ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Научный руководитель,
Кандидат экон. наук

_____ Е.Ю. Песоцкая
«___» _____ 2020 г.

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»
профессор департамента программной
инженерии, канд. техн. наук

_____ В.В. Шилов
«___» _____2020 г.

Программа идентификации и анализа рисков ИТ проектов

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ
RU.17701729.04.01-01 12 01-1-ЛУ

Исполнитель
Студент БПИ 181
_____/Карпова А.Е./
«___»_____2020 г.

Москва 2020

Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.04.01-01 12 01-1-ЛУ				

УТВЕРЖДЕН
RU.17701729.04.01-01 12 01-1-ЛУ

Программа идентификации и анализа рисков ИТ проектов

Текст программы

RU.17701729.04.01-01 12 01-1

Листов 143

Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.04.01-01 12 01-1				

Москва 2020

Содержание

1. ТЕКСТ ПРОГРАММЫ

1.1. ActionLibrary

1.1.1. Класс DatabaseActions.cs

```
 using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
using System.IO;
 using System.Windows;

 namespace RiskApp
{
    public class DatabaseActions
    {
        SqlConnection sqlConnection;

        public DatabaseActions()
        {
            string key = @"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\Database.mdf'
;Integrated Security=False;Trusted_Connection=True";
            sqlConnection = new SqlConnection(key);
        }

        /// <summary>
        /// метод, который доавляет новый риск в проект
        /// </summary>
        /// <param name="risk"></param>
        /// <param name="project"></param>
        /// <returns></returns>
        public async Task AddRisk(string project, Risk risk)
        {
            await sqlConnection.OpenAsync();
            SqlCommand sqlCommand =
                new SqlCommand("INSERT INTO [RiskData]
(RiskName,Probability,Influence,Project,Source,Effects,PossibleSolution,Type,
Status) " +

"VALUES(@riskName,@probability,@influence,@project,@source,@effects,@possible
Solution,@type,@Status)", sqlConnection);

            sqlCommand.Parameters.AddWithValue("RiskName", risk.RiskName);
```

```
            sqlCommand.Parameters.AddWithValue("Probability",
risk.Probability);
            sqlCommand.Parameters.AddWithValue("Influence", risk.Influence);
            sqlCommand.Parameters.AddWithValue("Project", project);
            sqlCommand.Parameters.AddWithValue("Source", risk.Source);
            sqlCommand.Parameters.AddWithValue("Effects", risk.Effects);
            sqlCommand.Parameters.AddWithValue("PossibleSolution",
risk.Solution);
            sqlCommand.Parameters.AddWithValue("Type", risk.Type);
            sqlCommand.Parameters.AddWithValue("Status", risk.Status);

            await sqlCommand.ExecuteNonQueryAsync();

            sqlConnection.Close();
        }

        /// <summary>
        /// метод, который добавляет новые риски в проект
        /// </summary>
        /// <param name="project"></param>
        /// <param name="risk"></param>
        /// <param name="user"></param>
        /// <returns></returns>
        public async Task AddRisk(string project, Risk risk, User user)
        {
            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand =
                new SqlCommand("INSERT INTO [RiskData]
(RiskName,Probability,Influence,Project,Source,Effects,PossibleSolution,Type,
OwnerLogin,OwnerId,Status) " +

"VALUES(@riskName,@probability,@influence,@project,@source,@effects,@possible
Solution,@type,@OwnerLogin,@OwnerId,@Status)", sqlConnection);

            sqlCommand.Parameters.AddWithValue("RiskName", risk.RiskName);
            sqlCommand.Parameters.AddWithValue("Probability",
risk.Probability);
            sqlCommand.Parameters.AddWithValue("Influence", risk.Influence);
            sqlCommand.Parameters.AddWithValue("Project", project);
            sqlCommand.Parameters.AddWithValue("Source", risk.Source);
            sqlCommand.Parameters.AddWithValue("Effects", risk.Effects);
            sqlCommand.Parameters.AddWithValue("PossibleSolution",
risk.Solution);
            sqlCommand.Parameters.AddWithValue("Type", risk.Type);
            sqlCommand.Parameters.AddWithValue("OwnerLogin", user.Login);
            sqlCommand.Parameters.AddWithValue("OwnerId", user.ID);
            sqlCommand.Parameters.AddWithValue("Status", risk.Status);
```

```
            await sqlCommand.ExecuteNonQueryAsync();

            sqlConnection.Close();
        }

        /// <summary>
        /// метод, который меняет характеристики риска
        /// </summary>
        /// <param name="risk"></param>
        /// <returns></returns>
        public async Task ChangeRisk(Risk risk)
        {
            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand =
                new SqlCommand("UPDATE [RiskData] SET [Probability]=
@Probability, [Influence]= @Influence, [OwnerLogin]= @OwnerLogin, [Status]=
@Status, [OwnerId]= @OwnerId WHERE id=@id", sqlConnection);

            sqlCommand.Parameters.AddWithValue("Id", risk.ID);
            sqlCommand.Parameters.AddWithValue("Status", risk.Status);
            sqlCommand.Parameters.AddWithValue("Probability",
risk.Probability);
            sqlCommand.Parameters.AddWithValue("Influence", risk.Influence);
            sqlCommand.Parameters.AddWithValue("OwnerLogin",
risk.OwnerLogin);
            sqlCommand.Parameters.AddWithValue("OwnerId", risk.IdUser);

            await sqlCommand.ExecuteNonQueryAsync();

            sqlConnection.Close();
        }

        /// <summary>
        /// метод, который меняет характеристики риска для пользователя
        /// </summary>
        /// <param name="risk"></param>
        /// <param name="user"></param>
        /// <returns></returns>
        public async Task ChangeRisk(Risk risk, User user)
        {
            await sqlConnection.OpenAsync();
            SqlCommand sqlCommand =
                new SqlCommand("UPDATE [RiskData] SET [Probability]=
@Probability, [Influence]= @Influence, [OwnerLogin]= @OwnerLogin, [Status]=
@Status, [OwnerId]= @OwnerId WHERE id=@id", sqlConnection);

            sqlCommand.Parameters.AddWithValue("Id", risk.ID);
```

```
            sqlCommand.Parameters.AddWithValue("Probability",
risk.Probability);
            sqlCommand.Parameters.AddWithValue("Influence", risk.Influence);
            sqlCommand.Parameters.AddWithValue("OwnerLogin", user.Login);
            sqlCommand.Parameters.AddWithValue("OwnerId", user.ID);
            sqlCommand.Parameters.AddWithValue("Status", risk.Status);

            await sqlCommand.ExecuteNonQueryAsync();
            sqlConnection.Close();
        }

        /// <summary>
        /// метод, который возвращает список рисков проекта
        /// </summary>
        /// <param name="ProjectName"></param>
        /// <returns></returns>
        public async Task<List<Risk>> ShowRisks(Project project)
        {
            List<Risk> listRisks = new List<Risk>();
            SqlDataReader sqlDataReader = null;

            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand = new SqlCommand("SELECT * FROM[RiskData]",
sqlConnection);

            try
            {
                sqlDataReader = await sqlCommand.ExecuteReaderAsync();

                while (await sqlDataReader.ReadAsync())
                {
                    if (project.Name ==
Convert.ToString(sqlDataReader["Project"]))
                    {
                        if (Convert.ToInt32(sqlDataReader["Status"]) == 0)
                        {
                            listRisks.Add(new
Risk(Convert.ToInt32(sqlDataReader["Id"]),
                                        Convert.ToInt32(sqlDataReader["Status"]),

Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Probability"]))),

Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Influence"]))),
                                    Convert.ToString(sqlDataReader["RiskName"]),
Convert.ToString(sqlDataReader["Source"]),
                                Convert.ToString(sqlDataReader["Effects"]),
                                Convert.ToString(sqlDataReader["Type"]),
Convert.ToString(sqlDataReader["PossibleSolution"])));
```

```
                        }
                        else
                        {
                            listRisks.Add(new
Risk(Convert.ToInt32(sqlDataReader["Id"]),
                                Convert.ToInt32(sqlDataReader["Status"]),
                                Convert.ToInt32(sqlDataReader["OwnerId"]),

Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Probability"]))),

Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Influence"]))),
                                Convert.ToString(sqlDataReader["RiskName"]),
                                Convert.ToString(sqlDataReader["Source"]),
                                Convert.ToString(sqlDataReader["Effects"]),
                                Convert.ToString(sqlDataReader["Type"]),

Convert.ToString(sqlDataReader["OwnerLogin"]),

Convert.ToString(sqlDataReader["PossibleSolution"])));
                        }
                    }
                }

                return listRisks;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return null;
            }
            finally
            {
                if (sqlDataReader != null)
                    sqlDataReader.Close();

                sqlConnection.Close();
            }
        }

        /// <summary>
        /// метод,  который возвращает список проетков полльзователя
        /// </summary>
        /// <param name="user"></param>
        /// <returns></returns>
        public async Task<List<string>> ShowUserProjects(User user)
        {
            List<string> listProjects = new List<string>();
            SqlDataReader sqlDataReader = null;
```

```
            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand = new SqlCommand("SELECT * FROM[RiskData]",
sqlConnection);

            try
            {
                sqlDataReader = await sqlCommand.ExecuteReaderAsync();

                while (await sqlDataReader.ReadAsync())
                {
                    if (user.Login ==
Convert.ToString(sqlDataReader["OwnerLogin"]))
                    {
                        if
(CheckIfInList(Convert.ToString(sqlDataReader["Project"]), listProjects))

listProjects.Add(Convert.ToString(sqlDataReader["Project"]));
                    }
                }

                return listProjects;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return null;
            }
            finally
            {
                if (sqlDataReader != null)
                    sqlDataReader.Close();

                sqlConnection.Close();
            }
        }

        /// <summary>
        /// метод проверяет, находится ли проект в списке
        /// </summary>
        /// <param name="project"></param>
        /// <param name="lst"></param>
        /// <returns></returns>
        private bool CheckIfInList(string project, List<string> listProject)
        {
            for (int i = 0; i < listProject.Count; i++)
                if (project == listProject[i])
                    return false;
```

```
            return true;
        }

        /// <summary>
        /// метод для парсинга строки
        /// . заменяется на ,
        /// </summary>
        /// <param name="line"></param>
        /// <returns>возвращается строка</returns>
        private string ParseLine(string line)
        {
            string result = "";

            for (int i = 0; i < line.Length; i++)
            {
                if (line[i] == '.')
                    result += ',';
                else
                    result += line[i];
            }

            return result;
        }
    }
}
```

1.1.2. Класс Project.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RiskApp
{
    public class Project
    {
        public int Id { get; }
        public string Name { get;}
        public string Owner { get; }
        public string Type { get; }
        public Project(string name, string owner, string type)
        {
            Name = name;
            Owner = owner;
            Type = type;
        }
```

```
        public Project(int id, string name, string owner, string type)
        {
            Id = id;
            Name = name;
            Owner = owner;
            Type = type;
        }

        public override string ToString() => $"{Name}\nOwner: {Owner}\t
{Type}";
    }
}
```

1.1.3. Класс ProjectActions.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
using System.Windows;
using System.Windows.Controls;
using System.IO;

namespace RiskApp
{
    public class ProjectActions
    {
        SqlConnection sqlConnection;
        public ProjectActions()
        {
            string key = @"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\Database.mdf'
;Integrated Security=False;Trusted_Connection=True";
            sqlConnection = new SqlConnection(key);
        }

        /// <summary>
        /// метод добавляет новый проект
        /// </summary>
        /// <param name="name"></param>
        /// <param name="owner"></param>
        /// <param name="type"></param>
        public async Task AddProject(string name, string type, string owner)
        {
            if (name == "")
                throw new FormatException("Error! Inappropriate Name!");
```

```
            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand = new SqlCommand("INSERT INTO [Projects]
(Name,Owner,Type) VALUES(@Name,@OWner,@Type)", sqlConnection);

            sqlCommand.Parameters.AddWithValue("Name", name);
            sqlCommand.Parameters.AddWithValue("Owner", owner);
            sqlCommand.Parameters.AddWithValue("Type", type);

            await sqlCommand.ExecuteNonQueryAsync();

            sqlConnection.Close();
        }

        /// <summary>
        /// метод, который возвращает список всех проектов в базе
        /// </summary>
        /// <returns></returns>
        public async Task<List<Project>> ShowProjects()
        {
            List<Project> listProjects = new List<Project>();
            SqlDataReader sqlDataReader = null;

            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand = new SqlCommand("SELECT * FROM[Projects]",
sqlConnection);

            try
            {
                sqlDataReader = await sqlCommand.ExecuteReaderAsync();

                while (await sqlDataReader.ReadAsync())
                {
                    listProjects.Add(new
Project(Convert.ToInt32(sqlDataReader["id"]),
                        Convert.ToString(sqlDataReader["Name"]),
                        Convert.ToString(sqlDataReader["Owner"]),
                        Convert.ToString(sqlDataReader["Type"])));
                }

                return listProjects;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return null;
            }
            finally
            {
                if (sqlDataReader != null)
```

```
                    sqlDataReader.Close();

                sqlConnection.Close();
            }
        }

        /// <summary>
        /// метод , который возвращает список проектов владельца
        /// </summary>
        /// <param name="owner"></param>
        /// <returns></returns>
        public async Task<List<Project>> ShowUsersProjects(string user)
        {
            List<Project> listProjects = new List<Project>();
            SqlDataReader sqlDataReader = null;

            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand = new SqlCommand("SELECT * FROM[Projects]",
sqlConnection);

            try
            {
                sqlDataReader = await sqlCommand.ExecuteReaderAsync();

                while (await sqlDataReader.ReadAsync())
                {
                    if (user == Convert.ToString(sqlDataReader["Owner"]))
                        listProjects.Add(new
Project(Convert.ToInt32(sqlDataReader["id"]),
                            Convert.ToString(sqlDataReader["Name"]),
                            Convert.ToString(sqlDataReader["Owner"]),
Convert.ToString(sqlDataReader["Type"])));
                }

                return listProjects;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return null;
            }
            finally
            {
                if (sqlDataReader != null)
                    sqlDataReader.Close();

                sqlConnection.Close();
            }
        }
    }
}
```

1.1.4. Класс Risk.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Shapes;
using System.Windows;

namespace RiskApp
{
    public class Risk
    {
        private int status;
        private int idUser;
        private double probability;
        private double influence;

        public Point point;

        public int ID { get; }
        public string RiskName { get; }
        public string Source { get; }
        public string Effects { get; }
        public string Solution { get; }
        public string Type { get; }
        public string OwnerLogin { get; set; }
        public int Status
        {
            get => status;
            set { status = value; }
        }
        public int IdUser
        {
            get => idUser;
            set
            {
                idUser = value;
            }
        }
        public double Probability
        {
            get => probability;
            set
            {
                if (value > 1 || value < 0)
                    throw new ArgumentException("Probability must be in the
interval (0; 1]");

                probability = value;
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.13-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```
            }
        }
        public double Influence
        {
            get => influence;
            set
            {
                if (value > 1 || value < 0)
                    throw new ArgumentException("Influence must be in the
interval (0; 1]");

                influence = value;
            }
        }
        public double Rank { get => Influence * Probability; }
        public Risk(string riskName, string source, string effects, string
type, string description, int id)
        {
            RiskName = riskName;
            Source = source;
            Effects = effects;
            Type = type;
            Solution = description;
            ID = id;
        }

        public Risk(int id, int status, double probability, double influence,
            string riskName, string source, string effects,
            string type, string description)
        {
            ID = id;
            Status = status;
            Probability = probability;
            Influence = influence;
            RiskName = riskName;
            Source = source;
            Effects = effects;
            Type = type;
            Solution = description;
        }
        public Risk(int id, int status, int idUser, double probability,
double influence,
            string riskName, string source, string effects,
            string type, string ownerLogin, string description)
        {
            ID = id;
            Status = status;
            IdUser = idUser;
            Probability = probability;
            Influence = influence;
            RiskName = riskName;
            Source = source;
```

```
            Effects = effects;
            Type = type;
            OwnerLogin = ownerLogin;
            Solution = description;
        }

        public override string ToString() => $"Name: {RiskName}\nType:
{Type}";
    }
}
```

1.1.5. Класс RiskActions.cs

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
using System.Windows;
using System.Windows.Controls;
using System.Collections.Generic;
using System;
using System.IO;

namespace RiskApp
{
    public class RiskActions
    {
        SqlConnection sqlConnection;

        public RiskActions()
        {
            string key = @"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\Database.mdf'
;Integrated Security=False;Trusted_Connection=True";
            sqlConnection = new SqlConnection(key);
        }

        /// <summary>
        /// метод, который проверяет, находится ли элемент в списке
        /// </summary>
        /// <param name="line"></param>
        /// <param name="list"></param>
        /// <returns></returns>
        private bool CheckIfInList(string line, List<string> list)
        {
            for (int i = 0; i < list.Count; i++)
                if (line == list[i])
                    return false;

            return true;
```

```
        }

        /// <summary>
        /// метод, который возвращает список всех рисков проекта
        /// </summary>
        /// <returns></returns>
        public async Task<List<Risk>> ShowRisks()
        {
            List<Risk> listRisks = new List<Risk>();
            SqlDataReader sqlDataReader = null;

            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand = new SqlCommand("SELECT * FROM[Risks]",
sqlConnection);

            try
            {
                sqlDataReader = await sqlCommand.ExecuteReaderAsync();

                while (await sqlDataReader.ReadAsync())
                {
                    listRisks.Add(new
Risk(Convert.ToString(sqlDataReader["RiskName"]),
Convert.ToString(sqlDataReader["Source"]),
                        Convert.ToString(sqlDataReader["Effects"]),
                        Convert.ToString(sqlDataReader["Type"]),
                        Convert.ToString(sqlDataReader["PossibleSolution"]),
Convert.ToInt32(sqlDataReader["Id"])));
                }

                return listRisks;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return null;
            }
            finally
            {
                if (sqlDataReader != null)
                    sqlDataReader.Close();

                sqlConnection.Close();
            }
        }

        /// <summary>
        /// метод, который возвращает список всех источников
        /// </summary>
        /// <returns></returns>
        public async Task<List<string>> ShowSources()
```

```
        {
            List<string> listSources = new List<string>();
            SqlDataReader sqlDataReader = null;

            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand = new SqlCommand("SELECT * FROM[Risks]",
sqlConnection);

            try
            {
                sqlDataReader = await sqlCommand.ExecuteReaderAsync();

                while (await sqlDataReader.ReadAsync())
                {
                    if
(CheckIfInList(Convert.ToString(sqlDataReader["Source"]), listSources))

listSources.Add(Convert.ToString(sqlDataReader["Source"]));
                }

                return listSources;
            }
            catch (ArgumentException ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return null;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return null;
            }
            finally
            {
                if (sqlDataReader != null)
                    sqlDataReader.Close();

                sqlConnection.Close();
            }
        }
    }
}
```

