i]) == true)

{

Upper = true;

}

if (char.IsLower(password[i]) == true)

{

Lower = true;

}

if (char.IsNumber(password[i])==true)

{

Numeric = true;

}

}

if (Upper==false||Lower==false||Numeric==false)

{

return false;

}

//Role

switch(role)

{

case "Policyholder":

break;

case "Insurer":

break;

case "BankEmployee":

break;

default:

return false;

}

//

accounts.Add(new List<string>());

accounts[accounts.Count - 1].Add(login);

accounts[accounts.Count - 1].Add(password);

accounts[accounts.Count - 1].Add(role);

return true;

}

public virtual void Exit()

{

return;

}

public virtual void Select\_profile()

{

throw new System.NotImplementedException();

}

}

ТЕКСТ ПРОГРАММЫ

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Diagnostics;

namespace UnitTestAccBase

{

[TestClass]

public class UnitTest1

{

[TestInitialize]

public void TestInitialize()

{

Trace.WriteLine("Инициализация теста началася в " + DateTime.Now);

}

[ClassInitialize]

public static void ClassInitialize(TestContext testContext)

{

Trace.WriteLine("Инициализация тестового класса началася в " + DateTime.Now);

}

#region TestMethod

[TestMethod]

public void TestRegistration()

{

AccountBase accBase = new AccountBase();

Assert.IsFalse(accBase.Registration("Padme", "Wakaa5564", "Policyholder"));

}

[TestMethod]

public void TestRegError1()

{

AccountBase accBase = new AccountBase();

string actualValuaError = accBase.GetErrorMassage("Adam", "Petra88zf", "Policyholder");

string expectedValuaError = "Логин должен состоять из строчных букв.\n";

Assert.AreEqual(expectedValuaError, actualValuaError);

}

[TestMethod]

public void TestLogin()

{

AccountBase accBase = new AccountBase();

string actual\_value = accBase.Login("admin", "admin");

string excpected\_value = "Insurer";

Assert.AreEqual(excpected\_value,actual\_value);

}

[TestMethod]

public void TestRegError2()

{

AccountBase accBase = new AccountBase();

string actual\_value = accBase.GetErrorMassage("admin1", "1admin1", "Insurer");

string excpected\_value = "Пароль должен состоять не менее чем из 8х символом.\nПароль должен содержать символы верхнего регистра.\n";

Assert.AreEqual(excpected\_value, actual\_value);

}

[TestMethod]

[ExpectedException(typeof(NotImplementedException))]

public void TestException()

{

AccountBase accBase = new AccountBase();

accBase.Select\_profile();

}

#endregion

[TestCleanup]

public void TestCleanup()

{

Trace.WriteLine("Сборка начата в " + DateTime.Now);

}

[ClassCleanup]

public static void ClassCleanup()

{

Trace.WriteLine("Тест завершон в " + DateTime.Now);

}

}

}

ТЕСТОВЫЕ ПРИМЕРЫ

На рисунке 1 представлено содержимое обозревателя тестов после построения.

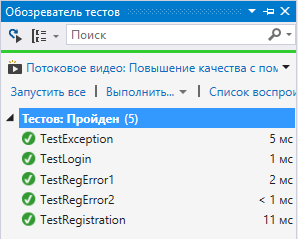


Рисунок 5 – Обозреватель тестов после построения

На рисунках 5-10 изображены результаты трассировки отладки.

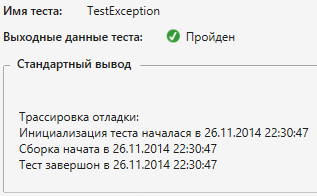


Рисунок 6 – Трассировка отладки для TestException()

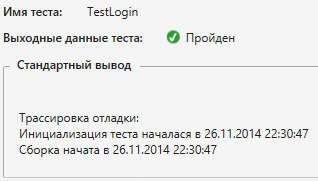


Рисунок 7 – Трассировка отладки для TestLogin()

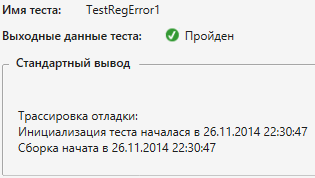


Рисунок 8 – Трассировка отладки для TestRegError1()