1.1.6. Класс Tree.cs

```
 using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using System.Data;
using System.Data.SqlClient;
using System.IO;
 using System.Windows;

 namespace RiskApp
{
    class Tree
    {
        SqlConnection sqlConnection;

        public Tree()
        {
            string key = @"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\Database.mdf'
;Integrated Security=False;Trusted_Connection=True";
            sqlConnection = new SqlConnection(key);
        }

        /// <summary>
        /// метод, который добавляет новую вершину в базу
        /// </summary>
        /// <param name="vertex"></param>
        /// <param name="id"></param>
        /// <returns></returns>
        public async Task AddVertex(int id, Vertex vertex)
        {
            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand =
                new SqlCommand("INSERT INTO [RiskTree]
(ParentId,Description,Cost,Probability,X,Y)
VALUES(@ParentId,@Description,@Cost,@Probability,@X,@Y)", sqlConnection);

            sqlCommand.Parameters.AddWithValue("ParentId", id);
            sqlCommand.Parameters.AddWithValue("Description",
vertex.Description);
            sqlCommand.Parameters.AddWithValue("Cost", vertex.Cost);
            sqlCommand.Parameters.AddWithValue("Probability",
vertex.Probability);
            sqlCommand.Parameters.AddWithValue("X", vertex.X);
            sqlCommand.Parameters.AddWithValue("Y", vertex.Y);

            await sqlCommand.ExecuteNonQueryAsync();

            sqlConnection.Close();
        }

        /// <summary>
        /// метод, который проверяет, есть ли заданная вершина в базе данных
        /// </summary>
        /// <param name="parentId"></param>
```

```
        /// <returns></returns>
        public async Task<bool> CheckIfExistInDataBase(int idParent)
        {
            SqlDataReader sqlDataReader = null;

            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand = new SqlCommand("SELECT * FROM[RiskTree]",
sqlConnection);

            try
            {
                sqlDataReader = await sqlCommand.ExecuteReaderAsync();

                while (await sqlDataReader.ReadAsync())
                {
                    if (idParent ==
Convert.ToInt32(sqlDataReader["ParentId"]) &&
Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Probability"])))
== default)
                        return true;
                }

                return false;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return false;
            }
            finally
            {
                if (sqlDataReader != null)
                    sqlDataReader.Close();

                sqlConnection.Close();
            }
        }

        /// <summary>
        /// метод для удаления вершин
        /// </summary>
        /// <param name="listVertex"></param>
        /// <returns></returns>
        public async Task DeleteVertex(List<Vertex> listVertex)
        {
            try
            {
                await sqlConnection.OpenAsync();

                for (int i = 0; i < listVertex.Count; i++)
                {
```

```
                    SqlCommand sqlCommand = new SqlCommand("DELETE FROM
[RiskTree] WHERE [Id]=@Id", sqlConnection);

                    sqlCommand.Parameters.AddWithValue("Id",
listVertex[i].ID);
                    await sqlCommand.ExecuteNonQueryAsync();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }
            finally
            {
                sqlConnection.Close();
            }
        }

        /// <summary>
        /// метод, который показывает первую вершину графа
        /// </summary>
        /// <param name="parentId"></param>
        /// <returns></returns>
        public async Task<Vertex> ShowVertex(int idParent)
        {
            SqlDataReader sqlDataReader = null;

            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand = new SqlCommand("SELECT * FROM[RiskTree]",
sqlConnection);

            try
            {
                sqlDataReader = await sqlCommand.ExecuteReaderAsync();
                while (await sqlDataReader.ReadAsync())
                {
                    if (idParent ==
Convert.ToInt32(sqlDataReader["ParentId"]) &&
Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Probability"])))
== default)
                        return new
Vertex(Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["X"]))),

Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Y"]))),
                            Convert.ToInt32(sqlDataReader["Id"]),
Convert.ToInt32(sqlDataReader["ParentId"]),
                            Convert.ToString(sqlDataReader["Description"]));
                }

                return null;
            }
```

```
            catch (ArgumentException ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return null;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return null;
            }
            finally
            {
                if (sqlDataReader != null)
                    sqlDataReader.Close();

                sqlConnection.Close();
            }
        }

        /// <summary>
        /// метод, который возвращает вершину
        /// </summary>
        /// <param name="vertex"></param>
        /// <returns></returns>
        public async Task<Vertex> ShowVertex(Vertex vertex)
        {
            SqlDataReader sqlDataReader = null;

            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand = new SqlCommand("SELECT * FROM[RiskTree]",
sqlConnection);

            try
            {
                sqlDataReader = await sqlCommand.ExecuteReaderAsync();

                while (await sqlDataReader.ReadAsync())
                {
                    if (vertex.X ==
Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["X"]))) &&
                        vertex.Y ==
Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Y"]))) &&
                        vertex.IDParent ==
Convert.ToInt32(sqlDataReader["ParentId"]) &&
                        default !=
Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Probability"]))))
                    {
                        return new Vertex(

Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["X"]))),
```

```
Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Y"]))),

Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Cost"]))),

Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Probability"]))),
                            Convert.ToInt32(sqlDataReader["Id"]),
Convert.ToInt32(sqlDataReader["ParentId"]),
                            Convert.ToString(sqlDataReader["Description"]));
                    }
                }

                return null;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return null;
            }
            finally
            {
                if (sqlDataReader != null)
                    sqlDataReader.Close();

                sqlConnection.Close();
            }
        }


        /// <summary>
        /// метод, который возвращает все вершины
        /// </summary>
        /// <returns></returns>
        public async Task<List<Vertex>> ShowListVertexes()
        {
            SqlDataReader sqlDataReader = null;
            List<Vertex> listVertex = new List<Vertex>();

            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand = new SqlCommand("SELECT * FROM[RiskTree]",
sqlConnection);

            try
            {
                sqlDataReader = await sqlCommand.ExecuteReaderAsync();

                while (await sqlDataReader.ReadAsync())
                {
                    listVertex.Add(new
Vertex(Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["X"]))),
```

```
Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Y"]))),

Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Cost"]))),

Convert.ToDouble(ParseLine(Convert.ToString(sqlDataReader["Probability"]))),
                        Convert.ToInt32(sqlDataReader["Id"]),
Convert.ToInt32(sqlDataReader["ParentId"]),
                        Convert.ToString(sqlDataReader["Description"])));
                }

                return listVertex;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return null;
            }
            finally
            {
                if (sqlDataReader != null)
                    sqlDataReader.Close();

                sqlConnection.Close();
            }
        }

        /// <summary>
        /// метод для парсинга строки
        /// . заменяется на ,
        /// </summary>
        /// <param name="line"></param>
        /// <returns>возвращается строка</returns>
        private string ParseLine(string line)
        {
            string result = "";

            for (int i = 0; i < line.Length; i++)
            {
                if (line[i] == '.')
                    result += ',';
                else
                    result += line[i];
            }

            return result;
        }
    }
}
```

1.1.7. Класс User.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RiskApp
{
    public class User
    {
        public int ID { get; set; }
        public string Name { get; }
        public string Login { get; set; }
        public string Password { get; }
        public string Position { get; }
        public User(int id, string name, string login, string password,
string position)
        {
            ID = id;
            Name = name;
            Login = login;
            Password = password;
            Position = position;
        }
        public User() { }

        public override string ToString() => $"User's ID:
{ID},\nName:{Name},\nPosition: {Position}";
    }
}
```

1.1.8. Класс UserActions.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
using System.IO;
 using System.Windows;

 namespace RiskApp
{
    public class UserActions
    {
        SqlConnection sqlConnection;
```

```
        public UserActions()
        {
            string key = @"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\Database.mdf'
;Integrated Security=False;Trusted_Connection=True";
            sqlConnection = new SqlConnection(key);
        }

        /// <summary>
        /// метод, который проводит идентификацию пользователя
        /// </summary>
        /// <param name="login"></param>
        /// <param name="password"></param>
        /// <returns> возвращает 2, если менеджер, 1 если тестировщик
проекта, 0 если ввод неверный</returns>
        public async Task<int> CheckLogin(string login, string password)
        {
            SqlDataReader sqlDataReader = null;

            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand = new SqlCommand("SELECT * FROM[Users]",
sqlConnection);

            try
            {
                sqlDataReader = await sqlCommand.ExecuteReaderAsync();

                while (await sqlDataReader.ReadAsync())
                {
                    if (login == Convert.ToString(sqlDataReader["Login"]) &&
password == Convert.ToString(sqlDataReader["Password"]))
                    {
                        if ("MainManager" ==
Convert.ToString(sqlDataReader["Position"]))
                            return 3;
                        else
                        {
                            if ("ProjectManager" ==
Convert.ToString(sqlDataReader["Position"]))
                                return 2;

                            return 1;
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return 0;
            }
```

```
            finally
            {
                if (sqlDataReader != null)
                    sqlDataReader.Close();

                sqlConnection.Close();
            }

            return 0;
        }

        /// <summary>
        /// метод находит пользователя по логину и паролю
        /// </summary>
        /// <param name="login"></param>
        /// <param name="password"></param>
        /// <returns>возвращает пользователя</returns>
        public async Task<User> SearchForUser(string login, string password)
        {
            SqlDataReader sqlDataReader = null;

            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand = new SqlCommand("SELECT * FROM[Users]",
sqlConnection);

            try
            {
                sqlDataReader = await sqlCommand.ExecuteReaderAsync();

                while (await sqlDataReader.ReadAsync())
                {
                    if (login == Convert.ToString(sqlDataReader["Login"]) &&
password == Convert.ToString(sqlDataReader["Password"]))
                        return (new
User(Convert.ToInt32(sqlDataReader["id"]),
Convert.ToString(sqlDataReader["Name"]),
                            Convert.ToString(sqlDataReader["Login"]),
Convert.ToString(sqlDataReader["Password"]),
Convert.ToString(sqlDataReader["Position"])));
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return null;
            }
            finally
            {
                if (sqlDataReader != null)
                    sqlDataReader.Close();
```

```
                sqlConnection.Close();
            }

            return null;
        }

        /// <summary>
        /// метод, который возвращает список всех пользователей
        /// </summary>
        /// <returns></returns>
        public async Task<List<User>> ShowUsers()
        {
            SqlDataReader sqlDataReader = null;
            List<User> listUser = new List<User>();

            await sqlConnection.OpenAsync();

            SqlCommand sqlCommand = new SqlCommand("SELECT * FROM[Users]",
sqlConnection);

            try
            {
                sqlDataReader = await sqlCommand.ExecuteReaderAsync();

                while (await sqlDataReader.ReadAsync())
                {
                    listUser.Add(new
User(Convert.ToInt32(sqlDataReader["id"]),
Convert.ToString(sqlDataReader["Name"]),
                        Convert.ToString(sqlDataReader["Login"]),
Convert.ToString(sqlDataReader["Password"]),
                        Convert.ToString(sqlDataReader["Position"])));
                }

                return listUser;
            }
            catch(SqlException ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return null;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
                return null;
            }
            finally
            {
                if (sqlDataReader != null)
                    sqlDataReader.Close();

                sqlConnection.Close();
```

```
            }
        }
    }
}
```

1.1.9. Класс Vertex.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RiskApp
{
    public class Vertex
    {
        double x;
        double val;
        public double X
        {
            get => x;
            set { x = Double.Parse($"{value:f3}"); }
        }
        public double Y { get; }
        public double Value
        {
            get => val;
            set { val = double.Parse($"{value:f3}"); }
        }
        public int ID { get;}
        public int IDParent { get; }
        public double Cost { get; }
        public double Probability { get; }
        public string Description { get; }

        public Vertex(double x, double y, double cost, double probability,
int idParent, string description)
        {
            X = x;
            Y = y;
            Cost = cost;
            Probability = probability;
            IDParent = idParent;
            Description = description;
        }
        public Vertex(double x, double y, int idParent, string description)
        {
            X = x;
            Y = y;
```

```
            IDParent = idParent;
            Description = description;
        }

        public Vertex( double x, double y, int id, int idParent, string
description)
        {
            X = x;
            Y = y;
            ID = id;
            IDParent = idParent;
            Description = description;
        }

        public Vertex(double x, double y, double cost, double probability,
int id, int parentId, string description)
        {
            X = x;
            Y = y;
            Cost = cost;
            Probability = probability;
            ID = id;
            IDParent = parentId;
            Description = description;
        }
    }
}
```

1.2. Модуль AdministratorWindows

1.2.1. AdministratorGraphic.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace RiskApp
{
```

```
    public partial class AdministratorGraphic : Window
    {
        const double radius = 250;
        const int C = 100;

        double z = 1.0;
        bool flag = true;
        bool flag1 = true;

        string path =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "back.jpg");

        Project _project = null;

        List<Risk> listRisks = null;
        List<Risk> listSelected = null;
        List<string> listSource = null;

        Point mousePosition;
        Point currentCenter;
        Point center = new Point(500, 50);

        public AdministratorGraphic(Project project)
        {
            _project = project;
            InitializeComponent();
            MakeZoom();
        }

        /// <summary>
        /// метод выполянется, когда открывается окно
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void Window_Activated(object sender, EventArgs e)
        {
            try
            {
                if (flag)
                {
                    Label label = new Label();

                    label.VerticalAlignment = VerticalAlignment.Top;
                    label.HorizontalAlignment = HorizontalAlignment.Center;
                    label.FontSize = 17;
                    label.Margin = new Thickness(0, 25, 0, 0);
                    label.Content = $"Project: { _project.Name}";
                    grid.Children.Add(label);
```

```
                    BackButton.Background = new ImageBrush(new
BitmapImage(new Uri(path)));

                    DatabaseActions databaseActions = new DatabaseActions();

                    listSelected = await databaseActions.ShowRisks(_project);

                    if (listSelected == null)
                        listSelected = new List<Risk>();

                    RiskActions riskActions = new RiskActions();

                    listRisks = await riskActions.ShowRisks();
                    SeletionCombobox.Items.Add("Общие риски");
                    SeletionCombobox.Text = "Common Risks";
                    SeletionCombobox.Items.Add(_project.Type);
                    listSource = await riskActions.ShowSources();

                    AddToSelected();

                    for (int i = 0; i < listSource.Count; i++)
                        SeletionCombobox.Items.Add(listSource[i]);

                    Drawing();
                    SearchForDangerousRisks();

                    await SetUsers();
                    flag = false;
                }
            }
            catch(Exception ex)
            {
                MessageBox.Show(ex.Message + ex.TargetSite);
            }
        }


        private async Task SetUsers()
        {
            List<User> listUsers = await new UserActions().ShowUsers();

            for (int i = 0; i < listUsers.Count; i++)
            {
                Owner.Items.Add(listUsers[i]);
                NewOwnerCombobox.Items.Add(listUsers[i]);
            }
        }

        private void MakeZoom()
```

```
{
    ZoomINButton.Click += new RoutedEventHandler(ZoomIn);
    ZoomOUTButton.Click += new RoutedEventHandler(ZoomOut);
}

private void ZoomIn(object sender, RoutedEventArgs e)
{
    var scaler = grid.LayoutTransform as ScaleTransform;

    if (scaler == null)
    {
        scaler = new ScaleTransform(z, z);
        grid.LayoutTransform = scaler;
    }

    DoubleAnimation animator = new DoubleAnimation()
    {
        Duration = new Duration(TimeSpan.FromMilliseconds(600)),
    };
    if (scaler.ScaleX == z)
    {
        z = z + 0.5;
        animator.To = z;
    }
    else if (scaler.ScaleX > 1.0)
    {
        z = z + 0.5;
        animator.To = z;
    }
    scaler.BeginAnimation(ScaleTransform.ScaleXProperty, animator);
    scaler.BeginAnimation(ScaleTransform.ScaleYProperty, animator);
}

private void ZoomOut(object sender, RoutedEventArgs e)
{
    var scaler = grid.LayoutTransform as ScaleTransform;
    if (scaler == null)
    {
        scaler = new ScaleTransform(1.0, 1.0);
        grid.LayoutTransform = scaler;
    }
    DoubleAnimation animator = new DoubleAnimation()
    {
        Duration = new Duration(TimeSpan.FromMilliseconds(600)),
    };
    if (scaler.ScaleX > 1.0)
    {
        z = scaler.ScaleX - 0.5;
        animator.To = z;
```

```
            }

            scaler.BeginAnimation(ScaleTransform.ScaleXProperty, animator);
            scaler.BeginAnimation(ScaleTransform.ScaleYProperty, animator);
        }
        /// <summary>
        /// метод добавляет выбранный риск во вкладку
        /// также убирает их из выбора
        /// </summary>
        private void AddToSelected()
        {
            if (listSelected != null)
            {
                for (int i = 0; i < listSelected.Count; i++)
                {
                    if (listSelected[i].Status == 1)
                        listRisksSelected.Items.Add(listSelected[i]);
                    else
                    {
                        if (listSelected[i].Status == 0)
                            listNewRisks.Items.Add(listSelected[i]);
                        else
                            listRisksNonselected.Items.Add(listSelected[i]);
                    }
                }
            }
        }

        /// <summary>
        /// метод прооверяет, нет ли такого элемента в списке
        /// </summary>
        private void CheckIfAlreadyInProject()
        {
            if (listSelected != null)
            {
                for (int i = 0; i < listRisks.Count; i++)
                {
                    for (int j = 0; j < listSelected.Count; j++)
                    {
                        if (listRisks[i].RiskName ==
listSelected[j].RiskName)
                            listRisks.RemoveAt(i);
                    }
                }
            }
        }

        /// <summary>
        /// метод, который рисует вершины и гиперболу
```

```
        /// </summary>
        private void Drawing()
        {
            canvas.Children.Clear();

            for (int i = 0; i < C - 1; i++)
            {
                double oldX = center.X - radius + radius / C * i;
                double currentX = center.X - radius + radius / C * (i + 1);

                Line line = new Line();

                line.X1 = oldX;
                line.X2 = currentX;
                line.Y1 = FindYCoordinate(oldX, radius, center);
                line.Y2 = FindYCoordinate(currentX, radius, center);
                line.Stroke = Brushes.Black;

                canvas.Children.Add(line);
            }

            for (int i = 0; i < listRisksSelected.Items.Count; i++)
            {
                if(((Risk)listRisksSelected.Items[i]).Probability!=default &&
((Risk)listRisksSelected.Items[i]).Influence!=default &&
                    ((Risk)listRisksSelected.Items[i]).Status==1)
                {
                    ((Risk)listRisksSelected.Items[i]).point.X = 425 *
((Risk)listRisksSelected.Items[i]).Probability + 75;
                    ((Risk)listRisksSelected.Items[i]).point.Y = -350 *
((Risk)listRisksSelected.Items[i]).Influence + 400;

                    Ellipse ellipse = new Ellipse();
                    ellipse.Height = 12;
                    ellipse.Width = 12;
                    ellipse.StrokeThickness = 3;
                    if (Math.Sqrt((((Risk)listRisksSelected.Items[i]).point.X
- center.X) * (((Risk)listRisksSelected.Items[i]).point.X - center.X) +
                        (((Risk)listRisksSelected.Items[i]).point.Y -
center.Y) * (((Risk)listRisksSelected.Items[i]).point.Y - center.Y)) <
radius)
                    {
                        ellipse.Stroke = Brushes.Red;
                        ellipse.Fill = Brushes.Red;
                    }
                    else
                    {
                        ellipse.Stroke = Brushes.Green;
                        ellipse.Fill = Brushes.Green;
```

```
                    }

                    ellipse.HorizontalAlignment = HorizontalAlignment.Left;
                    ellipse.Margin = new
Thickness(((Risk)listRisksSelected.Items[i]).point.X,
                        ((Risk)listRisksSelected.Items[i]).point.Y, 0, 0);
                    ellipse.VerticalAlignment = VerticalAlignment.Top;

                    canvas.Children.Add(ellipse);
                }
            }
        }

        /// <summary>
        /// метод, который вычисляет координату y с помощью квадратного
уравнения
        /// </summary>
        /// <param name="x"></param>
        /// <param name="radius"></param>
        /// <param name="center"></param>
        /// <returns></returns>
        /// <exception cref="Exception"></exception>
        static public double FindYCoordinate(double x, double radius, Point
center)
        {
            double a = 1;
            double b = -2 * center.Y;
            double c = -(radius * radius) + (x - center.X) * (x - center.X) +
center.Y * center.Y;
            double d = (b * b - 4 * a * c);

            if (d < 0)
                throw new Exception("There's no such point!");

            return (-b + Math.Sqrt(d)) / 2 * a;
        }

        /// <summary>
        /// метод срабатывает при нажатии на кнопку мыши
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Canvas_MouseDown(object sender, MouseButtonEventArgs e)
        {
            if (e.GetPosition(null).X < 540 && e.GetPosition(null).Y < 450)
            {
                mousePosition = e.GetPosition(null);
                currentCenter = center;
            }
```

```
        }

        private void SelectComboBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            listOfRisks.Items.Clear();
            CheckIfAlreadyInProject();

            if (flag){ }

            if (SeletionCombobox.SelectedItem.ToString() == "Общие риски")
            {
                for (int i = 0; i < listRisks.Count; i++)
                {
                    if (listRisks[i].Type == "default")
                        listOfRisks.Items.Add(listRisks[i]);
                }
            }
            else
            {
                if (SeletionCombobox.SelectedItem.ToString() ==
_project.Type)
                {
                    for (int i = 0; i < listRisks.Count; i++)
                        if (listRisks[i].Type == _project.Type)
                            listOfRisks.Items.Add(listRisks[i]);
                }
                else
                {
                    for (int i = 0; i < listSource.Count; i++)
                    {
                        if (SeletionCombobox.SelectedItem.ToString() ==
listSource[i])
                        {
                            for (int j = 0; j < listRisks.Count; j++)
                            {
                                if ((listRisks[j].Type == _project.Type ||
listRisks[j].Type == "default") &&
                                    listRisks[j].Source == listSource[i])
                                {
                                    listOfRisks.Items.Add(listRisks[j]);
                                }
                            }

                        }
                    }
                }
            }
        }
```

```
        private void ChangeSelected()
        {
            if (SeletionCombobox.SelectedItem != null)
                SeletionCombobox.SelectedItem = "Common Risks";

            listOfRisks.Items.Clear();
            CheckIfAlreadyInProject();

            if (flag) { }

            if (SeletionCombobox.SelectedItem.ToString() == "Common Risks")
            {
                for (int i = 0; i < listRisks.Count; i++)
                {
                    if (listRisks[i].Type == "default")
                        listOfRisks.Items.Add(listRisks[i]);
                }
            }
            else
            {
                if (SeletionCombobox.SelectedItem.ToString() ==
_project.Type)
                {
                    for (int i = 0; i < listRisks.Count; i++)
                    {
                        if (listRisks[i].Type == _project.Type)
                            listOfRisks.Items.Add(listRisks[i]);
                    }
                }
                else
                {
                    for (int i = 0; i < listSource.Count; i++)
                    {
                        if (SeletionCombobox.SelectedItem.ToString() ==
listSource[i])
                        {
                            for (int j = 0; j < listRisks.Count; j++)
                            {
                                if ((listRisks[j].Type == _project.Type ||
listRisks[j].Type == "default") && listRisks[j].Source == listSource[i])
                                    listOfRisks.Items.Add(listRisks[j]);
                            }
                        }
                    }
                }
            }
        }
```

```
        /// <summary>
        /// проверяет нет ли в выбранных уже такого элемента
        /// </summary>
        /// <param name="checkrisk"></param>
        /// <returns></returns>
        private bool CheckIfAlreadySelected(Risk risk)
        {
            if (listRisksSelected.Items.Count != 0)
            {
                for (int i = 0; i < listRisksSelected.Items.Count; i++)
                {
                    if (risk.ID == ((Risk)listRisksSelected.Items[i]).ID)
                        return false;
                }

                return true;
            }

            return true;
        }

        /// <summary>
        /// метод, который срабатывает при нажатии кнопки и
        /// устанавливает характерстики для риска и добавляет в бд
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void SetUpRisk_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                if (Double.Parse(ParseLine(InfluenceTextbox.Text)) == default
|| Double.Parse(ParseLine(ProbabilityTextbox.Text)) == default)
                    throw new ArgumentException("Values of Probability and
Influence fields must stay (0,1)");

                if(Owner.SelectedItem == null)
                    throw new NullReferenceException("You need to choose the
user you want to assign risk to!");

                ((Risk)listRisksSelected.SelectedItem).Influence =
double.Parse(ParseLine(InfluenceTextbox.Text));
                ((Risk)listRisksSelected.SelectedItem).Probability =
double.Parse(ParseLine(ProbabilityTextbox.Text));

                DatabaseActions databaseActions = new DatabaseActions();
```

```
                await
databaseActions.ChangeRisk((Risk)listRisksSelected.SelectedItem,
(User)Owner.SelectedItem);

                listRisksSelected.Items.Clear();
                listSelected = await databaseActions.ShowRisks(_project);

                for (int i = 0; i < listSelected.Count; i++)
                {
                    if(listSelected[i].Status==1)
                        listRisksSelected.Items.Add(listSelected[i]);
                }

                Drawing();
            }
            catch (NullReferenceException ex)
            {
                MessageBox.Show(ex.Message,"Exception");

            }
            catch (ArgumentException ex)
            {
                MessageBox.Show(ex.Message, "Exception");

            }
            catch (Exception)
            {
                MessageBox.Show("Something went wrong!");

            }
        }

        /// <summary>
        /// метод для парсинга строки
        /// . заменяется на ,
        /// </summary>
        /// <param name="line"></param>
        /// <returns>возвращается строка</returns>
        private string ParseLine(string line)
        {
            string result = "";

            for (int i = 0; i < line.Length; i++)
            {
                if (line[i] == '.')
                    result += ',';
                else
                    result += line[i];
            }
```

```
            return result;
        }

        /// <summary>
        /// вводим значение рисков в текстбоксы
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Selected_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            if (listRisksSelected.SelectedItem!=null)
            {
                InfluenceTextbox.Text =
((Risk)listRisksSelected.SelectedItem).Influence.ToString();
                ProbabilityTextbox.Text =
((Risk)listRisksSelected.SelectedItem).Probability.ToString();
            }
        }

        /// <summary>
        /// перемещие гиперболы во время движения мыши
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Canvas_MouseMove(object sender, MouseEventArgs e)
        {
            if (e.GetPosition(null).X < 540 && e.GetPosition(null).Y < 450 &&
e.LeftButton == MouseButtonState.Pressed)
            {
                center.X = currentCenter.X - mousePosition.X +
e.GetPosition(null).X;
                center.Y = currentCenter.Y + (mousePosition.X -
e.GetPosition(null).X) / 1.2;

                if (center.X > 650 || center.Y < -100)
                {
                    center.Y = -100;
                    center.X = 650;
                }
                if (center.Y > 230 || center.X < 250)
                {
                    center.Y = 230;
                    center.X = 250;
                }
                Drawing();
                SearchForDangerousRisks();
            }
```

```
        }

        /// <summary>
        /// метод, который ищет среди рисков проекта опасные и добавляет в
раздел Dangerous Risks
        /// </summary>
        private void SearchForDangerousRisks()
        {
            listDangerous.Items.Clear();

            if (listRisksSelected.Items.Count != 0)
            {
                for (int i = 0; i < listRisksSelected.Items.Count; i++)
                {
                    double c = Math.Sqrt((((Risk)
listRisksSelected.Items[i]).point.X - center.X) *
                                         (((Risk)
listRisksSelected.Items[i]).point.X - center.X) +
                                         (((Risk)
listRisksSelected.Items[i]).point.Y - center.Y) *
                                         (((Risk)
listRisksSelected.Items[i]).point.Y - center.Y));
                    if (c < radius)

listDangerous.Items.Add((Risk)listRisksSelected.Items[i]);
                }
            }
        }

        /// <summary>
        /// метод, который выводит в MessageBox информацию о выбранном на
графике риске
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Canvas_MouseRightButtonDown(object sender,
MouseButtonEventArgs e)
        {
            if (listRisksSelected.Items.Count != 0)
            {
                List<Risk> list = new List<Risk>();

                for (int i = 0; i < listRisksSelected.Items.Count; i++)
                {
                    double _x = Math.Abs(((Risk)
listRisksSelected.Items[i]).point.X - e.GetPosition(null).X);
                    double _y = Math.Abs(((Risk)
listRisksSelected.Items[i]).point.Y - e.GetPosition(null).Y);
```

```
                    if ( _x <= 10 && _y <= 10)
                        list.Add((Risk)listRisksSelected.Items[i]);
                }

                if(list.Count!=0)
                    MessageBox.Show(CreateLine(list), "Information about
Selected Risks");
            }
        }

        /// <summary>
        /// метод, который возвращает строку с необходимой информацией
        /// </summary>
        /// <param name="click"></param>
        /// <returns></returns>
        private string CreateLine(List<Risk> list)
        {
            string line = "";

            for (int i = 0; i < list.Count; i++)
                line += $"Name of Risk: {listRisks[i].RiskName}\nSource:
{listRisks[i].Source}\nType: {listRisks[i].Type}\nEffects:
{listRisks[i].Effects}\nPossible Solution: {listRisks[i].Solution}";

            return line;
        }

        /// <summary>
        /// метод для удаления
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void Delete_Click(object sender, RoutedEventArgs e)
        {
            DatabaseActions databaseActions = new DatabaseActions();
            Risk risk = (Risk)((Button)sender).DataContext;
            User user = new User();

            if (listSelected == null)
                listSelected = new List<Risk>();

            listRisksSelected.Items.Remove(risk);
            risk.Status = 2;
            SearchForCurrentRisk(risk);

            user.ID = risk.IdUser;
            user.Login = risk.OwnerLogin;

            await databaseActions.ChangeRisk(risk);
```

```
            listRisksNonselected.Items.Add(risk);
            Drawing();
        }

        /// <summary>
        /// метод ищет риск по его номеру в списке выбранных рисков
        /// </summary>
        /// <param name="risk"></param>
        private void SearchForCurrentRisk(Risk risk)
        {
            for (int i = 0; i < listSelected.Count; i++)
            {
                if(listSelected[i].ID == risk.ID)
                {
                    listSelected[i].Status = risk.Status;
                    listSelected[i].IdUser = risk.IdUser;
                    listSelected[i].OwnerLogin = risk.OwnerLogin;
                }
            }
        }

        /// <summary>
        /// метод добавляет риск в проект
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void AddToProject_Click(object sender, RoutedEventArgs
e)
        {
            if (CheckIfAlreadySelected(((Risk)((Button)sender).DataContext)))
            {
                RiskSettingsWindow settingsWindow = new RiskSettingsWindow();
                Risk risk = (Risk)(((Button)sender).DataContext);

                if (settingsWindow.ShowDialog() == true)
                {
                    try
                    {
                        risk.Probability = settingsWindow.Probability;
                        risk.Influence = settingsWindow.Influence;

                        if (settingsWindow.Influence != default)
                            ((Risk)((Button)sender).DataContext).Status = 1;
                        else
                            ((Risk)((Button)sender).DataContext).Status = 0;

                        DatabaseActions databaseActions = new
DatabaseActions();
```

```
                        if (settingsWindow.Owner == null)
                            await databaseActions.AddRisk(_project.Name,
risk);
                        else
                        {
                            risk.OwnerLogin = settingsWindow.Owner.Login;
                            risk.IdUser = settingsWindow.Owner.ID;
                            await databaseActions.AddRisk(_project.Name,
risk, settingsWindow.Owner);
                        }

                        listSelected.Add(risk);

                        SearchForCurrentRisk(risk);
                        ChangeSelected();

                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show("Wrong in enpty" + ex.Message +
ex.Source);
                    }
                }

                listRisksSelected.Items.Clear();
                listNewRisks.Items.Clear();

                for (int i = 0; i < listSelected.Count; i++)
                {
                    if (listSelected[i].Status == 1)
                        listRisksSelected.Items.Add(listSelected[i]);

                    if (listSelected[i].Status == 0)
                        listNewRisks.Items.Add(listSelected[i]);
                }

                Drawing();
                listRisks.Remove(risk);
                SeletionCombobox.SelectedItem =
SeletionCombobox.SelectedItem;
            }
            else
            {
                MessageBox.Show("Данный элемент уже выбран");
            }
        }

        /// <summary>
        /// метод выполняется при двойном нажатии на риск из раздела опасных
```

```
        /// и открывает окно дерева рисков
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void DangerousRisks_MouseDoubleClick(object sender,
MouseButtonEventArgs e)
        {
            if(listDangerous.SelectedItem != null && flag1)
            {
                AdminTree adminTree = new
AdminTree((Risk)listDangerous.SelectedItem, _project, center);

                Close();
                adminTree.Show();

                flag1 = false;
            }
        }

        /// <summary>
        /// метод выполняется при нажатии кнопки Back, возвращает к окну
выбора проектов
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void GetBack_Click(object sender, RoutedEventArgs e)
        {
            AdminProjects project = new AdminProjects();
            Close();

            project.Show();
        }

        /// <summary>
        /// метод , который добавляет риск в раздел активных рисков таблице
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void AddToActiveRisks_Click(object sender,
RoutedEventArgs e)
        {
            Risk risk = (Risk)((Button)sender).DataContext;

            listRisksNonselected.Items.Remove(risk);
            listSelected.Add(risk);
            listRisksSelected.Items.Add(risk);

            risk.Status = 1;
            SearchForCurrentRisk(risk);
```

```
            DatabaseActions databaseActions = new DatabaseActions();

            await databaseActions.ChangeRisk(risk);
            Drawing();
        }

        /// <summary>
        /// метод выполняется при нажатии кнопки Set Up в разделе таблицы New
Risks
        /// устанавливает значения характеристик для нового риска
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void SetUpNewRisk_Click(object sender, RoutedEventArgs
e)
        {
            try
            {
                if (listNewRisks.SelectedItems != null &&
NewOwnerCombobox.SelectedItem != null &&
                    Double.Parse(ParseLine(NewInfluenceTextbox.Text)) !=
default &&
                    Double.Parse(ParseLine(NewProbabilityTextbox.Text)) !=
default)
                {
                    ((Risk)listNewRisks.SelectedItem).Status = 1;
                    ((Risk)listNewRisks.SelectedItem).IdUser =
((User)NewOwnerCombobox.SelectedItem).ID;
                    ((Risk)listNewRisks.SelectedItem).OwnerLogin =
((User)NewOwnerCombobox.SelectedItem).Login;

                    ((Risk)listNewRisks.SelectedItem).Influence =
double.Parse(ParseLine(NewInfluenceTextbox.Text));
                    ((Risk)listNewRisks.SelectedItem).Probability =
double.Parse(ParseLine(NewProbabilityTextbox.Text));

                    SearchForCurrentRisk(((Risk)listNewRisks.SelectedItem));

                    DatabaseActions databaseActions = new DatabaseActions();

                    await
databaseActions.ChangeRisk((Risk)listNewRisks.SelectedItem,
(User)NewOwnerCombobox.SelectedItem);

                    listNewRisks.Items.Clear();
                    listRisksSelected.Items.Clear();

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

```
                    {
                        if(listSelected[i].Status==0)
                            listNewRisks.Items.Add(listSelected[i]);

                        if (listSelected[i].Status == 1)
                            listRisksSelected.Items.Add(listSelected[i]);
                    }

                    Drawing();
                }
                else
                {
                    MessageBox.Show("Wrong in enpty1");
                }
            }
            catch (Exception)
            {
                MessageBox.Show("Wrong in enpty2");
            }
        }

        /// <summary>
        /// метод для элемента combobox
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void NewRisksBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            if (listNewRisks.SelectedItem != null)
            {
                NewInfluenceTextbox.Text = "0";
                NewProbabilityTextbox.Text = "0";
            }
        }
    }
}
```

1.2.2. AdminNewProject.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
```

```
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace RiskApp
{
    /// <summary>
    /// Логика взаимодействия для NewProject.xaml
    /// </summary>
    public partial class AdminNewProject : Window
    {
        string path =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "back.jpg");
        bool flag =true;
        public AdminNewProject()
        {
            InitializeComponent();
        }

        /// <summary>
        /// метод срабатывает при нажатии кнопки Back-кнопки возврата в
предыдущее окно
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void BackButton_Click(object sender, RoutedEventArgs e)
        {
            AdminProjects project = new AdminProjects();
            Close();

            project.Show();
        }

        /// <summary>
        /// метод, который при нажатии кнопки Create Project создаёт новый
проект с заданными пользователем параметрами
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void CreateButton_Click(object sender, RoutedEventArgs
e)
        {
            try
            {
                ProjectActions projectActions = new ProjectActions();

                if (TypeCombobox.Text == "Choose Type of the Project")
                    MessageBox.Show("You have to choose the type of a
project!");
                else if (Name.Text == null)
```

```
                    MessageBox.Show("You have to enter the name of the
project!");
                else if (listOwners.SelectedItem == null)
                    MessageBox.Show("You have to choose project's owner!");
                else
                {
                    await projectActions.AddProject(Name.Text,
TypeCombobox.Text, ((User)(listOwners.SelectedItem)).Login);
                    AdministratorGraphic graphic = new
AdministratorGraphic(new Project(Name.Text,
((User)(listOwners.SelectedItem)).Login, TypeCombobox.Text));
                    Close();
                    graphic.Show();
                }
}
            catch(ArgumentException ex)
            {
                MessageBox.Show(ex.Message);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        /// <summary>
        /// метод, который срабатывает при активации окна
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void Window_Activated(object sender, EventArgs e)
        {
            if (flag)
            {
                BackButton.Foreground = new ImageBrush(new BitmapImage(new
Uri(path)));
                BackButton.Background = new ImageBrush(new BitmapImage(new
Uri(path)));

                TypeCombobox.Text = "Choose Type of the Project";

                List<User> listUsers = await new UserActions().ShowUsers();

                for (int i = 0; i < listUsers.Count; i++)
                {
                    if (listUsers[i].Position != "RiskManager")
                        listOwners.Items.Add(listUsers[i]);
                }
                flag = false;
            }
        }
    }
```

```
}
```

1.2.3. AdminProjects.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace RiskApp
{
    /// <summary>
    /// Логика взаимодействия для Projects.xaml
    /// </summary>
    public partial class AdminProjects : Window
    {
        bool flag = true;
        string path =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "back.jpg");

        public AdminProjects()
        {
            InitializeComponent();
        }

        /// <summary>
        /// метод, запускающийся при активации окна
        /// отображается список существующих проектов пользователя
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void Window_Activated(object sender, EventArgs e)
        {
            if (flag)
            {
                BackButton.Background = new ImageBrush(new BitmapImage(new
Uri(path)));
                BackButton.Foreground = new ImageBrush(new BitmapImage(new
Uri(path)));

                ProjectActions projectActions = new ProjectActions();
```

```
                List<Project> listProjects = await
projectActions.ShowProjects();

                for (int i = 0; i < listProjects.Count; i++)
                {
                    listBoxProjects.Items.Add(listProjects[i]);
                    listBoxProjects.Items.ToString();
                }

                flag = false;
            }
        }


        /// <summary>
        /// метод, котрый запускается при нажатии кнопки Back
        /// возвращает окно ввода логина и пароля
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void BackButton_Click(object sender, RoutedEventArgs e)
        {
            MainWindow mainWindow = new MainWindow();
            Close();
            mainWindow.Show();
        }


        /// <summary>
        /// метод, который срабатывает при нажатии кнопки Create New Project
        /// запускает окно для создания нового проекта
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void CreateButton_Click(object sender, RoutedEventArgs e)
        {
            AdminNewProject newProject = new AdminNewProject();
            Close();
            newProject.Show();
        }

        /// <summary>
        /// метод, который запускается при двойном нажатии кнопкой мыши на
один из элементов
        /// списка в ListBox
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ListBox_MouseDoubleClick(object sender,
MouseButtonEventArgs e)
        {
            if (listBoxProjects.SelectedItem != null)
```

```
            {
                Project project = (Project)listBoxProjects.SelectedItem;
                AdministratorGraphic graphic = new
AdministratorGraphic(project);