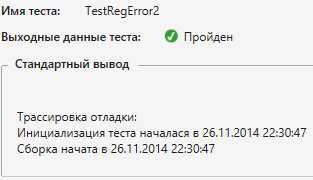


Рисунок 9– Трассировка отладки для TestError2()

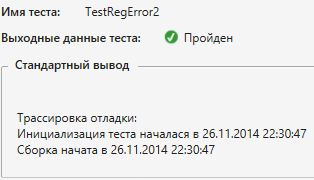


Рисунок 10 – Трассировка отладки для TestRegistration()

3.6. Этап 6 – Профилирование

Задача

Разработать программу на основе библиотеки классов, реализованной и протестированной в предыдущей работе. Программа должна как можно более полно использовать функциональность класса. При необходимости для наглядности профилирования в методы класса следует искусственно внести задержку выполнения.

Выполнить профилирование разработанной программы, выявить функции, на выполнение которых тратится наибольшее время.

Модифицировать программу с целью оптимизации времени выполнения.

Выполнить повторное профилирование программы, сравнить новые результаты и полученные ранее, сделать выводы.

Теоретические сведенья

Методы профилирования

**Sampling**

Sampling — собирает статистические данные о работе приложения (во время профилирования). Этот метод легковесный и поэтому, в результате его работы очень маленькая погрешность в полученных данных.

Каждый определенный интервал времени собирается информация о стеке вызовов (call stack). На основе этих данные производится подсчет производительности. Используется для первоначального профилирования и для определения проблем связанных с использование процессора.

**Instrumentation**

Instrumentation — собирает детализированную информацию о времени работы каждой вызванной функции. Используется для замера производительности операций ввода/вывода.

Метод внедряет свой код в двоичный файл, который фиксирует информацию о синхронизации (времени) для каждой функции в файл, и для каждой функции которые вызываются в этой.

Отчет содержит 4 значения для предоставления затраченного времени:

Elapsed Inclusive — общее время, затраченное на выполнение функции

Application Inclusive — время, затраченное на выполнение функции, за исключением времени обращений к операционной системе.

Elapsed Exclusive — время, затраченное на выполнение кода в теле. Время, которое тратят функции, вызванные целевой функцией.

Application Exclusive — время, затраченное на выполнение кода в теле. Исключается время, которое тратится выполнения вызовов операционной системы и время, затраченное на выполнение функций, вызванные целевой функцией.

**Concurrency**

Concurrency – собирает информацию о многопоточных приложения (как отлаживать многопоточные приложения см. «Руководство по отладке многопоточных приложений в Visual Studio 2010»). Метод собирает подробную информацию о стеке вызовов, каждый раз, когда конкурирующие потоки вынуждены ждать доступа к ресурсу.

**.NET Memory**

.NET Memory — профайлер собирает информацию о типе, размере, а также количество объектов, которые были созданы в распределении или были уничтожены сборщиком мусора. Профилирование памяти почти не влияет на производительность приложения в целом.

**Tier Interaction**

Tier Interaction – добавляет информацию в файл для профилирования о синхронных вызовах ADO.NET между страницей ASP.NET или другими приложениями и SQL сервера. Данные включают число и время вызовов, а также максимальное и минимальное время.

Текст первоначального варианта программы

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace InsuranceServiceConsole

{

class Program

{

static void Main(string[] args)

{

do

{

Console.WriteLine(">Войти -1<");

Console.WriteLine(">Регистрация-2<");

Console.WriteLine(">Выход -3<");

try

{

switch (Console.ReadLine())

{

case "1":

Console.Clear();

Entry();

break;

case "2":

Console.Clear();

Registration();

break;

case "3":

return;

default:

Console.Clear();

break;

}

}

catch (Exception e)

{

Console.WriteLine(e.Message.ToString());

Console.ReadKey();

Console.Clear();

}

} while (true);

}

/// <summary>

/// Регистрация

/// </summary>

public static void Registration()

{

AccountBase regAccountBase = new AccountBase();

do

{

Console.Clear();

Console.WriteLine("Созадание учетной записи");