                Close();
                graphic.Show();
            }
            else
            {
                MessageBox.Show("You need to select project or click the
button to create a new one!");
            }
        }
    }
}
```

1.2.4. AdminTree.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace RiskApp
{
    public partial class AdminTree : Window
    {
        bool flag = true;

        double Width;
        new double Height;

        string pathToBack =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "back.jpg");
        string pathToPlus =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "plus.png");

        Point center;
```

```
        Risk risk = null;
        Project project;
        Vertex _vertexFirst;

        //List<double> listValues = new List<double>();
        List<Vertex> listVertex = new List<Vertex>();
        public AdminTree(Risk risk, Project project,Point center)
        {
            this.center = center;
            this.risk = risk;
            this.project = project;

            InitializeComponent();
        }

        /// <summary>
        /// метод запускается при открытии окна
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void Window_Activated(object sender, EventArgs e)
        {
            if (flag)
            {
                try
                {
                    Label label = new Label();
                    Tree tree = new Tree();
                    Button button = new Button();

                    Width = canvas.ActualWidth;
                    Height = canvas.ActualHeight;

                    label.HorizontalAlignment = HorizontalAlignment.Center;
                    label.VerticalAlignment = VerticalAlignment.Top;
                    label.Content = risk.RiskName;
                    grid.Children.Add(label);

                    button.HorizontalAlignment = HorizontalAlignment.Left;
                    button.VerticalAlignment = VerticalAlignment.Top;
                    button.Margin = new Thickness(Width / 2 - 10, 50, Width /
2 - 10, Height - 70);
                    button.Background = new ImageBrush(new BitmapImage(new
Uri(pathToPlus)));

                    BackButton.Background = new ImageBrush(new
BitmapImage(new Uri(pathToBack)));
                    BackButton.Foreground = new ImageBrush(new
BitmapImage(new Uri(pathToBack)));
```

```
                    if (!await tree.CheckIfExistInDataBase(risk.ID))
                    {
                        _vertexFirst = new Vertex(Width / 2, 50, risk.ID,
risk.RiskName);
                        await tree.AddVertex(risk.ID, _vertexFirst);
                        _vertexFirst = await tree.ShowVertex(risk.ID);
                        button.DataContext = _vertexFirst;
                    }
                    else
                    {
                        _vertexFirst = await tree.ShowVertex(risk.ID);
                        button.DataContext = _vertexFirst;
                    }

                    button.Height = 20;
                    button.Width = 20;
                    button.Click += Button_Click;

                    canvas.Children.Add(button);
                    listVertex = await tree.ShowListVertexes();

                    DrawRoot(_vertexFirst);
                    CurrentBranchCost(_vertexFirst, 0);
                    DrawDangerousMaximum();

                    flag = false;
                }
                catch (NullReferenceException ex)
                {
                    MessageBox.Show("Error: " + ex.Message);
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Error: " + ex.Message);
                }
            }
        }

        /// <summary>
        /// метод, который добавляет новую вершину с установленными
характеристиками
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void AddButton_Click(object sender, RoutedEventArgs e)
        {
            try
            {
```

```
                if (((Button)sender).DataContext == null)
                    throw new Exception("You need to choose the top!");

                if (TextboxDescription.Text == "")
                    throw new Exception("You must fill the field
'Description'!");

                if (!double.TryParse(TextboxCost.Text, out double d) || d <=
0)
                    throw new Exception("The value of 'Cost' can only be of
Double type and it must be more than 0!");

                if (!double.TryParse(TextboxProbability.Text, out double d1)
|| d1 > 1 || d1 < 0)
                    throw new Exception("The value of 'Probability' can only
be of Double type and it must be in the interval [0, 1]!");

                Vertex parentVertex = ((Vertex)((Button)sender).DataContext);

                int row = 1;

                SearchForRow(parentVertex, ref row);

                if (parentVertex.Probability != 0)
                    row++;

                if (row >= 4)
                    throw new ArgumentException("Branches cannot be more than
4!");

                Vertex currentVertex;
                string line = $"{(parentVertex.X - Width / (2 * Math.Pow(4,
row))):f3}";

                if (CheckIfInList(double.Parse(line)))
                    currentVertex = new Vertex(parentVertex.X - Width / (2 *
Math.Pow(4, row)), parentVertex.Y + 50, double.Parse(TextboxCost.Text),
                        double.Parse(TextboxProbability.Text),
parentVertex.ID, TextboxDescription.Text);
                else
                {
                    line = $"{(parentVertex.X + Width / (2 * Math.Pow(4,
row))):f3}";

                    if (CheckIfInList(double.Parse(line)))
                        currentVertex = new Vertex(parentVertex.X + Width /
(2 * Math.Pow(4, row)), parentVertex.Y + 50, double.Parse(TextboxCost.Text),
double.Parse(TextboxProbability.Text),
                            parentVertex.ID, TextboxDescription.Text);
```

```
                    else
                    {
                        line = $"{(parentVertex.X - 3 * Width / (2 *
Math.Pow(4, row))):f3}";

                        if (CheckIfInList(double.Parse(line)))
                            currentVertex = new Vertex(parentVertex.X - 3 *
Width / (2 * Math.Pow(4, row)), parentVertex.Y + 50,
                                double.Parse(TextboxCost.Text),
double.Parse(TextboxProbability.Text), parentVertex.ID,
TextboxDescription.Text);
                        else
                        {
                            line = $"{(parentVertex.X + 3 * Width / (2 *
Math.Pow(4, row))):f3}";

                            if (CheckIfInList(double.Parse(line)))
                                currentVertex = new Vertex(parentVertex.X + 3
* Width / (2 * Math.Pow(4, row)), parentVertex.Y + 50,
double.Parse(TextboxCost.Text), double.Parse(TextboxProbability.Text),
parentVertex.ID, TextboxDescription.Text);
                            else
                                throw new Exception("The amount of children
cannot be more than 4!");
                        }
                    }
                }

                Tree tree = new Tree();

                await tree.AddVertex(parentVertex.ID, currentVertex);
                currentVertex = await tree.ShowVertex(currentVertex);
                listVertex.Add(currentVertex);

                RefreshTree(_vertexFirst);

                for (int i = 0; i < listVertex.Count; i++)
                    listVertex[i].Value = default;

                CurrentBranchCost(_vertexFirst, 0);
                DrawDangerousMaximum();
            }
            catch (ArgumentException ex)
            {
                MessageBox.Show(ex.Message, "Enpty Exception");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Enpty Exception");
```

```
            }
        }

        /// <summary>
        /// метод для 'обновления' дерева
        /// добавляются, либо удаляются элементы
        /// </summary>
        /// <param name="currentVertex"></param>
        private void RefreshTree(Vertex currentVertex)
        {
            canvas.Children.Clear();
            Button button = new Button();

            button.HorizontalAlignment = HorizontalAlignment.Left;
            button.VerticalAlignment = VerticalAlignment.Top;
            button.Margin = new Thickness(Width / 2 - 10, 50, Width / 2 - 10,
Height - 70);

            button.Background = new ImageBrush(new BitmapImage(new
Uri(pathToPlus)));
            BackButton.Background = new ImageBrush(new BitmapImage(new
Uri(pathToBack)));

            button.DataContext = _vertexFirst;
            button.Height = 20;
            button.Width = 20;
            button.Click += Button_Click;

            canvas.Children.Add(button);

            for (int i = 0; i < listVertex.Count; i++)
            {
                if (listVertex[i].IDParent == currentVertex.ID &&
listVertex[i].Probability != default)
                {
                    AddNewVertexToTree(listVertex[i]);
                    DrawNewLine(listVertex[i], currentVertex);
                    DrawRoot(listVertex[i]);
                }
            }
        }

        /// <summary>
        /// метод проверяет по значению вершины x, находится ли такой же
элемент в списке
        /// </summary>
        /// <param name="x"></param>
        /// <returns></returns>
        private bool CheckIfInList(double x)
```

```
        {
            for (int i = 0; i < listVertex.Count; i++)
                if (x == listVertex[i].X)
                    return false;

            return true;
        }

        /// <summary>
        /// метод для установления стоимости ветки
        /// </summary>
        /// <param name="curver"></param>
        /// <param name="k"></param>
        private void CurrentBranchCost(Vertex currentVertex, double c)
        {
            double cost = c;

            for (int i = 0; i < listVertex.Count; i++)
            {
                if (currentVertex.ID == listVertex[i].IDParent &&
listVertex[i].Probability != default)
                {
                    cost += listVertex[i].Probability * listVertex[i].Cost;
                    CurrentBranchCost(listVertex[i], cost);
                    cost -= listVertex[i].Probability * listVertex[i].Cost;
                }
            }

            if (CheckIfChildExists(currentVertex))
                currentVertex.Value = cost;
        }

        /// <summary>
        /// метод, котоый проверяет, есть ли у вершины ребенок
        /// </summary>
        /// <param name="currentVertex"></param>
        /// <returns></returns>
        private bool CheckIfChildExists(Vertex currentVertex)
        {
            for (int i = 0; i < listVertex.Count; i++)
                if (currentVertex.ID == listVertex[i].IDParent &&
listVertex[i].Probability != default)
                    return false;

            return true;
        }


        /// <summary>
```

```
        /// метод, который возвращает ряд дерева
        /// </summary>
        /// <param name="parent"></param>
        /// <param name="k"></param>
        private void SearchForRow(Vertex parentVertex, ref int k)
        {
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (parentVertex.IDParent == listVertex[i].ID &&
listVertex[i].Probability == default)
                    return;

                if (parentVertex.IDParent == listVertex[i].ID &&
listVertex[i].Probability != default)
                {
                    k++;
                    SearchForRow(listVertex[i], ref k);
                }
            }
        }

        /// <summary>
        /// метод, который рисует линии между вершинами
        /// </summary>
        /// <param name="newver"></param>
        /// <param name="parent"></param>
        private void DrawNewLine(Vertex vertex, Vertex parentVertex)
        {
            Line line = new Line();

            line.X1 = parentVertex.X;
            line.Y1 = parentVertex.Y + 20;
            line.X2 = vertex.X;
            line.Y2 = vertex.Y;
            line.Stroke = Brushes.Black;

            canvas.Children.Add(line);
        }

        /// <summary>
        /// метод, который добавляет новые вершины
        /// </summary>
        /// <param name="newver"></param>
        private void AddNewVertexToTree(Vertex vertex)
        {
            Button plusButton = new Button();

            plusButton.DataContext = vertex;
            plusButton.HorizontalAlignment = HorizontalAlignment.Left;
```

```
            plusButton.VerticalAlignment = VerticalAlignment.Top;
            plusButton.Margin = new Thickness(vertex.X - 10, vertex.Y, Width
- vertex.X - 10, Height - vertex.Y - 20);
            plusButton.Height = 20;
            plusButton.Width = 20;
            plusButton.Background = new ImageBrush(new BitmapImage(new
Uri(pathToPlus)));
            plusButton.Click += Button_Click;

            canvas.Children.Add(plusButton);
        }

        /// <summary>
        /// метод, который находит наиболее опасный рис и рисует его
        /// </summary>
        private void DrawDangerousMaximum()
        {
            Vertex vertexMaximum = listVertex[0];
            Vertex vertexMinimum = null;

            for (int i = 0; i < listVertex.Count; i++)
            {
                if (listVertex[i].Value != 0)
                {
                    vertexMinimum = listVertex[i];

                    for (int j = 0; j < listVertex.Count; j++)
                    {
                        if (listVertex[j].Value < vertexMinimum.Value &&
listVertex[j].Value != 0)
                            vertexMinimum = listVertex[j];
                    }

                    break;
                }
            }

            for (int i = 1; i < listVertex.Count; i++)
            {
                if (listVertex[i].Value > vertexMaximum.Value)
                    vertexMaximum = listVertex[i];
            }

            if (vertexMaximum != null)
            {
                Label label = new Label();

                label.Content = vertexMaximum.Value;
```

```
                label.Margin = new Thickness(vertexMaximum.X, vertexMaximum.Y
+ 20, 0, 0);
                label.VerticalAlignment = VerticalAlignment.Top;
                label.HorizontalAlignment = HorizontalAlignment.Left;
                label.Height = 40;
                label.Foreground = Brushes.Red;

                canvas.Children.Add(label);
                DrawLineRed(vertexMaximum);
            }

            if (vertexMinimum != null)
            {
                Label label1 = new Label();

                label1.Content = vertexMinimum.Value;
                label1.Margin = new Thickness(vertexMinimum.X,
vertexMinimum.Y + 20, 0, 0);
                label1.VerticalAlignment = VerticalAlignment.Top;
                label1.HorizontalAlignment = HorizontalAlignment.Left;
                label1.Height = 40;
                label1.Foreground = Brushes.Green;

                canvas.Children.Add(label1);
                DrawLineGreen(vertexMinimum);
            }
        }

        private void DrawLineGreen(Vertex minimumVertex)
        {
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (minimumVertex.IDParent == listVertex[i].ID)
                {
                    Line line = new Line();
                    line.X1 = listVertex[i].X;
                    line.Y1 = listVertex[i].Y + 20;
                    line.X2 = minimumVertex.X;
                    line.Y2 = minimumVertex.Y;
                    line.Stroke = Brushes.Green;
                    canvas.Children.Add(line);

                    if (listVertex[i].Probability == default)
                        break;
                    else
                        DrawLineGreen(listVertex[i]);

                }
            }
```

```
        }

        private void DrawLineRed(Vertex maximumVertexes)
        {
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (maximumVertexes.IDParent == listVertex[i].ID)
                {
                    Line l = new Line();
                    l.X1 = listVertex[i].X;
                    l.Y1 = listVertex[i].Y + 20;
                    l.X2 = maximumVertexes.X;
                    l.Y2 = maximumVertexes.Y;

                    l.Stroke = Brushes.Red;
                    canvas.Children.Add(l);

                    if (listVertex[i].Probability == default)
                        break;
                    else
                        DrawLineRed(listVertex[i]);
                }
            }
        }

        /// <summary>
        /// рисует дерево при начальной загрузке
        /// </summary>
        /// <param name="currentVertexes"></param>
        private void DrawRoot(Vertex currentVertexes)
        {
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (listVertex[i].IDParent == currentVertexes.ID &&
listVertex[i].Probability != default)
                {
                    AddNewVertexToTree(listVertex[i]);
                    DrawNewLine(listVertex[i], currentVertexes);
                    DrawRoot(listVertex[i]);
                }
            }
        }

        /// <summary>
        /// метод срабатывает при нажатии на вершину графа
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Button_Click(object sender, RoutedEventArgs e)
```

```
        {
            AddButton.DataContext = (Vertex)((Button)sender).DataContext;
            Vertex vertex = (Vertex)AddButton.DataContext;


        }

        /// <summary>
        /// метод для удаления вершины из дерева
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void DeleteButton_Click(object sender, RoutedEventArgs
e)
        {
            canvas.Children.Clear();

            Tree tree = new Tree();

            if (((Vertex)AddButton.DataContext).Probability != 0)
            {
                List<Vertex> listDelete =
DeleteVertexes((Vertex)AddButton.DataContext);
                listDelete.Add((Vertex)AddButton.DataContext);
                await tree.DeleteVertex(listDelete);
                listVertex = await tree.ShowListVertexes();
            }

            Button button = new Button();

            button.HorizontalAlignment = HorizontalAlignment.Left;
            button.VerticalAlignment = VerticalAlignment.Top;
            button.Margin = new Thickness(Width / 2 - 10, 50, Width / 2 - 10,
Height - 70);
            button.Background = new ImageBrush(new BitmapImage(new
Uri(pathToPlus)));

            BackButton.Background = new ImageBrush(new BitmapImage(new
Uri(pathToBack)));

            button.DataContext = _vertexFirst;
            button.Height = 20;
            button.Width = 20;
            button.Click += Button_Click;

            canvas.Children.Add(button);

            DrawRoot(_vertexFirst);
            CurrentBranchCost(_vertexFirst, 0);
```

```
            DrawDangerousMaximum();
        }

        /// <summary>
        /// метод для удаления вершин
        /// </summary>
        /// <param name="currentvertex"></param>
        /// <returns></returns>
        private List<Vertex> DeleteVertexes(Vertex currentVertex)
        {
            List<Vertex> listDelete = new List<Vertex>();

            for (int i = 0; i < listVertex.Count; i++)
            {
                if (listVertex[i].Probability != 0 && listVertex[i].IDParent
== currentVertex.ID)
                    listDelete.Add(listVertex[i]);
            }

            for (int i = 0; i < listVertex.Count; i++)
            {
                for (int j = 0; j < listDelete.Count; j++)
                {
                    if (listVertex[i].Probability != 0 &&
listVertex[i].IDParent == listDelete[j].ID)
                        listDelete.Add(listVertex[i]);
                }
            }

            return listDelete;
        }

        /// <summary>
        /// метод, который срабатывает при нажатии на кнопку Create Report,
        /// закрывает окно дерева, открывает окно отчёта
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void GoToReport_Click(object sender, RoutedEventArgs e)
        {
            ReportWindow reportWindow = new ReportWindow(risk, project,
center);
            Close();
            reportWindow.Show();
        }

        /// <summary>
        /// метод срабатывает при нажатии кнопки Back
        /// закрывается окно с деревом, снова открывается окно с графиком
```

```
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void BackButton_Click(object sender, RoutedEventArgs e)
        {
            AdministratorGraphic graphicWindow = new
AdministratorGraphic(project);
            Close();
            graphicWindow.Show();
        }
    }
}
```

1.3. Модуль ProjectManagerWindows

1.3.1. ChoiceWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace RiskApp
{
    /// <summary>
    /// Логика взаимодействия для ProjectChoise.xaml
    /// </summary>
    public partial class ChoiceWindow : Window
    {
        User user = null;
        bool flag = true;
        string path =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "back.jpg");

        public ChoiceWindow(User user)
        {
            this.user = user;
            InitializeComponent();
        }

        /// <summary>
        /// метод запускается при открытии окна
```

```
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void Window_Activated(object sender, EventArgs e)
        {
            if (flag)
            {
                BackButton.Background = new ImageBrush(new BitmapImage(new
Uri(path)));
                BackButton.Foreground = new ImageBrush(new BitmapImage(new
Uri(path)));

                ProjectActions projectActions = new ProjectActions();
                List<Project> listProjects = await
projectActions.ShowUsersProjects(user.Login);

                for (int i = 0; i < listProjects.Count; i++)
                    listBoxProjects.Items.Add(listProjects[i]);
            }
        }

        /// <summary>
        /// метод для элемента ListBox, отображающего проекты пользователя
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ListBoxProjects_MouseDoubleClick(object sender,
MouseButtonEventArgs e)
        {
            if (listBoxProjects.SelectedItem != null)
            {
                Project project = (Project)listBoxProjects.SelectedItem;
                ProjectManagerGraphic graphic = new
ProjectManagerGraphic(project,user);

                Close();
                graphic.Show();
            }
            else
                MessageBox.Show("You need to select project!");
        }
        private void BackButton_Click(object sender, RoutedEventArgs e)
        {
            MainWindow mainWindow = new MainWindow();

            Close();
            mainWindow.Show();
        }
    }
}
```

1.3.2. ProjectManagerGraphic.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace RiskApp
{
    /// <summary>
    /// Логика взаимодействия для GraphicForProjectManager.xaml
    /// </summary>
    public partial class ProjectManagerGraphic : Window
    {
        const int K = 100;
        const double radius = 250;

        User _user = null;

        string path =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "back.jpg");

        List<Risk> listAllRisks = null;
        List<string> listSource = null;
        List<Risk> listRisksSelected = null;

        bool flag = true;
        bool flag1 = true;

        Project _project = null;

        Point center = new Point(500, 50);
        Point mousePosition;
        Point currentCenter;
        public ProjectManagerGraphic(Project project, User user)
        {
            _project = project;
            _user=user;
```

```
            InitializeComponent();
        }

        /// <summary>
        /// метод запускается при открытии окна
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void Window_Activated(object sender, EventArgs e)
        {
            if (flag)
            {
                Label label = new Label();
                label.VerticalAlignment = VerticalAlignment.Top;
                label.HorizontalAlignment = HorizontalAlignment.Center;
                label.FontSize = 15;
                label.Margin = new Thickness(0, 25, 0, 0);
                label.Content = $"Risks' matrix for { _project.Name}";

                grid.Children.Add(label);
                BackButton.Background = new ImageBrush(new BitmapImage(new
Uri(path)));

                DatabaseActions databaseActions = new DatabaseActions();

                listRisksSelected = await
databaseActions.ShowRisks(_project);

                if (listRisksSelected == null)
                    listRisksSelected = new List<Risk>();

                RiskActions riskActions = new RiskActions();

                listAllRisks = await riskActions.ShowRisks();
                ComboBoxTypes.Items.Add("Общие риски");
                ComboBoxTypes.Text = "Common Risks";
                ComboBoxTypes.Items.Add(_project.Type);
                listSource = await riskActions.ShowSources();

                AddInSelected();

                for (int i = 0; i < listSource.Count; i++)
                    ComboBoxTypes.Items.Add(listSource[i]);

                DrawHyperbola();
                FindDangerousRisks();
                await SetOwners();
                flag = false;
            }
```

```
}


private async Task SetOwners()
{
    List<User> listUsers = await new UserActions().ShowUsers();

    for (int i = 0; i < listUsers.Count; i++)
    {
        OwnerCombobox.Items.Add(listUsers[i]);
        OwnerNewCombobox.Items.Add(listUsers[i]);
    }
}

/// <summary>
/// метод добавляет во вкладку выбранные риски
/// и убирает их из выбора
/// </summary>
private void AddInSelected()
{
    if (listRisksSelected != null)
    {
        for (int i = 0; i < listRisksSelected.Count; i++)
        {
            if (listRisksSelected[i].Status == 1)
                listSelected.Items.Add(listRisksSelected[i]);
            else
            {
                if (listRisksSelected[i].Status == 0)
                    listNewRisks.Items.Add(listRisksSelected[i]);
                else
                    listUnSelected.Items.Add(listRisksSelected[i]);
            }
        }
    }
}

/// <summary>
/// метод проверяет, не нахоядтся ли данные элементы уже в проекте
/// </summary>
private void CheckIfInProject()
{
    if (listRisksSelected != null)
    {
        for (int i = 0; i < listAllRisks.Count; i++)
        {
            for (int j = 0; j < listRisksSelected.Count; j++)
            {
```

```
                        if (listAllRisks[i].RiskName ==
listRisksSelected[j].RiskName)
                            listAllRisks.RemoveAt(i);
                    }
                }
            }
        }

        /// <summary>
        /// метод отрисовывает гиперболу и точки
        /// </summary>
        private void DrawHyperbola()
        {
            canvas.Children.Clear();

            for (int i = 0; i < K - 1; i++)
            {
                double oldX = center.X - radius + radius / K * i;
                double currentX = center.X - radius + radius / K * (i + 1);

                Line line = new Line();

                line.X1 = oldX;
                line.X2 = currentX;
                line.Y1 = FindYCoordinate(oldX, radius, center);
                line.Y2 = FindYCoordinate(currentX, radius, center);
                line.Stroke = Brushes.Black;

                canvas.Children.Add(line);
            }

            for (int i = 0; i < listSelected.Items.Count; i++)
            {
                if (((Risk)listSelected.Items[i]).Probability != default &&
                    ((Risk)listSelected.Items[i]).Influence != default&&
((Risk)listSelected.Items[i]).Status == 1)
                {
                    ((Risk)listSelected.Items[i]).point.X = 425 *
((Risk)listSelected.Items[i]).Probability + 75;
                    ((Risk)listSelected.Items[i]).point.Y = -350 *
((Risk)listSelected.Items[i]).Influence + 400;

                    Ellipse ellipse = new Ellipse();
                    ellipse.Height = 10;
                    ellipse.Width = 10;
                    ellipse.StrokeThickness = 2;

                    if (Math.Sqrt((((Risk)listSelected.Items[i]).point.X -
center.X) * (((Risk)listSelected.Items[i]).point.X - center.X) +
```

```
                        (((Risk)listSelected.Items[i]).point.Y - center.Y) *
(((Risk)listSelected.Items[i]).point.Y - center.Y)) < radius)
                    {
                        ellipse.Stroke = Brushes.Red;
                        ellipse.Fill = Brushes.Red;
                    }
                    else
                    {
                        ellipse.Stroke = Brushes.Green;
                        ellipse.Fill = Brushes.Green;
                    }

                    ellipse.VerticalAlignment = VerticalAlignment.Top;
                    ellipse.HorizontalAlignment = HorizontalAlignment.Left;
                    ellipse.Margin = new
Thickness(((Risk)listSelected.Items[i]).point.X,
((Risk)listSelected.Items[i]).point.Y, 0, 0);

                    canvas.Children.Add(ellipse);
                }
            }
        }

        static public double FindYCoordinate(double x, double radius, Point
center)
        {
            double a = 1;
            double b = -2 * center.Y;
            double c = -radius * radius + (x - center.X) * (x - center.X) +
center.Y * center.Y;
            double d = (b * b - 4 * a * c);

            if (d < 0)
                throw new Exception("There's no such point!");

            return (-b + Math.Sqrt(d)) / 2 * a;
        }

        /// <summary>
        /// метод срабатывает при нажатии кнопки мыши
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Canvas_MouseDown(object sender, MouseButtonEventArgs e)
        {
            if (e.GetPosition(null).X < 540 && e.GetPosition(null).Y < 450)
            {
                mousePosition = e.GetPosition(null);
                currentCenter = center;
```

```
            }
        }

        private void ComboBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            listRisks.Items.Clear();
            CheckIfInProject();
            if (flag)
            {
            }
            if (ComboBoxTypes.SelectedItem.ToString() == "Общие риски")
            {
                for (int i = 0; i < listAllRisks.Count; i++)
                {
                    if (listAllRisks[i].Type == "default")
                        listRisks.Items.Add(listAllRisks[i]);
                }
            }
            else
            {
                if (ComboBoxTypes.SelectedItem.ToString() == _project.Type)
                {
                    for (int i = 0; i < listAllRisks.Count; i++)
                    {
                        if (listAllRisks[i].Type == _project.Type)
                            listRisks.Items.Add(listAllRisks[i]);
                    }
                }
                else
                {
                    for (int i = 0; i < listSource.Count; i++)
                    {
                        if (ComboBoxTypes.SelectedItem.ToString() ==
listSource[i])
                        {
                            for (int j = 0; j < listAllRisks.Count; j++)
                            {
                                if ((listAllRisks[j].Type == _project.Type ||
listAllRisks[j].Type == "default") && listAllRisks[j].Source ==
listSource[i])
                                    listRisks.Items.Add(listAllRisks[j]);
                            }
                        }
                    }
                }
            }
        }
```

```
        /// <summary>
        /// метод для комбобокса
        /// </summary>
        private void ChangeSelected()
        {
            if (ComboBoxTypes.SelectedItem != null)
                ComboBoxTypes.SelectedItem = "Общие риски";

            listRisks.Items.Clear();
            CheckIfInProject();
            if (flag) { }

            if (ComboBoxTypes.SelectedItem.ToString() == "Общие риски")
            {
                for (int i = 0; i < listAllRisks.Count; i++)
                {
                    if (listAllRisks[i].Type == "default")
                        listRisks.Items.Add(listAllRisks[i]);
                }
            }
            else
            {
                if (ComboBoxTypes.SelectedItem.ToString() == _project.Type)
                {
                    for (int i = 0; i < listAllRisks.Count; i++)
                    {
                        if (listAllRisks[i].Type == _project.Type)
                            listRisks.Items.Add(listAllRisks[i]);
                    }
                }
                else
                {
                    for (int i = 0; i < listSource.Count; i++)
                    {
                        if (ComboBoxTypes.SelectedItem.ToString() ==
listSource[i])
                        {
                            for (int j = 0; j < listAllRisks.Count; j++)
                            {
                                if ((listAllRisks[j].Type == _project.Type ||
listAllRisks[j].Type == "default")
                                    && listAllRisks[j].Source ==
listSource[i])
                                    listRisks.Items.Add(listAllRisks[j]);
                            }
                        }
                    }
                }
            }
```

```
        }

        /// <summary>
        /// метод для проверки существования риска в списке
        /// </summary>
        /// <param name="checkrisk"></param>
        /// <returns></returns>
        private bool CheckIfSelected(Risk risk)
        {
            if (listSelected.Items.Count != 0)
            {
                for (int i = 0; i < listSelected.Items.Count; i++)
                {
                    if (risk.ID == ((Risk)listSelected.Items[i]).ID)
                        return false;
                }

                return true;
            }

            return true;
        }

        /// <summary>
        /// метод для установления вероятности и влияния риска при нажатии на
кнопку
        /// и внесении данных в бд
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void SetUpButton_Click(object sender, RoutedEventArgs
e)
        {
            try
            {
                if (Double.Parse(ParseLine(InfluenceTextbox.Text)) == default
|| Double.Parse(ParseLine(ProbabilityTextbox.Text)) == default)
                    throw new ArgumentException("The values of probability
and influence must lay in the interval (0,1)!");

                if (OwnerCombobox.SelectedItem == null)
                    throw new NullReferenceException("You have to choose the
user you want to assign the risk to!");

                ((Risk)listSelected.SelectedItem).Influence =
double.Parse(ParseLine(InfluenceTextbox.Text));
                ((Risk)listSelected.SelectedItem).Probability =
double.Parse(ParseLine(ProbabilityTextbox.Text));
```

```
                DatabaseActions databaseActions = new DatabaseActions();
                await
        databaseActions.ChangeRisk((Risk)listSelected.SelectedItem,
        (User)OwnerCombobox.SelectedItem);

                listSelected.Items.Clear();
                listRisksSelected = await
        databaseActions.ShowRisks(_project);

                for (int i = 0; i < listRisksSelected.Count; i++)
                {
                    if (listRisksSelected[i].Status == 1)
                        listSelected.Items.Add(listRisksSelected[i]);
                }

                DrawHyperbola();
            }
            catch (NullReferenceException ex)
            {
                MessageBox.Show(ex.Message, "Exception");

            }
            catch (ArgumentException ex)
            {
                MessageBox.Show(ex.Message, "Exception");

            }
            catch (Exception)
            {
                MessageBox.Show("Something's not right!");
            }
        }

        private string ParseLine(string line)
        {
            string result = "";

            for (int i = 0; i < line.Length; i++)
            {
                if (line[i] == '.')
                    result += ',';
                else
                    result += line[i];
            }

            return result;
        }

        /// <summary>
```

```
        /// метод, считывающий введенные в текстбоксы значения
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void SelectedRisks_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            if (listSelected.SelectedItem != null)
            {
                InfluenceTextbox.Text =
((Risk)listSelected.SelectedItem).Influence.ToString();
                ProbabilityTextbox.Text =
((Risk)listSelected.SelectedItem).Probability.ToString();
            }
        }

        /// <summary>
        /// перемещие гиперболы во время движения мыши
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Canvas_MouseMove(object sender, MouseEventArgs e)
        {
            if (e.GetPosition(null).X < 540 && e.GetPosition(null).Y < 450 &&
e.LeftButton == MouseButtonState.Pressed)
            {
                center.X = currentCenter.X - mousePosition.X +
e.GetPosition(null).X;
                center.Y = currentCenter.Y + (mousePosition.X -
e.GetPosition(null).X) / 1.2;

                if (center.X > 650 || center.Y < -100)
                {
                    center.Y = -100;
                    center.X = 650;
                }

                if (center.Y > 230 || center.X < 250)
                {
                    center.Y = 230;
                    center.X = 250;
                }

                DrawHyperbola();
                FindDangerousRisks();
            }
        }

        /// <summary>
```

```
        /// метод ищет опасные риски среди представленных на координатной
плоскости
        /// </summary>
        private void FindDangerousRisks()
        {
            listDangerous.Items.Clear();

            if (listSelected.Items.Count != 0)
            {
                for (int i = 0; i < listSelected.Items.Count; i++)
                {
                    if (Math.Sqrt((((Risk)listSelected.Items[i]).point.X -
center.X) * (((Risk)listSelected.Items[i]).point.X - center.X) +
                        (((Risk)listSelected.Items[i]).point.Y - center.Y) *
(((Risk)listSelected.Items[i]).point.Y - center.Y)) < radius)
                    {
                        listDangerous.Items.Add((Risk)listSelected.Items[i]);
                    }
                }
            }
        }

        private void Canvas_MouseRightButtonDown(object sender,
MouseButtonEventArgs e)
        {
            if (listSelected.Items.Count != 0)
            {
                List<Risk> click = new List<Risk>();
                for (int i = 0; i < listSelected.Items.Count; i++)
                {
                    if (Math.Abs(((Risk)listSelected.Items[i]).point.X -
e.GetPosition(null).X) <= 10
                        && Math.Abs(((Risk)listSelected.Items[i]).point.Y -
e.GetPosition(null).Y) <= 10)
                    {
                        click.Add((Risk)listSelected.Items[i]);
                    }
                }
                if (click.Count != 0)
                {
                    MessageBox.Show(CreateLine(click), "INformation about
Selected Risks");
                }
            }
        }

        /// <summary>
        /// метод, возвращающий строку для вывода информации при нажатии
праой кнопки мыши
```

```
        /// </summary>
        /// <param name="click"></param>
        /// <returns></returns>
        private string CreateLine(List<Risk> listClicks)
        {
            string line = "";

            for (int i = 0; i < listClicks.Count; i++)
                line += $"RiskName: {listClicks[i].RiskName}\nSource:
{listClicks[i].Source}\nType: {listClicks[i].Type}\nDescription:
{listClicks[i].Solution}";

            return line;
        }
        /// <summary>
        /// метод для удаления риска при нажатии кнопки
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void DeleteButton_Click(object sender, RoutedEventArgs
e)
        {
            DatabaseActions databaseActions = new DatabaseActions();
            Risk risk = (Risk)((Button)sender).DataContext;

            listRisksSelected.Remove(risk);

            if (listRisksSelected == null)
                listRisksSelected = new List<Risk>();

            listSelected.Items.Remove(risk);
            risk.Status = 2;
            SearchForCurrentRisk(risk);

            await databaseActions.ChangeRisk(risk);
            listUnSelected.Items.Add(risk);

            DrawHyperbola();
        }

        /// <summary>
        /// метод для поиска риска в списке выбранных
        /// </summary>
        /// <param name="risk"></param>
        private void SearchForCurrentRisk(Risk risk)
        {
            for (int i = 0; i < listRisksSelected.Count; i++)
            {
                if (risk.ID == listRisksSelected[i].ID)
```

```
                {
                    listRisksSelected[i].Status = risk.Status;
                    listRisksSelected[i].IdUser = risk.IdUser;
                    listRisksSelected[i].OwnerLogin = risk.OwnerLogin;
                }
            }
        }

        /// <summary>
        /// метод, который добавляет риск в список выбранных
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void AddButton_Click(object sender, RoutedEventArgs e)
        {
            if (CheckIfSelected(((Risk)((Button)sender).DataContext)))
            {
                RiskSettingsWindow window = new RiskSettingsWindow();
                Risk risk = (Risk)(((Button)sender).DataContext);

                if (window.ShowDialog() == true)
                {
                    try
                    {
                        risk.Influence = window.Influence;
                        risk.Probability = window.Probability;

                        if (window.Influence == default(double))
                            ((Risk)((Button)sender).DataContext).Status = 0;
                        else
                            ((Risk)((Button)sender).DataContext).Status = 1;

                        DatabaseActions databaseActions = new
DatabaseActions();

                        if (window.Owner == null)
                            await databaseActions.AddRisk(_project.Name,
risk);
                        else
                        {
                            risk.OwnerLogin = window.Owner.Login;
                            risk.IdUser = window.Owner.ID;

                            await databaseActions.AddRisk(_project.Name,
risk, window.Owner);
                            SearchForCurrentRisk(risk);
                        }

                        listRisksSelected.Add(risk);
```

```
                        ChangeSelected();
                    }
                    catch
                    {
                        MessageBox.Show("Something went wrong");
                    }
                }

                listSelected.Items.Clear();
                listNewRisks.Items.Clear();
                for (int i = 0; i < listRisksSelected.Count; i++)
                {
                    if (listRisksSelected[i].Status == 1)
                        listSelected.Items.Add(listRisksSelected[i]);

                    if (listRisksSelected[i].Status == 0)
                        listNewRisks.Items.Add(listRisksSelected[i]);

                }

                DrawHyperbola();
                listAllRisks.Remove(risk);
                ComboBoxTypes.SelectedItem = ComboBoxTypes.SelectedItem;
            }
            else
            {
                MessageBox.Show("This element has already been selected!");
            }
        }
        private void DangerousRisks_MouseDoubleClick(object sender,
MouseButtonEventArgs e)
        {
            if (flag1 && listDangerous.SelectedItem != null)
            {
                ProjectManagerTree tree = new
ProjectManagerTree((Risk)listDangerous.SelectedItem, _project, _user,
center);
                Close();
                tree.Show();
                flag1 = false;
            }
        }

        private void BackButton_Click(object sender, RoutedEventArgs e)
        {
            ChoiceWindow choice = new ChoiceWindow(_user);
            Close();
            choice.Show();
        }
```

```
        private async void AddToActive_Click(object sender, RoutedEventArgs
e)
        {
            Risk risk = (Risk)((Button)sender).DataContext;

            listUnSelected.Items.Remove(risk);
            listRisksSelected.Add(risk);
            listSelected.Items.Add(risk);
            risk.Status = 1;
            SearchForCurrentRisk(risk);

            DatabaseActions databaseActions = new DatabaseActions();

            await databaseActions.ChangeRisk(risk);
            DrawHyperbola();
        }

        private async void SetUpNewButton_Click(object sender,
RoutedEventArgs e)
        {
            try
            {
                if (listNewRisks.SelectedItems != null &&
                    Double.Parse(ParseLine(NewInfluenceTextbox.Text)) !=
default &&
                    Double.Parse(ParseLine(NewProbabilityTextbox.Text)) !=
default
                    && OwnerNewCombobox.SelectedItem != null)
                {
                    ((Risk)listNewRisks.SelectedItem).Influence =
double.Parse(ParseLine(NewInfluenceTextbox.Text));
                    ((Risk)listNewRisks.SelectedItem).Probability =
double.Parse(ParseLine(NewProbabilityTextbox.Text));
                    ((Risk)listNewRisks.SelectedItem).Status = 1;
                    SearchForCurrentRisk(((Risk)listNewRisks.SelectedItem));

                    DatabaseActions databaseActions = new DatabaseActions();

                    await
databaseActions.ChangeRisk((Risk)listNewRisks.SelectedItem,
(User)OwnerNewCombobox.SelectedItem);
                    listNewRisks.Items.Clear();
                    listSelected.Items.Clear();
                    listRisksSelected = await
databaseActions.ShowRisks(_project);

                    for (int i = 0; i < listRisksSelected.Count; i++)
                    {
```

```
                        if (listRisksSelected[i].Status == 0)
                            listNewRisks.Items.Add(listRisksSelected[i]);

                        if (listRisksSelected[i].Status == 1)
                            listSelected.Items.Add(listRisksSelected[i]);
                    }

                    DrawHyperbola();
                }
                else
                {
                    MessageBox.Show("Something went wrong!");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        /// <summary>
        /// метод устанавливает нулевые значения для нового риска
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void NewRisksSelect_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            if (listNewRisks.SelectedItem != null)
            {
                NewInfluenceTextbox.Text = "0";
                NewProbabilityTextbox.Text = "0";
            }
        }
    }
}
```

1.3.3. ProjectManagerTree.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
```

```
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace RiskApp
{
    public partial class ProjectManagerTree : Window
    {
        bool flag = true;
        Risk risk = null;

        Project project;

        double Widht;
        new double Height;
        Vertex _first;

        List<double> listValues = new List<double>();
        List<Vertex> listVertex = new List<Vertex>();

        string path =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "back.jpg");
        string pathToPlus =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "plus.png");

        User user = null;
        Point center;

        public ProjectManagerTree(Risk risk, Project project,User user,Point
center)
        {
            this.center = center;
            this.user = user;
            this.risk = risk;
            this.project = project;
            InitializeComponent();
        }

        private async void AddNewVertexButton_Click(object sender,
RoutedEventArgs e)
        {
            try
            {
                if (((Button)sender).DataContext == null)
                    throw new Exception("You need to choose vertex!");

                if (DescriptionTextbox.Text == "")
```

```
                        throw new Exception("You have to put a
description!");

                if (!double.TryParse(CostTextbox.Text, out double d) || d <=
0)
                     throw new Exception("The value of cost must be more than
0!");

                if (!double.TryParse(ProbabilityTextbox.Text, out double d1)
|| d1 <= 0 || d1 >= 1)
                    throw new Exception("The value of probability must be in
the interval (0;1)!");

                Vertex parentVertex = ((Vertex)((Button)sender).DataContext);

                int row = 1;
                GetCurrenRow(parentVertex, ref row);

                if (parentVertex.Probability != 0)
                    row++;

                if (row >= 4)
                    throw new ArgumentException("The tree branch cannot be
more than 4!");

                Vertex vertex;
                string line = $"{(parentVertex.X - Widht / (2 * Math.Pow(4,
row))):f3}";

                if (CheckIfAlreadyExists(double.Parse(line)))
                    vertex = new Vertex(parentVertex.X - Widht / (2 *
Math.Pow(4, row)), parentVertex.Y + 50, double.Parse(CostTextbox.Text),
double.Parse(ProbabilityTextbox.Text), parentVertex.ID,
DescriptionTextbox.Text);
                else
                {
                    line = $"{(parentVertex.X + Widht / (2 * Math.Pow(4,
row))):f3}";

                    if (CheckIfAlreadyExists(double.Parse(line)))
                        vertex = new Vertex( parentVertex.X + Widht / (2 *
Math.Pow(4, row)), parentVertex.Y + 50, double.Parse(CostTextbox.Text),
double.Parse(ProbabilityTextbox.Text), parentVertex.ID,
DescriptionTextbox.Text);
                    else
                    {
                        line = $"{(parentVertex.X - 3 * Widht / (2 *
Math.Pow(4, row))):f3}";
                        if (CheckIfAlreadyExists(double.Parse(line)))
```

```
                        {
                            vertex = new Vertex(parentVertex.X - 3 * Widht /
(2 * Math.Pow(4, row)), parentVertex.Y + 50, double.Parse(CostTextbox.Text),
double.Parse(ProbabilityTextbox.Text), parentVertex.ID,
DescriptionTextbox.Text);
                        }
                        else
                        {
                            line = $"{(parentVertex.X + 3 * Widht / (2 *
Math.Pow(4, row))):f3}";

                            if (CheckIfAlreadyExists(double.Parse(line)))
                                vertex = new Vertex(parentVertex.X + 3 *
Widht / (2 * Math.Pow(4, row)), parentVertex.Y + 50,
double.Parse(CostTextbox.Text), double.Parse(ProbabilityTextbox.Text),
parentVertex.ID, DescriptionTextbox.Text);
                            else
                                throw new Exception("The amount of children
cannot be more than 4!");
                        }
                    }
                }

                Tree tree = new Tree();

                await tree.AddVertex(parentVertex.ID, vertex);
                vertex = await tree.ShowVertex(vertex);
                listVertex.Add(vertex);

                RefreshTree(_first);
                Clear();
                CurrentBranchCost(_first, 0);
                DrawDangerousMaximum();
            }
            catch (ArgumentException ex)
            {
                MessageBox.Show(ex.Message);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
        /// <summary>
        /// метод рисует линии соединяющие точки графа
        /// </summary>
        /// <param name="newver"></param>
        /// <param name="parent"></param>
        private void DrawNewLine(Vertex vertex, Vertex parentVertex)
```

```
        {
            Line line = new Line();
            line.X1 = parentVertex.X;
            line.Y1 = parentVertex.Y + 20;
            line.X2 = vertex.X;
            line.Y2 = vertex.Y;
            line.Stroke = Brushes.Black;
            canvas.Children.Add(line);
        }

        /// <summary>
        /// метод рисует вершины
        /// </summary>
        /// <param name="current"></param>
        private void DrawNewVertex(Vertex current)
        {
            Button button = new Button();
            button.DataContext = current;
            button.HorizontalAlignment = HorizontalAlignment.Left;
            button.VerticalAlignment = VerticalAlignment.Top;
            button.Margin = new Thickness(current.X - 10, current.Y, Widht -
current.X - 10, Height - current.Y - 20);
            button.Height = 20;
            button.Width = 20;
            button.Background = new ImageBrush(new BitmapImage(new
Uri(pathToPlus)));
            button.Click += But_Click;
            canvas.Children.Add(button);
        }

        /// <summary>
        /// метод для очистки значений вершин
        /// </summary>
        private void Clear()
        {
            for (int i = 0; i < listVertex.Count; i++)
                listVertex[i].Value = default;
        }

        private void RefreshTree(Vertex cur)
        {
            canvas.Children.Clear();
            Button but = new Button();
            but.HorizontalAlignment = HorizontalAlignment.Left;
            but.VerticalAlignment = VerticalAlignment.Top;
            but.Margin = new Thickness(Widht / 2 - 10, 50, Widht / 2 - 10,
Height - 70);
            but.Background = new ImageBrush(new BitmapImage(new
Uri(pathToPlus)));
```

```
            Back.Background = new ImageBrush(new BitmapImage(new Uri(path)));
            but.DataContext = _first;
            but.Height = 20;
            but.Width = 20;
            but.Click += But_Click;
            canvas.Children.Add(but);
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (listVertex[i].IDParent == cur.ID &&
listVertex[i].Probability != default(double))
                {
                    DrawNewVertex(listVertex[i]);
                    DrawNewLine(listVertex[i], cur);
                    DrawVertexesByRoot(listVertex[i]);
                }
            }
        }

        /// <summary>
        /// метод для проверки на наличие вершины
        /// </summary>
        /// <param name="x"></param>
        /// <returns></returns>
        private bool CheckIfAlreadyExists(double x)
        {
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (x == listVertex[i].X)
                    return false;
            }

            return true;
        }

        /// <summary>
        /// метод для расчёта стоимости ветки
        /// </summary>
        /// <param name="currentVertex"></param>
        /// <param name="c"></param>
        private void CurrentBranchCost(Vertex currentVertex, double c)
        {
            double cost = c;

            for (int i = 0; i < listVertex.Count; i++)
            {
                if (currentVertex.ID == listVertex[i].IDParent &&
listVertex[i].Probability != default)
                {
                    cost += listVertex[i].Probability * listVertex[i].Cost;
```

```
                    CurrentBranchCost(listVertex[i], cost);
                    cost -= listVertex[i].Probability * listVertex[i].Cost;
                }
            }

            if (CheckIfChildExists(currentVertex))
                currentVertex.Value = cost;
        }

        /// <summary>
        /// метод для проверки наличия ребенка у вершины
        /// </summary>
        /// <param name="current"></param>
        /// <returns></returns>
        private bool CheckIfChildExists(Vertex current)
        {
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (current.ID == listVertex[i].IDParent &&
listVertex[i].Probability != default)
                    return false;
            }

            return true;
        }


        /// <summary>
        /// метод выдает ряд дерева
        /// </summary>
        /// <param name="parentVertex"></param>
        /// <param name="k"></param>
        private void GetCurrenRow(Vertex parentVertex, ref int k)
        {
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (parentVertex.IDParent == listVertex[i].ID &&
listVertex[i].Probability == default)
                    return;

                if (parentVertex.IDParent == listVertex[i].ID &&
listVertex[i].Probability != default)
                {
                    k++;
                    GetCurrenRow(listVertex[i], ref k);
                }
            }
        }
```

```
        private async void Window_Activated(object sender, EventArgs e)
        {
            if (flag)
            {
                try
                {
                    Widht = canvas.ActualWidth;
                    Height = canvas.ActualHeight;

                    Label label = new Label();
                    label.HorizontalAlignment = HorizontalAlignment.Center;
                    label.VerticalAlignment = VerticalAlignment.Top;
                    label.Content = risk.RiskName;

                    grid.Children.Add(label);

                    Tree tree = new Tree();
                    Button button = new Button();

                    button.HorizontalAlignment = HorizontalAlignment.Left;
                    button.VerticalAlignment = VerticalAlignment.Top;
                    button.Margin = new Thickness(Widht / 2 - 10, 50, Widht /
2 - 10, Height - 70);
                    button.Background = new ImageBrush(new BitmapImage(new
Uri(pathToPlus)));

                    Back.Background = new ImageBrush(new BitmapImage(new
Uri(path)));
                    Back.Foreground = new ImageBrush(new BitmapImage(new
Uri(path)));

                    if (!await tree.CheckIfExistInDataBase(risk.ID))
                    {
                        _first = new Vertex(Widht / 2, 50, risk.ID,
risk.RiskName);
                        await tree.AddVertex(risk.ID, _first);

                        _first = await tree.ShowVertex(risk.ID);
                        button.DataContext = _first;
                    }
                    else
                    {
                        _first = await tree.ShowVertex(risk.ID);
                        button.DataContext = _first;
                    }

                    button.Height = 20;
                    button.Width = 20;
```

```
                    button.Click += But_Click;

                    canvas.Children.Add(button);
                    listVertex = await tree.ShowListVertexes();

                    DrawVertexesByRoot(_first);
                    CurrentBranchCost(_first, 0);
                    DrawDangerousMaximum();

                    flag = false;
                }
                catch (NullReferenceException ex)
                {
                    MessageBox.Show(ex.Message);
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
        }

        /// <summary>
        /// метод выделяет самый опасный риск
        /// </summary>
        private void DrawDangerousMaximum()
        {
            Vertex Maximum = listVertex[0];
            Vertex Minimum = null;

            for (int i = 0; i < listVertex.Count; i++)
            {
                if (listVertex[i].Value != 0)
                {
                    Minimum = listVertex[i];

                    for (int j = 0; j < listVertex.Count; j++)
                        if (listVertex[j].Value < Minimum.Value &&
listVertex[j].Value != 0)
                            Minimum = listVertex[j];

                    break;
                }

            }

            for (int i = 1; i < listVertex.Count; i++)
            {
                if (listVertex[i].Value > Maximum.Value)
```

```
                    Maximum = listVertex[i];
                }

                if (Maximum != null)
                {
                    Label label = new Label();
                    label.Content = Maximum.Value;
                    label.Margin = new Thickness(Maximum.X, Maximum.Y + 20, 0,
0);
                    label.VerticalAlignment = VerticalAlignment.Top;
                    label.HorizontalAlignment = HorizontalAlignment.Left;
                    label.Height = 40;
                    label.Foreground = Brushes.Red;

                    canvas.Children.Add(label);
                    DrawLineRed(Maximum);
                }

                if (Minimum != null)
                {
                    Label l1 = new Label();
                    l1.Content = Minimum.Value;
                    l1.Margin = new Thickness(Minimum.X, Minimum.Y + 20, 0, 0);
                    l1.VerticalAlignment = VerticalAlignment.Top;
                    l1.HorizontalAlignment = HorizontalAlignment.Left;
                    l1.Height = 40;
                    l1.Foreground = Brushes.Green;
                    canvas.Children.Add(l1);
                    DrawLineGreen(Minimum);
                }
            }

            private void DrawLineGreen(Vertex min)
            {
                for (int i = 0; i < listVertex.Count; i++)
                {
                    if (min.IDParent == listVertex[i].ID)
                    {
                        Line line = new Line();

                        line.X1 = listVertex[i].X;
                        line.Y1 = listVertex[i].Y + 20;
                        line.X2 = min.X;
                        line.Y2 = min.Y;
                        line.Stroke = Brushes.Green;
                        canvas.Children.Add(line);

                        if (listVertex[i].Probability == default)
                            break;
```

```
                    else
                        DrawLineGreen(listVertex[i]);