Console.WriteLine("для выхода введите в поле 0");

Console.WriteLine();

Console.WriteLine("Введите логин >> ");

string login = Console.ReadLine();

if (login == "0")

{

Console.Clear();

return;

}

Console.WriteLine("Введите пароль >> ");

string password = Console.ReadLine();

if (password == "0")

{

Console.Clear();

return;

}

Console.WriteLine("Введите свой профиль >> ");

string role = Console.ReadLine();

if (role == "0")

{

Console.Clear();

return;

}

if (regAccountBase.Registration(login, password, role) == false)

{

Console.Clear();

Console.WriteLine("!Регистрация не возможна!");

Console.WriteLine(regAccountBase.GetErrorMassage(login, password, role));

Console.ReadKey();

}

} while (true);

}

/// <summary>

/// Вход в профиль

/// </summary>

public static void Entry()

{

AccountBase accountBase = new AccountBase();

do

{

try

{

Console.WriteLine("Введите логин и пароль(0-Выход)");

string login = Console.ReadLine();

string password = Console.ReadLine();

if (login == "0" || password == "0")

{

Console.Clear();

break;

}

if (accountBase.Login(login, password) != "")

{

Console.Clear();

Console.WriteLine("Вход...");

Initiate\_pprofile(accountBase.Login(login, password));

}

else

{

Console.WriteLine("не верный логин или пароль");

Console.ReadKey();

Console.Clear();

}

}

catch (Exception e)

{

Console.WriteLine(e.Message.ToString());

Console.ReadKey();

Console.Clear();

}

} while (true);

}

/// <summary>

/// выбор программного профиля

/// </summary>

/// <param name="profile"></param>

public static void Initiate\_pprofile(string profile)

{

if (profile == "Policyholder")

{

Console.WriteLine("Вы зашли как Policyholder");

Console.ReadKey();

Console.Clear();

Policyholder policyholder = new Policyholder();

Wk\_Policyholder(policyholder);

}

if (profile == "Insurer")

{

Console.WriteLine("Вы зашли как Insurer");

Console.ReadKey();

Console.Clear();

Insurer insurer = new Insurer();

Wk\_Insurer(insurer);

}

if (profile == "BankEmployee")

{

Console.WriteLine("Вы зашли как BankEmployee");

Console.ReadKey();

Console.Clear();

BankEmployee bankEmployee = new BankEmployee();

Wk\_BankEmployee(bankEmployee);

}

}

/// <summary>

/// Работа под профилем Страхователя

/// </summary>

/// <param name="policyholder"></param>

public static void Wk\_Policyholder(Policyholder policyholder)

{

do

{

try

{

Console.WriteLine(" МЕНЮ ПОЛЬЗОВАТЕЛЯ");

Console.WriteLine("> Просмотр личных данных-1 <");

Console.WriteLine("> Оформить новый страховой догор-2 <");

Console.WriteLine("> Финансовые операции-3 <");

Console.WriteLine("> Сменить профиль-4 <");

switch (Console.ReadLine())

{

case "1":

policyholder.View\_my\_info();

break;

case "2":

if (policyholder.Request\_add\_new\_contract() == true)

{

Console.Clear();

Console.WriteLine("Новый контракт создан...");

Console.WriteLine("!Нет подключенных баз данных!");

Console.ReadKey();

Console.Clear();

}

break;

case "3":

Console.Clear();

Console.WriteLine("Введи сумму перевода");

string pay = Console.ReadLine();

policyholder.Financial\_transactions(pay);

Console.Clear();

Console.WriteLine("Предположим, что вы перевили на счет банка пару шейкелей.");

Console.WriteLine("Теперь вы можете вернутся в главное меню..");

Console.ReadKey();

Console.Clear();

break;

case "4":

Console.Clear();

Console.WriteLine("Выход из профиля...");

Console.ReadKey();

Console.Clear();

return;

break;

default:

Console.Clear();

break;

}

}

catch (Exception e)

{

Console.WriteLine(e.Message.ToString());

Console.ReadKey();

Console.Clear();

}

} while (true);