                }
            }
        }

        private void DrawLineRed(Vertex max)
        {
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (max.IDParent == listVertex[i].ID)
                {
                    Line line = new Line();

                    line.X1 = listVertex[i].X;
                    line.Y1 = listVertex[i].Y + 20;
                    line.X2 = max.X;
                    line.Y2 = max.Y;

                    line.Stroke = Brushes.Red;
                    canvas.Children.Add(line);

                    if (listVertex[i].Probability == default)
                        break;
                    else
                        DrawLineRed(listVertex[i]);
                }
            }
        }

        /// <summary>
        /// рисует дерево при начальной загрузке
        /// </summary>
        /// <param name="current"></param>
        private void DrawVertexesByRoot(Vertex current)
        {
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (listVertex[i].IDParent == current.ID &&
listVertex[i].Probability != default)
                {
                    DrawNewVertex(listVertex[i]);
                    DrawNewLine(listVertex[i], current);
                    DrawVertexesByRoot(listVertex[i]);
                }
            }
        }
```

```
        /// <summary>
        /// нажатие на вершину графа
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void But_Click(object sender, RoutedEventArgs e)
        {
            AddButton.DataContext = (Vertex)((Button)sender).DataContext;
            Vertex vertex = (Vertex)AddButton.DataContext;
        }

        private void BackButton_Click(object sender, RoutedEventArgs e)
        {
            ProjectManagerGraphic graphic = new
ProjectManagerGraphic(project,user);
            Close();
            graphic.Show();
        }

        private async void DeleteButton_Click(object sender, RoutedEventArgs
e)
        {
            canvas.Children.Clear();
            Tree tree = new Tree();

            if (((Vertex)AddButton.DataContext).Probability != 0)
            {
                List<Vertex> list =
DeleteVertexes((Vertex)AddButton.DataContext);
                list.Add((Vertex)AddButton.DataContext);

                await tree.DeleteVertex(list);
                listVertex = await tree.ShowListVertexes();
            }

            Button buttonAdd = new Button();

            buttonAdd.HorizontalAlignment = HorizontalAlignment.Left;
            buttonAdd.VerticalAlignment = VerticalAlignment.Top;
            buttonAdd.Margin = new Thickness(Widht / 2 - 10, 50, Widht / 2 -
10, Height - 70);
            buttonAdd.Background = new ImageBrush(new BitmapImage(new
Uri(pathToPlus)));
            Back.Background = new ImageBrush(new BitmapImage(new Uri(path)));
            buttonAdd.DataContext = _first;
            buttonAdd.Height = 20;
            buttonAdd.Width = 20;
            buttonAdd.Click += But_Click;
```

```
            canvas.Children.Add(buttonAdd);
            DrawVertexesByRoot(_first);
            CurrentBranchCost(_first, 0);
            DrawDangerousMaximum();
        }

        /// <summary>
        /// метод для удаления вершины из списка
        /// </summary>
        /// <param name="currentVertex"></param>
        /// <returns></returns>
        private List<Vertex> DeleteVertexes(Vertex currentVertex)
        {
            List<Vertex> listDelete = new List<Vertex>();

            for (int i = 0; i < listVertex.Count; i++)
            {
                if (listVertex[i].Probability != 0 && listVertex[i].IDParent
== currentVertex.ID)
                    listDelete.Add(listVertex[i]);
            }
            for (int i = 0; i < listVertex.Count; i++)
            {
                for (int j = 0; j < listDelete.Count; j++)
                {
                    if (listVertex[i].Probability != 0 &&
listVertex[i].IDParent == listDelete[j].ID)
                        listDelete.Add(listVertex[i]);
                }
            }

            return listDelete;
        }

        private void GoReportButton_Click(object sender, RoutedEventArgs e)
        {
            ReportWindow window = new ReportWindow(risk, project, center);
            Close();
            window.Show();
        }
    }
}
```

1.4. Модуль RiskManagerWindows

1.4.1. RiskManagerGraphic.xaml.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace RiskApp
{
    public partial class RiskManagerGraphic : Window
    {
        const int C = 100;
        const double radius = 250;

        bool flag = true;
        bool flag1 = true;

        Project project = null;
        User user = null;

        Point mousePosition;
        Point currentCenter;
        Point center = new Point(500, 50);

        List<Risk> listSelected = null;

        string path =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "back.jpg");

        public RiskManagerGraphic(Project project, User user)
        {
            this.user=user;
            this.project = project;

            InitializeComponent();
        }
        /// <summary>
        /// метод при открытие окна
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void Window_Activated(object sender, EventArgs e)
        {
```

```
            if (flag)
            {
                Label label = new Label();

                DatabaseActions databaseActions = new DatabaseActions();

                label.VerticalAlignment = VerticalAlignment.Top;
                label.HorizontalAlignment = HorizontalAlignment.Center;
                label.FontSize = 15;
                label.Margin = new Thickness(0, 25, 0, 0);
                label.Content = $"Risks Matrix for { project.Name}";

                grid.Children.Add(label);
                BackButton.Background = new ImageBrush(new BitmapImage(new
Uri(path)));

                listSelected = await databaseActions.ShowRisks(project);

                if (listSelected == null)
                    listSelected = new List<Risk>();

                AddToSelected();
                DrawGraphic();
                SearchForDangerousRisks();

                flag = false;
            }
        }

        /// <summary>
        /// метод для добавления риска в список активных
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void AddToActiveRisks_Click(object sender,
RoutedEventArgs e)
        {
            Risk risk = (Risk)((Button)sender).DataContext;

            listRisksNonselected.Items.Remove(risk);
            listSelected.Add(risk);
            listRisksSelected.Items.Add(risk);

            risk.Status = 1;
            SearchForCurrentRisk(risk);

            DatabaseActions databaseActions = new DatabaseActions();

            await databaseActions.ChangeRisk(risk);
```

```
            DrawGraphic();
        }

        private async void SetUpNewButton_Click(object sender,
RoutedEventArgs e)
        {
            try
            {
                if (listNewRisks.SelectedItems != null &&
                    Double.Parse(ParseLine(TextboxInfluenceNew.Text)) !=
default &&
                    Double.Parse(ParseLine(TextboxProbabilityNew.Text)) !=
default
                    && UserNewCombobox.SelectedItem != null)
                {
                    ((Risk)listNewRisks.SelectedItem).Influence =
double.Parse(ParseLine(InfluenceTextbox.Text));
                    ((Risk)listNewRisks.SelectedItem).Probability =
double.Parse(ParseLine(ProbabilityTextbox.Text));
                    ((Risk)listNewRisks.SelectedItem).Status = 1;

                    SearchForCurrentRisk(((Risk)listNewRisks.SelectedItem));

                    DatabaseActions databaseActions = new DatabaseActions();

                    await
databaseActions.ChangeRisk((Risk)listNewRisks.SelectedItem,
(User)UserNewCombobox.SelectedItem);

                    listNewRisks.Items.Clear();
                    listRisksSelected.Items.Clear();
                    listSelected = await databaseActions.ShowRisks(project);

                    for (int i = 0; i < listSelected.Count; i++)
                    {
                        if (listSelected[i].Status == 0)
                            listNewRisks.Items.Add(listSelected[i]);

                        if (listSelected[i].Status == 1)
                            listRisksSelected.Items.Add(listSelected[i]);
                    }

                    DrawGraphic();
                }
                else
                {
                    MessageBox.Show("Wrong in enpty1");
                }
            }
```

```
            catch (Exception)
            {
                MessageBox.Show("Wrong in enpty2");

            }
        }

        private void ListNewRisks_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            if (listNewRisks.SelectedItem != null)
            {
                TextboxInfluenceNew.Text = "0";
                TextboxProbabilityNew.Text = "0";
            }
        }

        /// <summary>
        /// метод срабатывает при нажатии на кнопку мыши
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Canvas_MouseDown(object sender, MouseButtonEventArgs e)
        {
            if (e.GetPosition(null).X < 540 && e.GetPosition(null).Y < 450)
            {
                mousePosition = e.GetPosition(null);
                currentCenter = center;
            }
        }

        /// <summary>
        /// метод для кнопки Set up, который устанавливает новые значения
характеристик для выбранного риска
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void SetUpRiskButton_Click(object sender,
RoutedEventArgs e)
        {
            try
            {
                if (listRisksSelected.SelectedItems == null)
                    throw new NullReferenceException("You need to choose a
risk you want to update!");

                if (Double.Parse(ParseLine(InfluenceTextbox.Text)) == default
|| Double.Parse(ParseLine(ProbabilityTextbox.Text)) == default)
```

```
                    throw new ArgumentException("The 'Probability' and
'Infuence' values must lay in the interval [0;1]!");

                ((Risk)listRisksSelected.SelectedItem).Influence =
double.Parse(ParseLine(InfluenceTextbox.Text));
                ((Risk)listRisksSelected.SelectedItem).Probability =
double.Parse(ParseLine(ProbabilityTextbox.Text));

                DatabaseActions databaseActions = new DatabaseActions();

                await
databaseActions.ChangeRisk((Risk)listRisksSelected.SelectedItem);

                listRisksSelected.Items.Clear();
                listSelected = await databaseActions.ShowRisks(project);

                for (int i = 0; i < listSelected.Count; i++)
                    if (listSelected[i].Status == 1)
                        listRisksSelected.Items.Add(listSelected[i]);

                DrawGraphic();
            }
            catch (NullReferenceException ex)
            {
                MessageBox.Show(ex.Message, "Exception");

            }
            catch (ArgumentException ex)
            {
                MessageBox.Show(ex.Message, "Exception");

            }
            catch (Exception)
            {
                MessageBox.Show("Что-то пошло не так");

            }
        }

        /// <summary>
        /// вводим значение рисков в текстбоксы
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void RiskSelected_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            if (listRisksSelected.Items.Count != 0)
            {
```

```
                InfluenceTextbox.Text =
((Risk)listRisksSelected.SelectedItem).Influence.ToString();
                ProbabilityTextbox.Text =
((Risk)listRisksSelected.SelectedItem).Probability.ToString();
            }
        }

        /// <summary>
        /// метод, который позволяет перемещать гиперболу по графику
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Canvas_MouseMove(object sender, MouseEventArgs e)
        {
            if (e.GetPosition(null).X < 540 && e.GetPosition(null).Y < 450 &&
e.LeftButton == MouseButtonState.Pressed)
            {
                center.X = currentCenter.X - mousePosition.X +
e.GetPosition(null).X;
                center.Y = currentCenter.Y + (mousePosition.X -
e.GetPosition(null).X) / 1.2;

                if (center.X > 650 || center.Y < -100)
                {
                    center.Y = -100;
                    center.X = 650;
                }

                if (center.Y > 230 || center.X < 250)
                {
                    center.Y = 230;
                    center.X = 250;
                }

                DrawGraphic();
                SearchForDangerousRisks();
            }
        }

        /// <summary>
        /// метод, который выводит сообщение с информацией о риске при
нажатии на вершину на графике
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Canvas_MouseRightButtonDown(object sender,
MouseButtonEventArgs e)
        {
            if (listRisksSelected.Items.Count != 0)
```

```
            {
                List<Risk> listRisks = new List<Risk>();

                for (int i = 0; i < listRisksSelected.Items.Count; i++)
                {
                    double _x =
Math.Abs(((Risk)listRisksSelected.Items[i]).point.X - e.GetPosition(null).X);
                    double _y =
Math.Abs(((Risk)listRisksSelected.Items[i]).point.Y - e.GetPosition(null).Y);

                    if (_x <= 10 && _y <= 10)
                        listRisks.Add((Risk)listRisksSelected.Items[i]);
                }

                if (listRisks.Count != 0)
                    MessageBox.Show(CreateLine(listRisks), "Information");
            }
        }

        /// <summary>
        /// метод для удаления рискапри нажатии кнопки Delete
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void DeleteButton_Click(object sender, RoutedEventArgs
e)
        {
            DatabaseActions databaseActions = new DatabaseActions();
            Risk risk = (Risk)((Button)sender).DataContext;

            listSelected.Remove(risk);

            if (listSelected == null)
                listSelected = new List<Risk>();

            listRisksSelected.Items.Remove(risk);
            risk.Status = 2;

            SearchForCurrentRisk(risk);

            await databaseActions.ChangeRisk(risk);

            listRisksNonselected.Items.Add(risk);
            DrawGraphic();
        }

        /// <summary>
        /// метод, который закрывает текущее окно и открывает окно
        /// выбора проектов
```

```
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Back_Click(object sender, RoutedEventArgs e)
        {
            SelectionWindow selectWindow = new SelectionWindow(user);

            Close();
            selectWindow.Show();
        }

        /// <summary>
        /// метод, который при двойном нажатии мыши на риск в разделе опаных
        /// рисков закрывает текущее окно и открывает окно дерева рисков
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void DangerousListRisks_MouseDoubleClick(object sender,
MouseButtonEventArgs e)
        {
            if(listDangerous.SelectedItem!=null&& flag1)
            {
                RiskManagerTree riskTree = new
RiskManagerTree((Risk)listDangerous.SelectedItem, project, user, center);

                Close();
                riskTree.Show();
                flag1 = false;
            }
        }

        /// <summary>
        /// метод, который создаёт строку с информацией о риске
        /// </summary>
        /// <param name="listRisks"></param>
        /// <returns></returns>
        private string CreateLine(List<Risk> listRisks)
        {
            string line = "";

            for (int i = 0; i < listRisks.Count; i++)
                line += $"Name of Risk: {listRisks[i].RiskName}\nSource:
{listRisks[i].Source}\nType: {listRisks[i].Type}\nEffects:
{listRisks[i].Effects}\nDescription: {listRisks[i].Solution}";

            return line;
        }

        /// <summary>
```

```
/// метод ищет риск по его номеру в списке выбранных рисков
/// </summary>
/// <param name="risk"></param>
private void SearchForCurrentRisk(Risk risk)
{
    for (int i = 0; i < listSelected.Count; i++)
    {
        if (listSelected[i].ID == risk.ID)
        {
            listSelected[i].Status = risk.Status;
            listSelected[i].IdUser = risk.IdUser;
            listSelected[i].OwnerLogin = risk.OwnerLogin;
        }
    }
}


/// <summary>
/// метод для добавления риска в список выбранных
/// </summary>
private void AddToSelected()
{
    if (listSelected != null)
    {
        for (int i = 0; i < listSelected.Count; i++)
        {
            if (listSelected[i].Status == 1)
                listRisksSelected.Items.Add(listSelected[i]);
            else
            {
                if (listSelected[i].Status == 0)
                    listNewRisks.Items.Add(listSelected[i]);
                else
                    listRisksSelected.Items.Add(listSelected[i]);
            }
        }
    }
}

/// <summary>
/// метод, который отрисовывает гиперболу и вершины
/// </summary>
private void DrawGraphic()
{
    canvas.Children.Clear();

    for (int i = 0; i < C - 1; i++)
    {
        double oldX = center.X - radius + radius / C * i;
```

```
                double currentX = center.X - radius + radius / C * (i + 1);

                Line line = new Line();

                line.X1 = oldX;
                line.X2 = currentX;
                line.Y1 = FindYCoordinate(oldX, radius, center);
                line.Y2 = FindYCoordinate(currentX, radius, center);
                line.Stroke = Brushes.Black;

                canvas.Children.Add(line);
            }

            for (int i = 0; i < listRisksSelected.Items.Count; i++)
            {
                if (((Risk)listRisksSelected.Items[i]).Probability !=
default(Double) &&
                    ((Risk)listRisksSelected.Items[i]).Influence !=
default(Double) && ((Risk)listRisksSelected.Items[i]).Status == 1)
                {
                    ((Risk)listRisksSelected.Items[i]).point.X = 425 *
((Risk)listRisksSelected.Items[i]).Probability + 75;
                    ((Risk)listRisksSelected.Items[i]).point.Y = -350 *
((Risk)listRisksSelected.Items[i]).Influence + 400;

                    Ellipse ellipse = new Ellipse();

                    ellipse.Height = 10;
                    ellipse.Width = 10;
                    ellipse.StrokeThickness = 2;

                    if (Math.Sqrt((((Risk)listRisksSelected.Items[i]).point.X
- center.X) * (((Risk)listRisksSelected.Items[i]).point.X - center.X) +
                        (((Risk)listRisksSelected.Items[i]).point.Y -
center.Y) * (((Risk)listRisksSelected.Items[i]).point.Y - center.Y)) <
radius)
                    {
                        ellipse.Stroke = Brushes.Red;
                        ellipse.Fill = Brushes.Red;
                    }
                    else
                    {
                        ellipse.Stroke = Brushes.Green;
                        ellipse.Fill = Brushes.Green;
                    }

                    ellipse.VerticalAlignment = VerticalAlignment.Top;
                    ellipse.HorizontalAlignment = HorizontalAlignment.Left;
```

```
                    ellipse.Margin = new
Thickness(((Risk)listRisksSelected.Items[i]).point.X,
((Risk)listRisksSelected.Items[i]).point.Y, 0, 0);

                    canvas.Children.Add(ellipse);
                }
            }
        }
        /// <summary>
        /// метод, который вычисляет координату y с помощью квадратного
уравнения
        /// </summary>
        /// <param name="x"></param>
        /// <param name="radius"></param>
        /// <param name="center"></param>
        /// <returns></returns>
        /// <exception cref="Exception"></exception>
        static public double FindYCoordinate(double x, double radius, Point
center)
        {
            double a = 1;
            double b = -2 * center.Y;
            double c = -(radius * radius) + (x - center.X) * (x - center.X) +
center.Y * center.Y;
            double d = (b * b - 4 * a * c);

            if (d < 0)
                throw new Exception("There's no such point");

            return (-b + Math.Sqrt(d)) / 2 * a;
        }

        /// <summary>
        /// метод, который ищет среди рисков проекта опасные и добавляет в
раздел Dangerous Risks
        /// </summary>
        private void SearchForDangerousRisks()
        {
            listDangerous.Items.Clear();

            if (listRisksSelected.Items.Count != 0)
            {
                for (int i = 0; i < listRisksSelected.Items.Count; i++)
                {
                    double c =
Math.Sqrt((((Risk)listRisksSelected.Items[i]).point.X - center.X) *

(((Risk)listRisksSelected.Items[i]).point.X - center.X) +
```

```
(((Risk)listRisksSelected.Items[i]).point.Y - center.Y) *

(((Risk)listRisksSelected.Items[i]).point.Y - center.Y));
                    if (c < radius)

listDangerous.Items.Add((Risk)listRisksSelected.Items[i]);
                }
            }
        }


        /// <summary>
        /// метод для парсинга строки
        /// . заменяется на ,
        /// </summary>
        /// <param name="line"></param>
        /// <returns>возвращается строка</returns>
        private string ParseLine(string line)
        {
            string result = "";

            for (int i = 0; i < line.Length; i++)
            {
                if (line[i] == '.')
                    result += ',';
                else
                    result += line[i];
            }

            return result;
        }
    }
}
```

1.4.2. RiskManagerTree.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
```

```
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace RiskApp
{
    public partial class RiskManagerTree : Window
    {
        bool flag = true;

        double Widht;
        new double Height;

        string pathToBack =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "back.jpg");
        string pathToPlus =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "plus.png");

        Point center;

        User user = null;
        Risk risk = null;
        Project project;
        Vertex _vertexFirst;

        List<double> listValues = new List<double>();
        List<Vertex> listVertex = new List<Vertex>();
        public RiskManagerTree(Risk risk, Project project, User user, Point
center)
        {
            this.center = center;
            this.user = user;
            this.risk = risk;
            this.project = project;

            InitializeComponent();
        }

        /// <summary>
        /// метод запускается при открытии окна
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void Window_Activated(object sender, EventArgs e)
        {
            if (flag)
            {
                try
                {
                    Label label = new Label();
```

```
                    Tree tree = new Tree();
                    Button button = new Button();

                    Widht = canvas.ActualWidth;
                    Height = canvas.ActualHeight;

                    label.HorizontalAlignment = HorizontalAlignment.Center;
                    label.VerticalAlignment = VerticalAlignment.Top;
                    label.Content = risk.RiskName;
                    grid.Children.Add(label);

                    button.HorizontalAlignment = HorizontalAlignment.Left;
                    button.VerticalAlignment = VerticalAlignment.Top;
                    button.Margin = new Thickness(Widht / 2 - 10, 50, Widht /
2 - 10, Height - 70);
                    button.Background = new ImageBrush(new BitmapImage(new
Uri(pathToPlus)));

                    BackButton.Background = new ImageBrush(new
BitmapImage(new Uri(pathToBack)));
                    BackButton.Foreground = new ImageBrush(new
BitmapImage(new Uri(pathToBack)));

                    if (!await tree.CheckIfExistInDataBase(risk.ID))
                    {
                        _vertexFirst = new Vertex(Widht / 2, 50, risk.ID,
risk.RiskName);
                        await tree.AddVertex(risk.ID, _vertexFirst);
                        _vertexFirst = await tree.ShowVertex(risk.ID);
                        button.DataContext = _vertexFirst;
                    }
                    else
                    {
                        _vertexFirst = await tree.ShowVertex(risk.ID);
                        button.DataContext = _vertexFirst;
                    }

                    button.Height = 20;
                    button.Width = 20;
                    button.Click += Button_Click;

                    canvas.Children.Add(button);
                    listVertex = await tree.ShowListVertexes();

                    DrawRoots(_vertexFirst);
                    CurrentBranchCost(_vertexFirst, 0);
                    DrawDangerousMaximum();

                    flag = false;
```

```
                }
                catch (NullReferenceException ex)
                {
                    MessageBox.Show("Error: " + ex.Message);
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Error: " + ex.Message);
                }
            }
        }

        /// <summary>
        /// метод, который добавляет новую вершину с установленными
характеристиками
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void AddButton_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                if (((Button)sender).DataContext == null)
                    throw new Exception("You need to choose the top!");

                if (TextboxDescription.Text == "")
                    throw new Exception("You must fill the field
'Description'!");

                if (!double.TryParse(TextboxCost.Text, out double d) || d <=
0)
                    throw new Exception("The value of 'Cost' can only be of
Double type and it must be more than 0!");

                if (!double.TryParse(TextboxProbability.Text, out double d1)
|| d1 > 1 || d1 < 0)
                    throw new Exception("The value of 'Probability' can only
be of Double type and it must be in the interval [0, 1]!");

                Vertex parentVertex = ((Vertex)((Button)sender).DataContext);

                int row = 1;

                FindRow(parentVertex, ref row);

                if (parentVertex.Probability != 0)
                    row++;

                if (row >= 4)
```

```
                    throw new ArgumentException("Branches cannot be more than
4!");

                Vertex currentVertex;
                string line = $"{(parentVertex.X - Widht / (2 * Math.Pow(4,
row))):f3}";

                if (CheckIfInList(double.Parse(line)))
                    currentVertex = new Vertex(parentVertex.X - Widht / (2 *
Math.Pow(4, row)), parentVertex.Y + 50, double.Parse(TextboxCost.Text),
                        double.Parse(TextboxProbability.Text),
parentVertex.ID, TextboxDescription.Text);
                else
                {
                    line = $"{(parentVertex.X + Widht / (2 * Math.Pow(4,
row))):f3}";

                    if (CheckIfInList(double.Parse(line)))
                        currentVertex = new Vertex(parentVertex.X + Widht /
(2 * Math.Pow(4, row)), parentVertex.Y + 50, double.Parse(TextboxCost.Text),
double.Parse(TextboxProbability.Text),
                            parentVertex.ID, TextboxDescription.Text);
                    else
                    {
                        line = $"{(parentVertex.X - 3 * Widht / (2 *
Math.Pow(4, row))):f3}";

                        if (CheckIfInList(double.Parse(line)))
                            currentVertex = new Vertex(parentVertex.X - 3 *
Widht / (2 * Math.Pow(4, row)), parentVertex.Y + 50,
                                double.Parse(TextboxCost.Text),
double.Parse(TextboxProbability.Text), parentVertex.ID,
TextboxDescription.Text);
                        else
                        {
                            line = $"{(parentVertex.X + 3 * Widht / (2 *
Math.Pow(4, row))):f3}";

                            if (CheckIfInList(double.Parse(line)))
                                currentVertex = new Vertex(parentVertex.X + 3
* Widht / (2 * Math.Pow(4, row)), parentVertex.Y + 50,
double.Parse(TextboxCost.Text), double.Parse(TextboxProbability.Text),
parentVertex.ID, TextboxDescription.Text);
                            else
                                throw new Exception("The amount of children
cannot be more than 4!");
                        }
                    }
                }
```

```
            Tree tree = new Tree();

            await tree.AddVertex(parentVertex.ID, currentVertex);
            currentVertex = await tree.ShowVertex(currentVertex);
            listVertex.Add(currentVertex);

            RefreshTree(_vertexFirst);

            for (int i = 0; i < listVertex.Count; i++)
                listVertex[i].Value = default;

            CurrentBranchCost(_vertexFirst, 0);
            DrawDangerousMaximum();
        }
        catch (ArgumentException ex)
        {
            MessageBox.Show(ex.Message, "Enpty Exception");
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Enpty Exception");
        }
    }


    private void DrawLineGreen(Vertex minVertex)
    {
        for (int i = 0; i < listVertex.Count; i++)
        {
            if (minVertex.IDParent == listVertex[i].ID)
            {
                Line line = new Line();
                line.X1 = listVertex[i].X;
                line.Y1 = listVertex[i].Y + 20;
                line.X2 = minVertex.X;
                line.Y2 = minVertex.Y;
                line.Stroke = Brushes.Green;
                canvas.Children.Add(line);

                if (listVertex[i].Probability == default)
                    break;
                else
                    DrawLineGreen(listVertex[i]);
            }
        }
    }

    private void DrawLineRed(Vertex maxVertex)
```

```
        {
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (maxVertex.IDParent == listVertex[i].ID)
                {
                    Line line = new Line();
                    line.X1 = listVertex[i].X;
                    line.Y1 = listVertex[i].Y + 20;
                    line.X2 = maxVertex.X;
                    line.Y2 = maxVertex.Y;

                    line.Stroke = Brushes.Red;
                    canvas.Children.Add(line);

                    if (listVertex[i].Probability == default)
                        break;
                    else
                        DrawLineRed(listVertex[i]);

                }
            }
        }

        /// <summary>
        /// метод, который рисует дерево при начальной загрузке
        /// </summary>
        /// <param name="curver"></param>
        private void DrawRoots(Vertex curver)
        {
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (listVertex[i].IDParent == curver.ID &&
listVertex[i].Probability != default(double))
                {
                    AddNewVertex(listVertex[i]);
                    DrawNewLine(listVertex[i], curver);
                    DrawRoots(listVertex[i]);
                }
            }
        }

        /// <summary>
        /// нажатие на вершину графа
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Button_Click(object sender, RoutedEventArgs e)
        {
            AddButton.DataContext = (Vertex)((Button)sender).DataContext;
```

```
            Vertex vertex = (Vertex)AddButton.DataContext;

        }

        /// <summary>
        /// метод для удаления вершины из дерева
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void DeleteButton_Click(object sender, RoutedEventArgs
e)
        {
            canvas.Children.Clear();

            Tree tree = new Tree();

            if (((Vertex)AddButton.DataContext).Probability != 0)
            {
                List<Vertex> listDelete =
DeleteVertexes((Vertex)AddButton.DataContext);
                listDelete.Add((Vertex)AddButton.DataContext);
                await tree.DeleteVertex(listDelete);
                listVertex = await tree.ShowListVertexes();
            }

            Button button = new Button();

            button.HorizontalAlignment = HorizontalAlignment.Left;
            button.VerticalAlignment = VerticalAlignment.Top;
            button.Margin = new Thickness(Widht / 2 - 10, 50, Widht / 2 - 10,
Height - 70);
            button.Background = new ImageBrush(new BitmapImage(new
Uri(pathToPlus)));

            BackButton.Background = new ImageBrush(new BitmapImage(new
Uri(pathToBack)));

            button.DataContext = _vertexFirst;
            button.Height = 20;
            button.Width = 20;
            button.Click += Button_Click;

            canvas.Children.Add(button);

            DrawRoots(_vertexFirst);
            CurrentBranchCost(_vertexFirst, 0);
            DrawDangerousMaximum();
        }
```

```
        /// <summary>
        /// метод для удаления вершин
        /// </summary>
        /// <param name="currentvertex"></param>
        /// <returns></returns>
        private List<Vertex> DeleteVertexes(Vertex currentvertex)
        {
            List<Vertex> listDelete = new List<Vertex>();

            for (int i = 0; i < listVertex.Count; i++)
            {
                if (listVertex[i].Probability != 0 && listVertex[i].IDParent
== currentvertex.ID)
                    listDelete.Add(listVertex[i]);
            }

            for (int i = 0; i < listVertex.Count; i++)
            {
                for (int j = 0; j < listDelete.Count; j++)
                {
                    if (listVertex[i].Probability != 0 &&
listVertex[i].IDParent == listDelete[j].ID)
                        listDelete.Add(listVertex[i]);
                }
            }

            return listDelete;
        }

        /// <summary>
        /// метод, который срабатывает при нажатии на кнопку Create Report,
        /// закрывает окно дерева, открывает окно отчёта
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void GoToReport_Click(object sender, RoutedEventArgs e)
        {
            ReportWindow reportWindow = new ReportWindow(risk, project,
center);
            Close();
            reportWindow.Show();
        }

        /// <summary>
        /// метод срабатывает при нажатии кнопки Back
        /// закрывается окно с деревом, снова открывается окно с графиком
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
```

```
        private void BackButton_Click(object sender, RoutedEventArgs e)
        {
            RiskManagerGraphic graphicWindow = new
RiskManagerGraphic(project,user);
            Close();
            graphicWindow.Show();
        }
        /// <summary>
        /// метод проверяет по значению вершины x, находится ли такой же
элемент в списке
        /// </summary>
        /// <param name="x"></param>
        /// <returns></returns>
        private bool CheckIfInList(double x)
        {
            for (int i = 0; i < listVertex.Count; i++)
                if (x == listVertex[i].X)
                    return false;

            return true;
        }

        /// <summary>
        /// метод для установления стоимости ветки
        /// </summary>
        /// <param name="curver"></param>
        /// <param name="k"></param>
        private void CurrentBranchCost(Vertex currentVertex, double c)
        {
            double cost = c;

            for (int i = 0; i < listVertex.Count; i++)
            {
                if (currentVertex.ID == listVertex[i].IDParent &&
listVertex[i].Probability != default)
                {
                    cost += listVertex[i].Probability * listVertex[i].Cost;
                    CurrentBranchCost(listVertex[i], cost);
                    cost -= listVertex[i].Probability * listVertex[i].Cost;
                }
            }

            if (CheckIfChildExists(currentVertex))
                currentVertex.Value = cost;
        }

        /// <summary>
        /// метод, котоый проверяет, есть ли у вершины ребенок
        /// </summary>
```

```
        /// <param name="currentVertex"></param>
        /// <returns></returns>
        private bool CheckIfChildExists(Vertex currentVertex)
        {
            for (int i = 0; i < listVertex.Count; i++)
                if (currentVertex.ID == listVertex[i].IDParent &&
listVertex[i].Probability != default)
                    return false;

            return true;
        }


        /// <summary>
        /// метод, который возвращает ряд дерева
        /// </summary>
        /// <param name="parent"></param>
        /// <param name="k"></param>
        private void FindRow(Vertex parentVertex, ref int k)
        {
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (parentVertex.IDParent == listVertex[i].ID &&
listVertex[i].Probability == default)
                    return;

                if (parentVertex.IDParent == listVertex[i].ID &&
listVertex[i].Probability != default)
                {
                    k++;
                    FindRow(listVertex[i], ref k);
                }
            }
        }

        /// <summary>
        /// метод, который рисует линии между вершинами
        /// </summary>
        /// <param name="newver"></param>
        /// <param name="parent"></param>
        private void DrawNewLine(Vertex vertex, Vertex parentVertex)
        {
            Line line = new Line();

            line.X1 = parentVertex.X;
            line.Y1 = parentVertex.Y + 20;
            line.X2 = vertex.X;
            line.Y2 = vertex.Y;
            line.Stroke = Brushes.Black;
```

```
            canvas.Children.Add(line);
        }

        /// <summary>
        /// метод для 'обновления' дерева
        /// добавляются, либо удаляются элементы
        /// </summary>
        /// <param name="currentVertex"></param>
        private void RefreshTree(Vertex currentVertex)
        {
            canvas.Children.Clear();
            Button button = new Button();

            button.HorizontalAlignment = HorizontalAlignment.Left;
            button.VerticalAlignment = VerticalAlignment.Top;
            button.Margin = new Thickness(Widht / 2 - 10, 50, Widht / 2 - 10,
Height - 70);

            button.Background = new ImageBrush(new BitmapImage(new
Uri(pathToPlus)));
            BackButton.Background = new ImageBrush(new BitmapImage(new
Uri(pathToBack)));

            button.DataContext = _vertexFirst;
            button.Height = 20;
            button.Width = 20;
            button.Click += Button_Click;

            canvas.Children.Add(button);

            for (int i = 0; i < listVertex.Count; i++)
            {
                if (listVertex[i].IDParent == currentVertex.ID &&
listVertex[i].Probability != default)
                {
                    AddNewVertex(listVertex[i]);
                    DrawNewLine(listVertex[i], currentVertex);
                    DrawRoots(listVertex[i]);
                }
            }
        }

        /// <summary>
        /// метод, который добавляет новые вершины
        /// </summary>
        /// <param name="newver"></param>
        private void AddNewVertex(Vertex vertex)
        {
```

```
            Button plusButton = new Button();

            plusButton.DataContext = vertex;
            plusButton.HorizontalAlignment = HorizontalAlignment.Left;
            plusButton.VerticalAlignment = VerticalAlignment.Top;
            plusButton.Margin = new Thickness(vertex.X - 10, vertex.Y, Widht
- vertex.X - 10, Height - vertex.Y - 20);
            plusButton.Height = 20;
            plusButton.Width = 20;
            plusButton.Background = new ImageBrush(new BitmapImage(new
Uri(pathToPlus)));
            plusButton.Click += Button_Click;

            canvas.Children.Add(plusButton);
        }

        /// <summary>
        /// метод, который находит наиболее опасный рис и рисует его
        /// </summary>
        private void DrawDangerousMaximum()
        {
            Vertex vertexMaximum = listVertex[0];
            Vertex vertexMinimum = null;

            for (int i = 0; i < listVertex.Count; i++)
            {
                if (listVertex[i].Value != 0)
                {
                    vertexMinimum = listVertex[i];

                    for (int j = 0; j < listVertex.Count; j++)
                    {
                        if (listVertex[j].Value < vertexMinimum.Value &&
listVertex[j].Value != 0)
                            vertexMinimum = listVertex[j];
                    }

                    break;
                }

            }

            for (int i = 1; i < listVertex.Count; i++)
            {
                if (listVertex[i].Value > vertexMaximum.Value)
                    vertexMaximum = listVertex[i];
            }

            if (vertexMaximum != null)
```

```
            {
                Label label = new Label();

                label.Content = vertexMaximum.Value;
                label.Margin = new Thickness(vertexMaximum.X, vertexMaximum.Y
+ 20, 0, 0);
                label.VerticalAlignment = VerticalAlignment.Top;
                label.HorizontalAlignment = HorizontalAlignment.Left;
                label.Height = 40;
                label.Foreground = Brushes.Red;

                canvas.Children.Add(label);
                DrawLineRed(vertexMaximum);
            }

            if (vertexMinimum != null)
            {
                Label label1 = new Label();

                label1.Content = vertexMinimum.Value;
                label1.Margin = new Thickness(vertexMinimum.X,
vertexMinimum.Y + 20, 0, 0);
                label1.VerticalAlignment = VerticalAlignment.Top;
                label1.HorizontalAlignment = HorizontalAlignment.Left;
                label1.Height = 40;
                label1.Foreground = Brushes.Green;

                canvas.Children.Add(label1);
                DrawLineGreen(vertexMinimum);
            }
        }

    }
}
```

1.4.3. SelectionWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
```

```
using System.Windows.Shapes;

namespace RiskApp
{
    /// <summary>
    /// Логика взаимодействия для SelectProject.xaml
    /// </summary>
    public partial class SelectionWindow : Window
    {
        bool flag = true;
        User user = null;
        string path =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "back.jpg");

        public SelectionWindow(User user)
        {
            this.user = user;

            InitializeComponent();
        }

        /// <summary>
        /// метод, который срабатывает при нажатии кнопки BackButton,
        /// закрывает окно выбора и открывает окно ввода логина и пароля
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void BackButton_Click(object sender, RoutedEventArgs e)
        {
            MainWindow mainWindow = new MainWindow();

            Close();
            mainWindow.Show();
        }

        /// <summary>
        /// метод, который при выборе объекта в списке открывает окно проекта
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ListBox_MouseDoubleClick(object sender,
MouseButtonEventArgs e)
        {
            if (listBox.SelectedItem != null)
            {
                Project project = (Project)listBox.SelectedItem;
                RiskManagerGraphic graphic = new RiskManagerGraphic(project,
user);

                Close();
                graphic.Show();
            }
```

```
            else
            {
                MessageBox.Show("You need to select project!");
            }

        }

        /// <summary>
        /// метод, который запускается при открытии окна
        /// открывается список проектов менеджера
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async  void Window_Activated(object sender, EventArgs e)
        {
            if (flag)
            {
                ProjectActions projectActions = new ProjectActions();
                DatabaseActions databaseActions = new DatabaseActions();

                List<string> listName = await
databaseActions.ShowUserProjects(user);
                List<Project> listProjects = await
projectActions.ShowProjects();

                BackButton.Background = new ImageBrush(new BitmapImage(new
Uri(path)));
                BackButton.Foreground = new ImageBrush(new BitmapImage(new
Uri(path)));

                try
                {
                    for (int i = 0; i < listProjects.Count; i++)
                    {
                        for (int j = 0; j < listName.Count; j++)
                        {
                            if (listName[j] == listProjects[i].Name)
                                listBox.Items.Add(listProjects[i]);
                        }
                    }
                }
                catch (NullReferenceException)
                {
                    MessageBox.Show("You havent projects and risks");
                }

                flag = false;
            }
        }
    }
}
```

1.5. MainWindow.xaml.cs

```
/*
 * Курсовая работа
 * Программа иддентификации и анализа рисков ИТ-проектов
 * Ф.И.О. исполнителя: Карпова Александра Евгеньевна, группа БПИ 181
 * Ф.И.О. руководителя: Песоцкая Елена Юрьевна
 * Год создания: 2020
 */

using System.ComponentModel;
using System.Windows;

namespace RiskApp
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
        private async void EnterButton_Click(object sender, RoutedEventArgs
e)
        {
            UserActions userActions = new UserActions();

            if (await userActions.CheckLogin(loginBox.Text.Trim(),
passwordBox.Password.Trim()) == 3)
            {
                AdminProjects adminProject = new AdminProjects();
                Close();
                adminProject.Show();
            }
            else
            {
                if (await userActions.CheckLogin(loginBox.Text.Trim(),
passwordBox.Password.Trim()) == 2)
                {
                    User user = await
userActions.SearchForUser(loginBox.Text.Trim(), passwordBox.Password.Trim());
                    ChoiceWindow choice = new ChoiceWindow(user);
                    Close();
                    choice.Show();
                }
                else
                {
                    if (await userActions.CheckLogin(loginBox.Text.Trim(),
passwordBox.Password.Trim()) == 1)
                    {
                        User user = await
userActions.SearchForUser(loginBox.Text.Trim(), passwordBox.Password.Trim());
```

```
                        SelectionWindow selectWindow = new
SelectionWindow(user);
                        selectWindow.Show();
                        Close();
                    }
                    else
                        MessageBox.Show("Error occured after entering login
and password!");
                }
            }
        }

        private void Window_Closing(object sender, CancelEventArgs e)
        {

        }
    }
}
```

1.6. RiskSettingsWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace RiskApp
{
    public partial class RiskSettingsWindow : Window
    {
        bool flag = true;

        public double Influence
        {
            get => Double.Parse(ParseLine(InfluenceTextbox.Text));
        }
        public double Probability
        {
            get => Double.Parse(ParseLine(ProbabilityTextbox.Text));
        }
```

```
        public User Owner
        {
            get => (User)UsersCombobox.SelectedItem;
        }

        public RiskSettingsWindow()
        {
            InitializeComponent();
        }

        /// <summary>
        /// метод добавляет в combobox всех пользователей
        /// </summary>
        /// <returns></returns>
        private async Task AddUsersToCombobox()
        {
            List<User> listUser = await new UserActions().ShowUsers();

            for (int i = 0; i < listUser.Count; i++)
                UsersCombobox.Items.Add(listUser[i]);
        }

        /// <summary>
        /// метод срабатывает при нажатии кнопки "Add clean risk"
        /// создаётсяя риск без заданных характеристик
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void CleanRisk_Click(object sender, RoutedEventArgs e)
        {
            DialogResult = true;
            InfluenceTextbox.Text = default(double).ToString();
            ProbabilityTextbox.Text = default(double).ToString();
        }

        /// <summary>
        /// метод запускается при нажатии кнопки Create Estimated Risk,
        /// риск задаётся с установленными значениями характеристик
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        /// <exception cref="ArgumentException"></exception>
        /// <exception cref="NullReferenceException"></exception>
        private void SetEstimatedRisk_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                if (Double.Parse(ParseLine(InfluenceTextbox.Text)) >= 1 ||
Double.Parse(ParseLine(InfluenceTextbox.Text)) <= 0)
                    throw new ArgumentException("The value of the field
'Influence' must lay in the interval (0,1)");
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.13-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```
                if (Double.Parse(ParseLine(ProbabilityTextbox.Text)) >= 1 ||
Double.Parse(ParseLine(ProbabilityTextbox.Text)) <= 0)
                    throw new ArgumentException("The value of the field
'Probability' must lay in the interval (0,1)");

                if (UsersCombobox.SelectedItem == null)
                    throw new NullReferenceException("You must choose
project's owner in the combobox!");

                this.DialogResult = true;
            }
            catch(ArgumentException ex)
            {
                MessageBox.Show(ex.Message, "Exception");
            }
            catch (NullReferenceException ex)
            {
                MessageBox.Show(ex.Message, "Exception");
            }
            catch (Exception ex)
            {
                MessageBox.Show("Something went wrong!" + ex.Message);
            }
        }

        /// <summary>
        /// метод запускается при открытии окна настроек
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void Window_Activated(object sender, EventArgs e)
        {
            if (flag)
            {
                UsersCombobox.Text = "Choose the User ";
                await AddUsersToCombobox();
                flag = false;
            }
        }