}

/// <summary>

/// Работа под профилем Страховщик

/// </summary>

/// <param name="insurer"></param>

public static void Wk\_Insurer(Insurer insurer)

{

}

/// <summary>

/// Работа под профилем Банковского работника

/// </summary>

/// <param name="bankEmployee"></param>

public static void Wk\_BankEmployee(BankEmployee bankEmployee)

{

}

}

}

Результаты профилирования первоначального варианта программы с подробным анализом узких мест

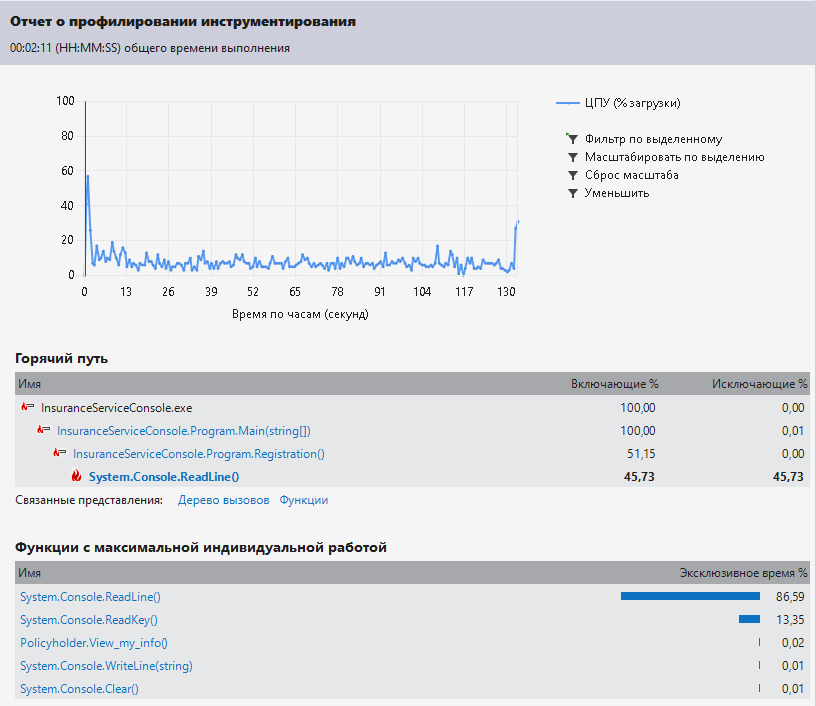


Рисунок 12 – Summary по методу Instrumentation

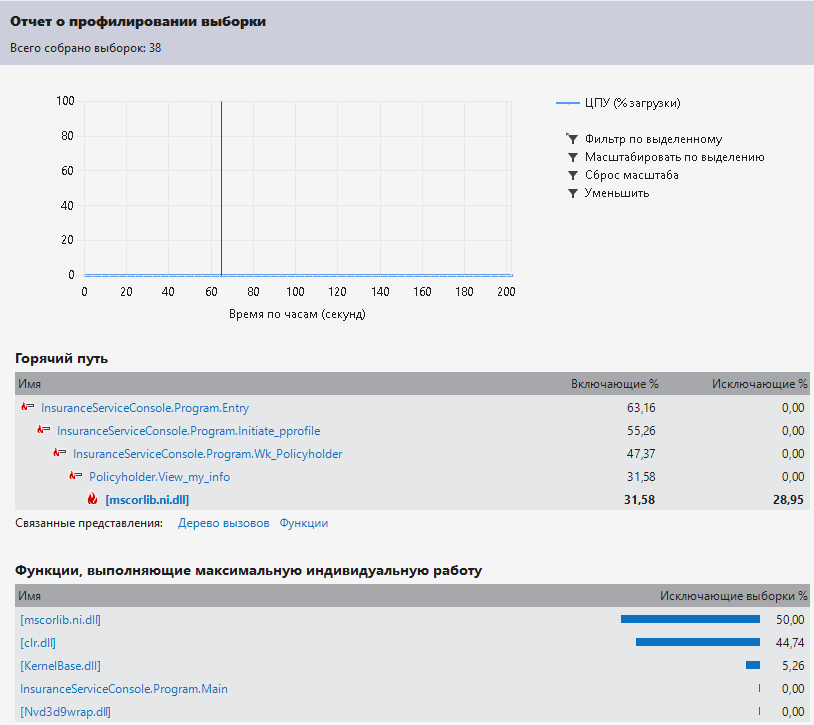


Рисунок 13 – Summary по методу Sampling

В Summary отображается график использования процессора в течение всего времени профилирования. Список Горячий Путь (Hot Path) показывает ветки вызовов, которые проявили наибольшую активность. А в списке Функции с маскимальной индивидуальной работой (Functions Doing Most Individual Work) (название которого говорит само за себя) – функции, которые занимали большее время процесса в теле этих функций.

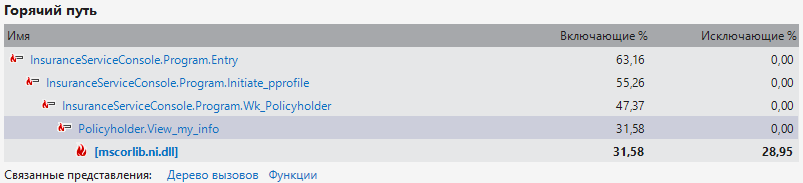


Рисунок 14 – Hot Path

Посмотрев на список Hot Path видим, что метод Policyholder.View\_my\_info занимает почти последнее место в ветке вызовов. Его то и можно изучить внимательнее на предмет улучшения производительности. Нажимаем на Policyholder.View\_my\_info и перед нами открывается Function Details.

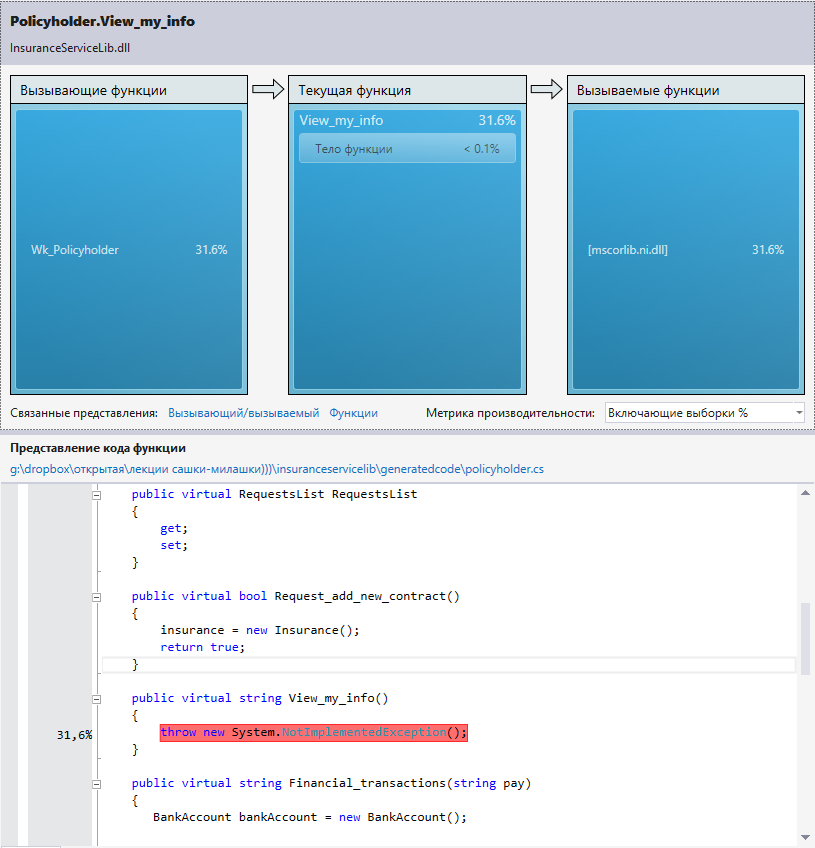


Рисунок 15 – Function Details для Policyholder.View\_my\_info

Это окно содержит две части. Окно расходов предусматривает графическое представление работы функций, и вклад функции и вызывающих ее на количество экземпляров, которые были отобраны. Можно изменить рассматриваемую функцию, нажав на нее мышкой.

Function Code View показывает код метода, когда он доступен и подсвечивает наиболее «дорогие» строки в выбранном методе.

Как бы это не было странно, но самой жадной функцией оказалась заглушка. Тогда попробуем взять другую функция из Hot Path



Рисунок 16 – Hot Path для InsuranceServiceConsole.Program.Initiate\_pprofile

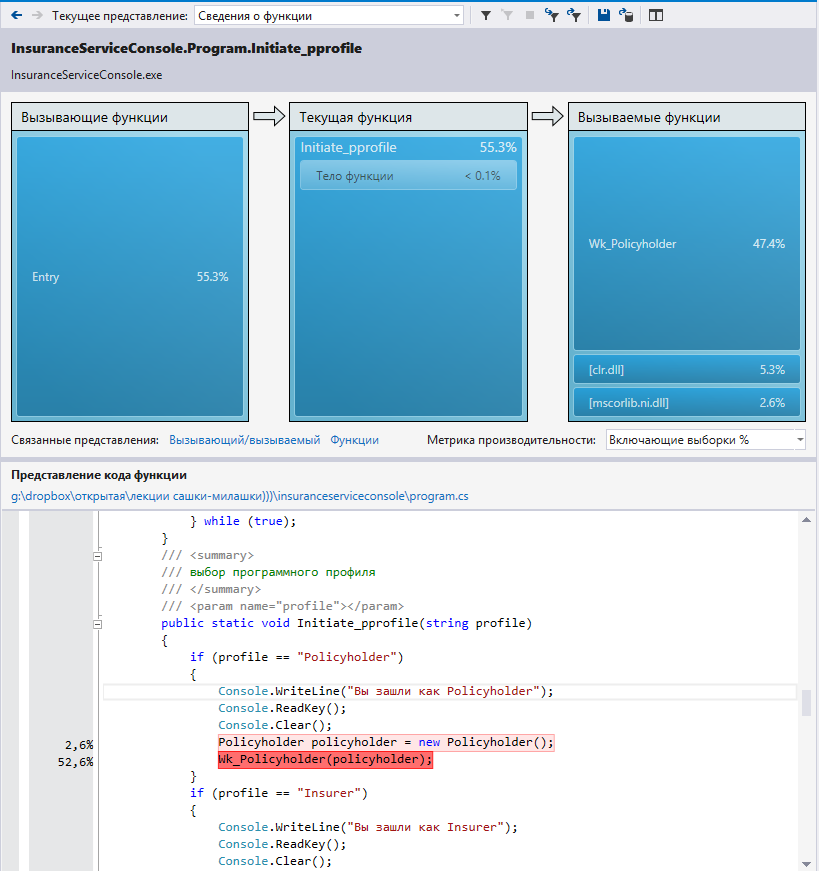


Рисунок 17 – Function Details для InsuranceServiceConsole.Program.Initiate\_pprofile

Текст модифицированного варианта программы со словесным описанием и обоснованием внесенных изменений

Следует создать альтернативную версию InsuranceServiceConsole.Program.Initiate\_pprofile.

public static void Initiate\_pprofile(string profile)

{

if (profile == "Policyholder")

{

Console.WriteLine("Вы зашли как Policyholder");

Console.ReadKey();

Console.Clear();

Policyholder policyholder = new Policyholder();

Wk\_Policyholder(policyholder);

#region Альтернатива на ухудшение

List<List<List<double>>> mas = new List<List<List<double>>>();

for (int i = 0; i < 100; i++)

{

mas.Add(new List<List<double>>());

for (int j = 0; j < 100; j++)

{

mas[i].Add(new List<double>());

for (int k = 0; k < 100; k++)

{

mas[i][j].Add(i \* j \* k);

}

}

}

#endregion

}

Результаты профилирования модифицированного варианта программы

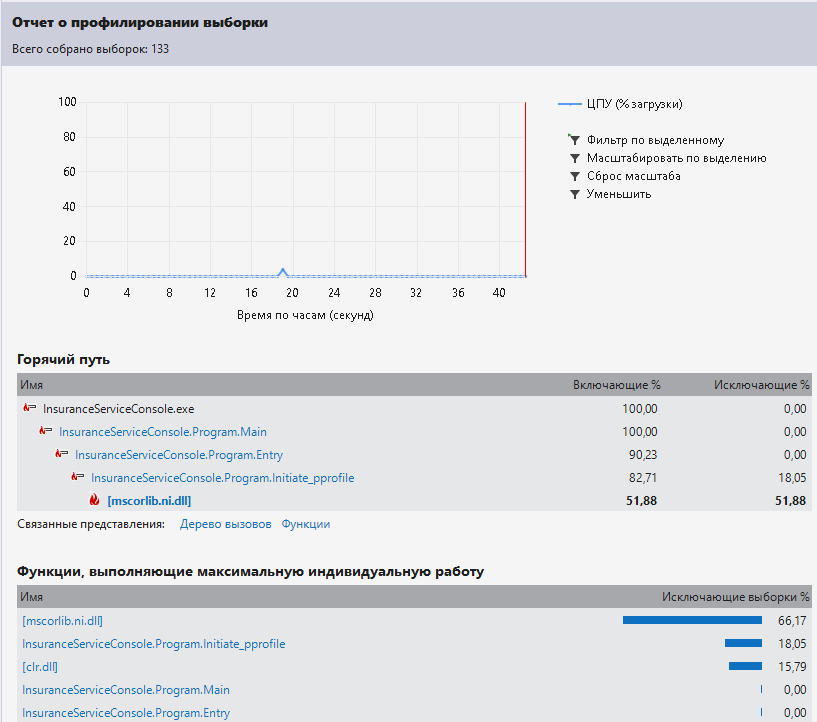


Рисунок 16 – Sumary для модифицированного метода InsuranceServiceConsole.Program.Initiate\_pprofile

Явно видно воздействия анти-оптимизация на нагрузку ЦПУ. Включающий процент возрос с 31.58% до 80.71%.

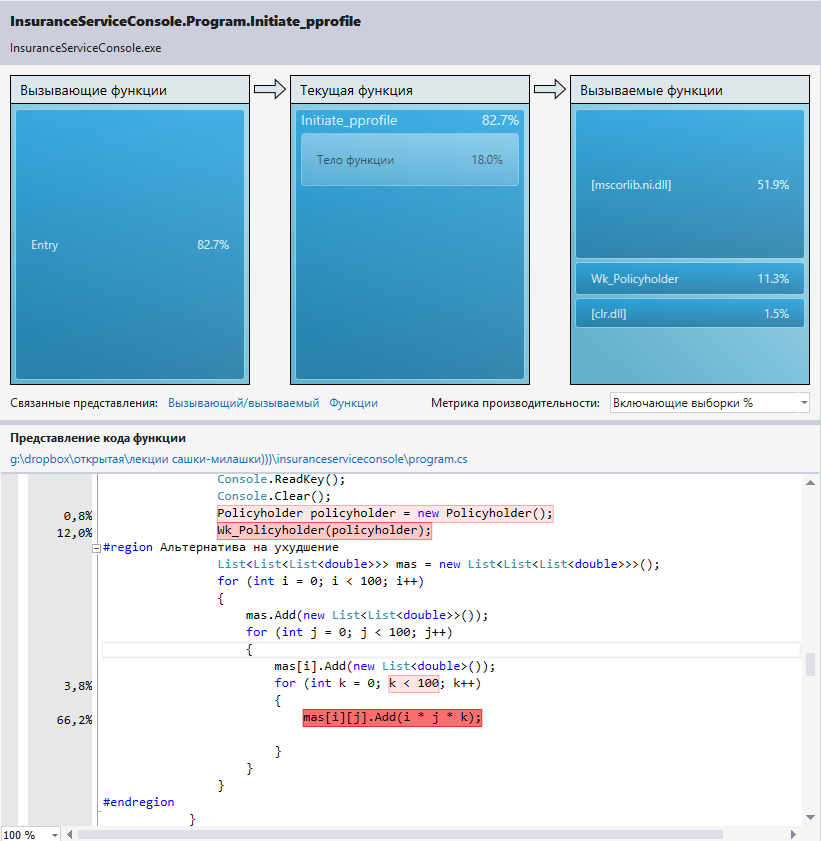


Рисунок 17 – Function Details для модифицированного InsuranceServiceConsole.Program.Initiate\_pprofile

В этом результате мы тое видим весомые изменений.

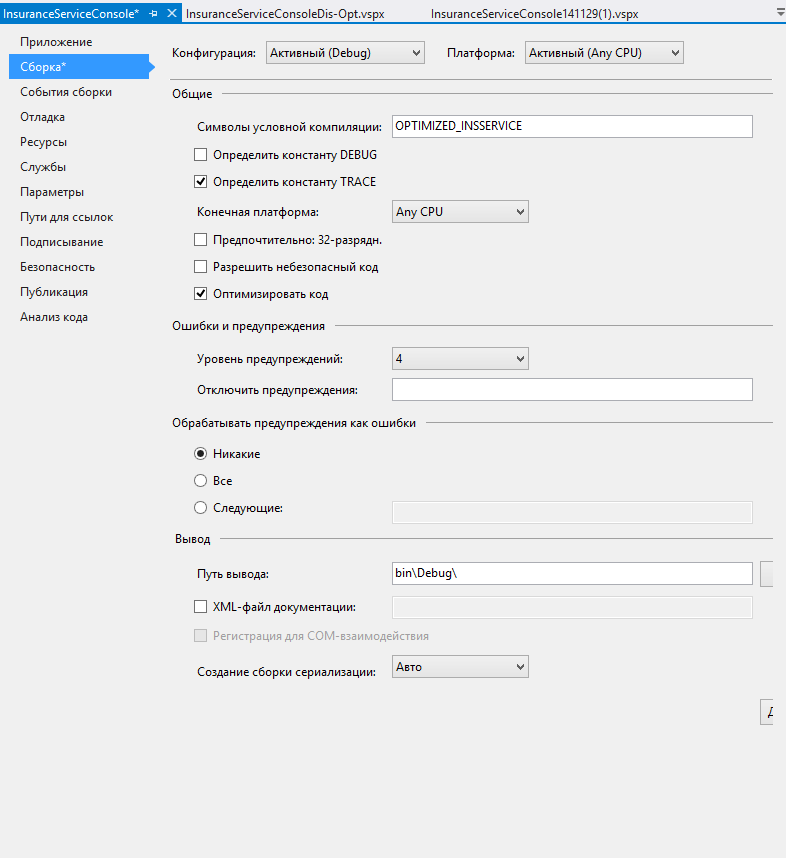


Рисунок 18 – Подключения альтернативы кода

Проведем сравнение отчетов о производительности:

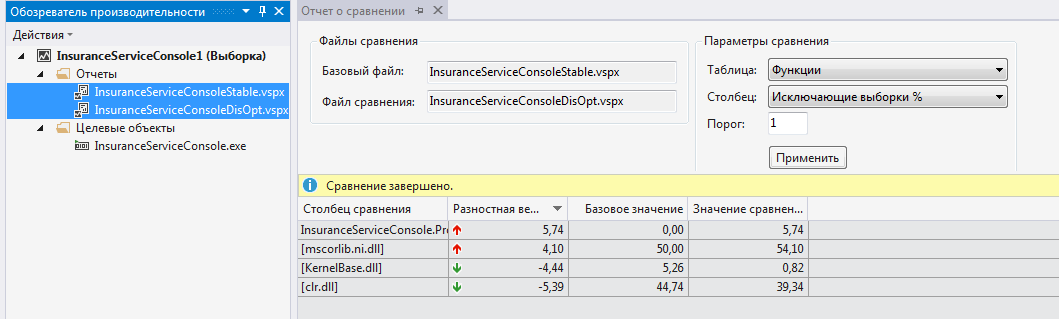


Рисунок 19 – Отчет о сравнении

Как мы видим явный регресс производительности модифицированной функции.

ВЫВОД

В ходе лабораторной работы было проведены основные подходы тестирования ПО. Были сформированы области эквивалентности и построены тестовые последовательности. Было проведено тестирование программного модуля. Были проведены ряд интеграционных тестов. Были протестированы наборы программных модулей и проведена профилировка этих модулей.