        /// <summary>
        /// метод для парсинга строки
        /// . заменяется на ,
        /// </summary>
        /// <param name="line"></param>
        /// <returns>возвращается строка</returns>
        private string ParseLine(string line)
        {
            string result = "";

            for (int i = 0; i < line.Length; i++)
            {
```

```
                if (line[i] == '.')
                    result += ',';
                else
                    result += line[i];
            }

            return result;
        }
    }
}
```

1.7. ReportWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using Word = Microsoft.Office.Interop.Word;

namespace RiskApp
{
    public partial class ReportWindow : Window
    {
        const int K = 100;
        const double radius = 250;

        bool flag = true;
        double Widht;
        new double Height;

        Risk _risk = null;
        Project _project = null;
        List<Risk> listSelected = null;
        List<Vertex> listVertex = null;
        Vertex firstVertex;

        private Point center;

        object offMissing = System.Reflection.Missing.Value;
```

```
        object oEndOfDoc = "\\endofdoc";

        Word.Application wordApplication = new Word.Application();
        Word.Document oDoc;

        public ReportWindow(Risk risk,Project project, System.Windows.Point
center)
        {
            this.center = center;
            _risk = risk;
            _project = project;

            InitializeComponent();
        }

        /// <summary>
        /// метод выполняеся при открытии окна отчёта
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void Window_Activated(object sender, EventArgs e)
        {
            if(flag)
            {
                Widht = canvas.ActualWidth;
                Height = canvas.ActualHeight;

                DatabaseActions databaseActions = new DatabaseActions();
                listSelected = await databaseActions.ShowRisks(_project);

                if(listSelected == null)
                    listSelected = new List<Risk>();

                DrawLine();

                Tree tree = new Tree();

                listVertex = await tree.ShowListVertexes();
                firstVertex = await tree.ShowVertex(_risk.ID);

                Label label = new Label();

                label.HorizontalAlignment = HorizontalAlignment.Left;
                label.VerticalAlignment = VerticalAlignment.Top;
                label.Margin = new Thickness(Widht/2-100, 0, 0, 0);
                label.FontSize = 17;
                label.Content = $"Report for {_project.Name}";

                canvas.Children.Add(label);
```

```
                DrawVertex();
                CurrentBranchCost(firstVertex, 0);
                DrawDangerousMaximum();
                WriteDangerousItems();

                flag = false;

                //wordApplication.Visible = true;
                CreateDocument();
            }
        }

        /// <summary>
        /// метод для добавления в документ таблицы опасных рисков
        /// </summary>
        private void CreateDocument()
        {
            oDoc = wordApplication.Documents.Add(ref offMissing, ref
offMissing,
                    ref offMissing, ref offMissing);

            Word.Paragraph para1;
            para1 = oDoc.Content.Paragraphs.Add(ref offMissing);
            para1.Range.Text = $"Report for {_risk} in {_project}";
            para1.Range.ParagraphFormat.Alignment =
Microsoft.Office.Interop.Word.WdParagraphAlignment.wdAlignParagraphCenter;
            para1.Range.Font.Bold = 1;
            para1.Format.SpaceAfter = 12;
            para1.Range.InsertParagraphAfter();

            Word.Table oTable;
            Word.Range wrdRng = oDoc.Bookmarks.get_Item(ref oEndOfDoc).Range;
            oTable = oDoc.Tables.Add(wrdRng, listDangerous.Items.Count + 1,
5, ref offMissing, ref offMissing);
            oTable.Range.ParagraphFormat.SpaceAfter = 2;

            string strText;
            oTable.Range.Borders.OutsideLineStyle =
Word.WdLineStyle.wdLineStyleSingle;
            oTable.Range.Borders.InsideLineStyle =
Word.WdLineStyle.wdLineStyleSingle;
            //   oTable.Borders.InsideLineWidth =
Word.WdLineWidth.wdLineWidth050pt;
            // oTable.Range.Borders.InsideLineStyle =
Word.WdLineStyle.wdLineStyleDashDot;
            oTable.Cell(1, 1).Range.Text = "Name of Risk";
            oTable.Cell(1, 2).Range.Text = "Source";
            oTable.Cell(1, 3).Range.Text = "Possible Solution";
```

```
            oTable.Cell(1, 4).Range.Text = "Effects";
            oTable.Cell(1, 5).Range.Text = "Rank";

            for (int r = 2, i = 0; r <= listDangerous.Items.Count + 1; r++,
i++)
            {
                strText =
System.Convert.ToString(((Risk)listDangerous.Items[i]).RiskName);
                oTable.Cell(r, 1).Range.Text = strText;

                strText =
System.Convert.ToString(((Risk)listDangerous.Items[i]).Source);
                oTable.Cell(r, 2).Range.Text = strText;

                strText =
System.Convert.ToString(((Risk)listDangerous.Items[i]).Solution);
                oTable.Cell(r, 3).Range.Text = strText;

                strText =
System.Convert.ToString(((Risk)listDangerous.Items[i]).Effects);
                oTable.Cell(r, 4).Range.Text = strText;

                strText =
System.Convert.ToString(((Risk)listDangerous.Items[i]).Rank);
                oTable.Cell(r, 5).Range.Text = strText;
            }
        }

        /// <summary>
        /// метод, который выводит в MessageBox информацию о выбранном на
графике риске
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Canvas_MouseRightButtonDown(object sender,
MouseButtonEventArgs e)
        {
            if (listDangerous.Items.Count != 0)
            {
                List<Risk> list = new List<Risk>();

                for (int i = 0; i < listDangerous.Items.Count; i++)
                {
                    double _x =
Math.Abs(((Risk)listDangerous.Items[i]).point.X - e.GetPosition(null).X);
                    double _y =
Math.Abs(((Risk)listDangerous.Items[i]).point.Y - e.GetPosition(null).Y);

                    if (_x <= 10 && _y <= 10)
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.13-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```
                        list.Add((Risk)listDangerous.Items[i]);
                }

                if (list.Count != 0)
                    MessageBox.Show(CreateLine(list), "Information about
Selected Risks");
            }
        }

        /// <summary>
        /// метод, который возвращает строку с необходимой информацией
        /// </summary>
        /// <param name="click"></param>
        /// <returns></returns>
        private string CreateLine(List<Risk> list)
        {
            string line = "";

            for (int i = 0; i < list.Count; i++)
                line += $"Name of Risk: {listSelected[i].RiskName}\nSource:
{listSelected[i].Source}\nType: {listSelected[i].Type}\nEffects:
{listSelected[i].Effects}\nPossible Solution: {listSelected[i].Solution}";

            return line;
        }

        /// <summary>
        /// метод составляет список опасных объектов
        /// </summary>
        private void WriteDangerousItems()
        {
            for (int i = 0; i < listSelected.Count; i++)
            {
                double m = Math.Sqrt((listSelected[i].point.X - center.X) *
(listSelected[i].point.X - center.X) +
                                     (listSelected[i].point.Y - center.Y) *
(listSelected[i].point.Y - center.Y));

                if ((listSelected[i].Status == 1) && (m < radius))
                    listDangerous.Items.Add(listSelected[i]);
            }
        }

        /// <summary>
        /// метод создаёт первую вершину графика
        /// </summary>
        private void DrawVertex()
        {
            Point point = new Point(firstVertex.X * 3 / 2, firstVertex.Y);
```

```
            Label label = new Label();

            label.HorizontalAlignment = HorizontalAlignment.Center;
            label.VerticalAlignment = VerticalAlignment.Top;
            label.FontSize = 17;
            label.Margin = new Thickness(point.X - 200, point.Y + 200, 0, 0);
            label.Content = _risk.RiskName;

            canvas.Children.Add(label);

            Ellipse ellipse = new Ellipse();

            ellipse.Width = 4.5;
            ellipse.Height = 4.5;
            ellipse.HorizontalAlignment = HorizontalAlignment.Left;
            ellipse.VerticalAlignment = VerticalAlignment.Top;
            ellipse.StrokeThickness = 3;
            ellipse.Stroke = Brushes.Black;
            ellipse.Margin = new Thickness(point.X - 2, point.Y - 2, 0, 0);

            canvas.Children.Add(ellipse);

            DrawByRoot(firstVertex);
        }

        /// <summary>
        /// метод добавляет вершины по корню
        /// </summary>
        /// <param name="currentVertex"></param>
        private void DrawByRoot(Vertex currentVertex)
        {
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (listVertex[i].Probability != default &&
listVertex[i].IDParent == currentVertex.ID)
                {
                    DrawNewVertex(listVertex[i]);
                    AddLine(listVertex[i], currentVertex);
                    DrawByRoot(listVertex[i]);
                }
            }
        }

        /// <summary>
        /// метод для добавления новой вершины
        /// </summary>
        /// <param name="vertex"></param>
        private void DrawNewVertex(Vertex vertex)
        {
```

```
            Point point = new Point(vertex.X, vertex.Y);
            Ellipse ellipse = new Ellipse();

            ellipse.Width = 6.5;
            ellipse.Height = 6.5;
            ellipse.HorizontalAlignment = HorizontalAlignment.Left;
            ellipse.VerticalAlignment = VerticalAlignment.Top;
            ellipse.StrokeThickness = 3;
            ellipse.Stroke = Brushes.Black;
            ellipse.Margin = new Thickness(vertex.X / 2 + Widht / 2 - 3,
vertex.Y -3, 0, 0);

            canvas.Children.Add(ellipse);
        }

        /// <summary>
        /// метод для добавления новой линии
        /// </summary>
        /// <param name="newver"></param>
        /// <param name="parent"></param>
        private void AddLine(Vertex vertex, Vertex parentVertex)
        {
            Line line = new Line();

            line.X1 = parentVertex.X / 2 + Widht / 2;
            line.Y1 = parentVertex.Y;
            line.X2 = vertex.X / 2 + Widht / 2;
            line.Y2 = vertex.Y ;
            line.Stroke = Brushes.Black;

            canvas.Children.Add(line);
        }
        /// <summary>
        /// метод для подсчёта стоимости ветки
        /// </summary>
        /// <param name="vertex"></param>
        /// <param name="c"></param>
        private void CurrentBranchCost(Vertex vertex, double c)
        {
            double cost = c;

            for (int i = 0; i < listVertex.Count; i++)
            {
                if (vertex.ID == listVertex[i].IDParent &&
listVertex[i].Probability != default)
                {
                    cost += listVertex[i].Probability * listVertex[i].Cost;
                    CurrentBranchCost(listVertex[i], cost);
                    cost -= listVertex[i].Probability * listVertex[i].Cost;
```

```
                }
            }

            if (CheckIfChildExists(vertex))
                vertex.Value = cost;
        }

        /// <summary>
        /// метод , который проверяет, есть ли заданаая вершина в списке
вершин
        /// </summary>
        /// <param name="curver"></param>
        /// <returns>false, если в списке уже есть такая вершина, иначе
true</returns>
        private bool CheckIfChildExists(Vertex vertex)
        {
            for (int i = 0; i < listVertex.Count; i++)
                if (listVertex[i].Probability != default && vertex.ID ==
listVertex[i].IDParent)
                    return false;

            return true;
        }

        /// <summary>
        /// метод, который сортирует вершины, а затем
        /// выделяет самую опасную
        /// </summary>
        private void DrawDangerousMaximum()
        {
            Vertex vertexMax = listVertex[0];
            Vertex vertexMin = null;

            for (int i = 0; i < listVertex.Count; i++)
            {
                if (listVertex[i].Value != 0)
                {
                    vertexMin = listVertex[i];

                    for (int j = 0; j < listVertex.Count; j++)
                        if (listVertex[j].Value < vertexMin.Value &&
listVertex[j].Value != 0)
                            vertexMin = listVertex[j];

                    break;
                }
            }

            for (int i = 1; i < listVertex.Count; i++)
```

```
                if (listVertex[i].Value > vertexMax.Value)
                    vertexMax = listVertex[i];

            if (vertexMax != null)
            {
                Label label = new Label();

                label.Content = vertexMax.Value;
                label.Margin = new Thickness(vertexMax.X / 2 + Widht / 2,
vertexMax.Y + 20, 0, 0);
                label.VerticalAlignment = VerticalAlignment.Top;
                label.HorizontalAlignment = HorizontalAlignment.Left;
                label.Height = 40;
                label.Foreground = Brushes.Red;

                canvas.Children.Add(label);
                DrawLineRed(vertexMax);
            }

            if (vertexMin != null)
            {
                Label label1 = new Label();

                label1.Content = vertexMin.Value;
                label1.Margin = new Thickness(vertexMin.X/2 + Widht / 2,
vertexMin.Y + 20, 0, 0);
                label1.VerticalAlignment = VerticalAlignment.Top;
                label1.HorizontalAlignment = HorizontalAlignment.Left;
                label1.Height = 40;
                label1.Foreground = Brushes.Green;

                canvas.Children.Add(label1);

                DrawLineGreen(vertexMin);
            }
        }

        /// <summary>
        /// метод для отрисовки зелёной линии
        /// </summary>
        /// <param name="min"></param>
        private void DrawLineGreen(Vertex vertexMinimum)
        {
            for (int i = 0; i < listVertex.Count; i++)
            {
                if (vertexMinimum.IDParent == listVertex[i].ID)
                {
                    Line line = new Line();
```

```
                line.X1 = listVertex[i].X/ 2 + Widht / 2;
                line.Y1 = listVertex[i].Y;
                line.X2 = vertexMinimum.X / 2 + Widht / 2;
                line.Y2 = vertexMinimum.Y;
                line.Width = 12;
                line.Stroke = Brushes.Green;

                canvas.Children.Add(line);

                if (listVertex[i].Probability == default(double))
                    break;
                else
                    DrawLineGreen(listVertex[i]);

            }
        }
    }

    /// <summary>
    /// метод для отрисовки красной линии
    /// </summary>
    /// <param name="vertexMaximum"></param>
    private void DrawLineRed(Vertex vertexMaximum)
    {
        for (int i = 0; i < listVertex.Count; i++)
        {
            if (vertexMaximum.IDParent == listVertex[i].ID)
            {
                Line line = new Line();

                line.X1 = listVertex[i].X / 2 + Widht / 2;
                line.Y1 = listVertex[i].Y;
                line.X2 = vertexMaximum.X / 2 + Widht / 2;
                line.Y2 = vertexMaximum.Y;

                line.Stroke = Brushes.Red;

                canvas.Children.Add(line);

                if (listVertex[i].Probability == default(double))
                    break;
                else
                    DrawLineRed(listVertex[i]);

            }
        }
    }

    /// <summary>
```

```
        /// метод, который вычисляет координату y для точки при помощи
вадратного уравнения
        /// </summary>
        /// <param name="x"></param>
        /// <param name="radius"></param>
        /// <param name="center"></param>
        /// <returns>координата y</returns>
        /// <exception cref="Exception"></exception>
        public double FindYCoordinate(double x, double radius, Point center)
        {
            double a = 1;
            double b = -2 * center.Y;
            double c = -radius * radius + (x - center.X) * (x - center.X) +
center.Y * center.Y;
            double d = (b * b - 4 * a * c);

            if (d < 0)
                throw new Exception("There's no such point!");

            return (-b + Math.Sqrt(d)) / 2 * a;
        }

        /// <summary>
        /// метод рисует линию и вершины из списка выбранных вершин
        /// </summary>
        private void DrawLine()
        {
            canvas.Children.Clear();

            for (int i = 0; i < K - 1; i++)
            {
                Line line = new Line();

                double oldX = center.X - radius + radius / K * i;
                double currentX = center.X - radius + radius / K * (i + 1);

                line.X1 = oldX;
                line.X2 = currentX;
                line.Y1 = FindYCoordinate(oldX, radius, center);
                line.Y2 = FindYCoordinate(currentX, radius, center);
                line.Stroke = Brushes.Black;

                canvas.Children.Add(line);
            }

            for (int i = 0; i < listSelected.Count; i++)
            {
                if (listSelected[i].Influence != default &&
listSelected[i].Probability != default  && listSelected[i].Status==1)
```

```
                {
                    Ellipse ellipse = new Ellipse();

                    listSelected[i].point.X = 425 *
listSelected[i].Probability + 75;
                    listSelected[i].point.Y = -350 *
listSelected[i].Influence + 400;

                    ellipse.Height = 12;
                    ellipse.Width = 12;
                    ellipse.StrokeThickness = 3;

                    double m = Math.Sqrt((listSelected[i].point.X - center.X)
* (listSelected[i].point.X - center.X) +
                                         (listSelected[i].point.Y - center.Y)
* (listSelected[i].point.Y - center.Y));

                    if (m < radius)
                    {
                        ellipse.Fill = Brushes.Red;
                        ellipse.Stroke = Brushes.Red;
                    }
                    else
                    {
                        ellipse.Fill = Brushes.Green;
                        ellipse.Stroke = Brushes.Green;
                    }

                    ellipse.VerticalAlignment = VerticalAlignment.Top;
                    ellipse.HorizontalAlignment = HorizontalAlignment.Left;
                    ellipse.Margin = new Thickness(listSelected[i].point.X,
listSelected[i].point.Y, 0, 0);

                    canvas.Children.Add(ellipse);
                }
            }
        }

        /// <summary>
        /// метод закрывает окно при нажатии кнопки
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Close_Click(object sender, RoutedEventArgs e)
        {
            AdministratorGraphic graphicWindow = new
AdministratorGraphic(_project);
            Close();
            graphicWindow.Show();
```

```
        }

        private void OpenDocument_Click(object sender, RoutedEventArgs e)
        {
            wordApplication.Visible = true;
        }
    }
}
```

2. Текст базы данных

2.1. Таблица Risks

```
SET IDENTITY_INSERT [dbo].[Risks] ON

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (11, N'Риск задержки сроков из-за организационных изменений', N'Изменения требований', N'Срыв сроков контракта, штрафные санкции', N'Продление сроков', N'default')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (12, N'Риск провала проекта из-за плохого анализа технологий в начале проекта', N'Низкая квалификация специалистов команды внедрения', N'Неудачный проект в целом', N'Смена персонала', N'default')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (13, N'Риск увеличивающейся сложности из-за недочета масштабов проекта', N'Изменения требований', N'Неудачный проект в целом', N'Увеличение трудовых часов', N'default')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (14, N'Риск высоких эксплуатационных издержек ', N'Низкая квалификация специалистов команды внедрения', N'Высокая стоимость для потребителей и как следствие неконкурентоспособность', N'Уменьшение затрат на разработку', N'default')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (15, N'Риск несоблюдения сроков и бюджета вследствие влияния других рисков', N'Изменения требований', N'Срыв сроков контракта, штрафные санкции, выход из бюджета', N'Устранение причин рисков, влияющих на график разработки', N'default')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (16, N'Риск текучести кадров', N'Низкая квалификация специалистов команды внедрения', N'Низкая производительность  при вводе новых участников в проект', N'Устранение причин изменения состава участников проекта', N'default')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (17, N'Риск неприятия системы коллективом вследствие корпоративной культуры', N'Изменения требований', N'Сопротивление при внедрении системы', N'Привлечение заинтересованных лиц', N'default')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (18, N'Риск выбора партнера/консультанта из-за недочета всех параметров', N'Изменения требований', N'Трата ресурсов и неполучение желаемого результата= неудачный проект', N'Привлечение нескольких специалистов для рассмотрения параметров', N'Desktop Application')
```

```
INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (19, N'Риск выбора продукта из-за недочета всех параметров', N'Низкая квалификация специалистов команды внедрения', N'Трата ресурсов и неполучение желаемого результата = неудачный проект', N'Привлечение нескольких специалистов для тестирования продукта', N'Desktop Application')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (20, N'Риск неверно реализованной архитектуры из-за недочета всех пареметров', N'Опаздывание сторонних поставщиков', N'Трата ресурсов и неполучение желаемого результата = неудачный проект', N'Привлечение сторонних специалистов для составления мнения об архитектуре системы', N'Web Application')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (21, N'Риск несоответствия долгосрочным целям бизнеса из-за отсутствия стратегии ', N'Опаздывание сторонних поставщиков', N'Непригодность внедренной системы в будущем', N'Составление стратегии разработки', N'Web Application')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (22, N'Риск выбора неправильного оборудования или платформы для разработки', N'Опаздывание сторонних поставщиков', N'Невозможность дальнейшего масштабирования системы, возможное снижение производительности', N'Привлечение специалистов', N'Mobile Application')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (23, N'Риск подбора неправильной проектной команды', N'Неполная проработка требований', N'Результат всего проекта ставится под сомнение, частые конфликты внутри команды ', N'Смена команды', N'Mobile Application')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (24, N'Риск непрофессиональной реализации системы', N'Неполная проработка требований', N'Качество реализованной системы может не соответствовать заявленному в проектной документации', N'Привлечение специалистов для стороннего мнения о корректности работы; проведение испытаний', N'Mobile Application')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (25, N'Риск недостаточности тестирования реализованной системы', N'Неполная проработка требований', N'Эксплуатация системы может сопровождаться репутационным риском компании, и непредвиденными расходами на доработку и техническую поддержку.', N'Составление подробного плана тестирования программного продукта', N'Mobile Application')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (26, N'Риск несоответствия продукта требуемым нормам качества из-за несоблюдения технолог. стандартов',
```

```
N'Неполная проработка требований', N'Несоответствие стандартам качества продукта', N'Составление плана разработки согласно техн. стандартам', N'Web Application')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (27, N'Риск неправильного распределения полномочий', N'Низкая квалификация специалистов', N'Срыв сроков', N'Распределение полномочий путем установления возможностей разработчиков ', N'default')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (28, N'Риск нарушения баланса интересов участников', N'Организационные риски', N'Скрытый или явный саботаж со стороны отдельных участников', N'Формирование организационных структур управления проектом, в кооторых обеспечено представительство всех заинтересованных сторон на всех уровнях управления', N'default')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (29, N'Риск забастовки персонал', N'Организационные риски', N'Усиление напряжения в коллективе', N'Договор с профсоюзами, Компенсационные выплаты', N'default')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (30, N'Риск недостаточной поддержки со стороны руководства заказчика', N'Организационные риски', N'Увеличение сроков исполнения работ вплоть до их остановки', N'Выделение ответственного со стороны высшего руковдства заказчика, контролирующего сроки и качество работ по проекту', N'default')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (31, N'Риск ошибочного выбора программной или технической платформы', N'Неполная проработка требований', N'Высокая стоимость владения', N'Проведение выбора платформ на тендерной основе, сравнение платформ и обоснование выбора с точки зрения стоимости владения', N'Mobile Application')

INSERT INTO [dbo].[Risks] ([Id], [RiskName], [Source], [Effects], [PossibleSolution], [Type]) VALUES (32, N'Риск потери знаний о системе', N'Низкая квалификация специалистов', N'Отсутствие возможности смены подрядчика или сопровождения собственными силами', N'Привлечение внутренних специалистов к участию в проекте', N'default')

SET IDENTITY_INSERT [dbo].[Risks] OFF
```

2.2. Таблица Users

```
SET IDENTITY_INSERT [dbo].[Users] ON
```

```
INSERT INTO [dbo].[Users] ([Id], [Name], [Login], [Password],
[Position]) VALUES (1, N'Mr Tester', N'Test', N'qwerty',
N'RiskManager')

INSERT INTO [dbo].[Users] ([Id], [Name], [Login], [Password],
[Position]) VALUES (2, N'Aleksandra Karpova', N'Alex', N'1701',
N'MainManager')

INSERT INTO [dbo].[Users] ([Id], [Name], [Login], [Password],
[Position]) VALUES (3, N'Maria ', N'Mary', N'1234', N'ProjectManager')

SET IDENTITY_INSERT [dbo].[Users] OFF
```

Лист регистрации изменений									
Изм.	Номера листов				Всего листов (страниц) в документе	№ документа	Входящий № сопроводит. докум. и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.13-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата