Министерство образования и науки РФ

ФГАОУ ВО «Уральский федеральный университет   
имени первого Президента России Б.Н. Ельцина»

Институт новых материалов и металлургии

Кафедра «Теплофизика и информатика в металлургии»

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_

Члены комиссии:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись расшифровка подписи

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись расшифровка подписи

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2023 г.

**Разработка графического приложения «Калькулятор для оптимизации доходов и расходов на производстве»**

**«Введение в технологию разработки программного обеспечения»**

Направление 09.03.02 «Информационные системы и технологии»   
(уровень бакалавриата)

Образовательная программа 09.03.02/03.01   
«Информационные системы и технологии в металлургии»

Руководитель

ст. преподаватель, к.т.н. А.С. Истомин

должность, звание подпись расшифровка подписи

Нормоконтролер

ст. преподаватель, к.т.н. А.С. Истомин

должность, звание подпись расшифровка подписи

Студенты:

НМТ-213901 О. А. Климов

номер группы подпись расшифровка подписи

НМТ-213901 Н.Е.Якимов

подпись расшифровка подписи

Екатеринбург

2023

**РЕФЕРАТ**

Пояснительная записка изложена на 45 листах и содержит 21 рисунк, 3 приложения.

РАСЧЕТ ДОХОДА И РАСХОДОВ ВО ВРЕМЯ ПРОИЗВОДСТВА, ПРОГРАММИРОВАНИЕ, ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, АРХИТЕКТУРА, ИНТЕРФЕЙС, VISUAL STUDIO C#, ИНСТАЛЛЯЦИЯ.

Проект по модулю посвящен разработке программного обеспечения для автоматического расчета оптимальных расходов вовремя производство при определенном доходе.

Отражены основные этапы разработки программного обеспечения: постановка задачи, реализация тестового варианта расчета в электронных таблицах Microsoft Excel; проектирование и реализация программного средства – математической библиотеки и пользовательского интерфейса; разработка системы автоматизированного тестирования математической библиотеки; создание справочной документации; подготовка дистрибутива. Размещение исходного программного кода выполнено в системе удаленного контроля версий Bitbucket.

Основными функциями программного обеспечения является расчет: процента желаемого дохода, суммы продаж, объема продаж, прибыли, себестоимости продукта, цены товара, безубыточности в количестве, безубыточности в деньгах, переменных затрат в процентах от постоянных затрат.

Основные конечные пользователи программного обеспечения – студенты вузов.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc137646243)

[1 ПОСТАНОВКА ЗАДАЧИ, ПРОВЕРКА КОРРЕКТНОСТИ АЛГОРИТМА РАСЧЕТА И ОРГАНИЗАЦИЯ РАБОТЫ НАД ПРОЕКТОМ В СИСТЕМЕ AZURE DEVOPS 5](#_Toc137646244)

[1.1Физическая постановка задачи 5](#_Toc137646245)

[1.2 Математическая модель задачи 5](#_Toc137646246)

[1.3 Создание тестового варианта расчета в электронных таблицах Microsoft Excel 6](#_Toc137646247)

[1.4 Постановки задач для выполнения отдельных этапов проекта в системе Azure DevOps 6](#_Toc137646248)

[2 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 10](#_Toc137646249)

[2.1 Разработка архитектуры системы 10](#_Toc137646250)

[2.2 Создание программного обеспечения в системе управления версиями GitHub 10](#_Toc137646251)

[2.3 Разработка математической библиотеки 11](#_Toc137646252)

[2.4 Реализация пользовательского интерфейса 12](#_Toc137646253)

[2.5 Обработка исключительных ситуаций 16](#_Toc137646254)

[2.6 Создание справочной помощи 17](#_Toc137646255)

[2.7 Создание дистрибутива 18](#_Toc137646256)

[3 РАЗРАБОТКА СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ МАТЕМАТИЧЕСКОЙ БИБЛИОТЕКИ 20](#_Toc137646257)

[3.1 Проектирование системы 20](#_Toc137646258)

[3.2 Реализация системы 20](#_Toc137646259)

[4 ОПИСАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 21](#_Toc137646260)

[4.1 Установка и настройка программного средства 21](#_Toc137646261)

[4.2 Функциональные возможности программного продукта 21](#_Toc137646262)

[ЗАКЛЮЧЕНИЕ 22](#_Toc137646263)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 23](#_Toc137646264)

[ПРИЛОЖЕНИЕ А 24](#_Toc137646265)

[ПРИЛОЖЕНИЕ Б 28](#_Toc137646266)

[ПРИЛОЖЕНИЕ В 36](#_Toc137646267)

# ВВЕДЕНИЕ

Курсовая работа посвящена разработке программного обеспечения для руководителей бухгалтерии с использованием современных методологий разработки, а также программных продуктов. Данная тема является актуальной, поскольку направлена на решение проблемы ошибок в итоговых подсчетах и вследствие чего, происходят большие траты времени. В современном мире оптимизация финансов уже заняло своё немаловажное место в жизни людей. Однако для того, чтобы люди могли не совершать ошибки при подсчетах своих финансов – необходимо оптимизировать этот процесс расчета.

Приложение позволяет автоматизировать процесс постройки плана для дальнейшей оптимизации доходов и расходов на предприятии, что значительно облегчает работу главным бухгалтерам.

Приложение представляет собой графический «калькулятор» с удобным и понятным, для пользователя, интерфейсом.

# 1 ПОСТАНОВКА ЗАДАЧИ, ПРОВЕРКА КОРРЕКТНОСТИ АЛГОРИТМА РАСЧЕТА И ОРГАНИЗАЦИЯ РАБОТЫ НАД ПРОЕКТОМ В СИСТЕМЕ AZURE DEVOPS

## Физическая постановка задачи

Создать графическое приложение в среде разработке VS 2022, которое с помощью вводимых данных и формул из библиотеки, будет выводить данные, которые помогут предприятию оптимизировать бюджет. Программа должна работать быстро и без ошибок, для экономии времени. Для наглядности в приложении должны создаваться отчеты в PDF формате и строиться графики.

## 1.2 Математическая модель задачи

Дано:

процент переменных затрат (ppz);

постоянные затраты (poz);

цена продукции (cp);

желаемая прибыль (zp);

объем производства (op).

Найти:

процент желаемого дохода(pzd);

сумму продаж (sp);

объем продаж (opr);

прибыль (pr);

себестоимость продукта (spr);

цену товара (ct);

безубыточность в количестве (bvk);

безубыточность в деньгах (bvd);

переменные затраты (pez)

оптимальный процент переменных затрат (oppz).

Решение:

Чтобы найти процент желаемого дохода нужно разделить желаемую прибыль на цену продукции (pzd = zp / cp).

Чтобы найти объем продаж нужно постоянные затраты сложить с желаемым доходом и поделить эту сумму на разность цены продукции и переменных затрат (opr = (poz + zp) / (cp - pez)).

Чтобы найти сумму продаж нужно цену продукции умножить на объем продаж (sp = cp \* opr).

Чтобы найти прибыль нужно из цены продукции вычесть переменные затраты, эту разность умножить на объем продаж и вычесть постоянные затраты (pr = (cp - pez) \* opr - poz).

Чтобы найти себестоимость продукции нужно к переменным затратам прибавить частное от постоянных затрат и объема производства (pez + poz / op).

Чтобы найти цену товара нужно себестоимость продукции умножить на сумму единицы и частного от желаемого дохода в процентах и 100 (ct = spr \* (1 + pzd / 100))

Чтобы найти безубыточность в количестве нужно постоянные затраты поделить на разность цены продукции и переменных затрат (bvk = poz / (cp - pez)).

Чтобы найти безубыточность в деньгах нужно цену продукции умножить на безубыточность в количестве (bvd = cp \* bvk).

Чтобы найти переменные затраты нужно переменные затраты в процентах поделить на 100 и умножить на цену продукции (pez = ppz / 100 \* cp).

Чтобы найти оптимальные переменные затраты в процентах нужно переменные затраты умножить на 100 и поделить на цену продукции (oppz = pez \* 100 / cp).

## 1.3 Создание тестового варианта расчета в электронных таблицах Microsoft Excel

В приложении Microsoft Excel были созданы 2 таблицы (рисунок 1.1): в первой таблице подавались входные данные; во второй таблице выводился результат расчетов по формулам, указанных в пункте 1.2 математическая модель задачи.

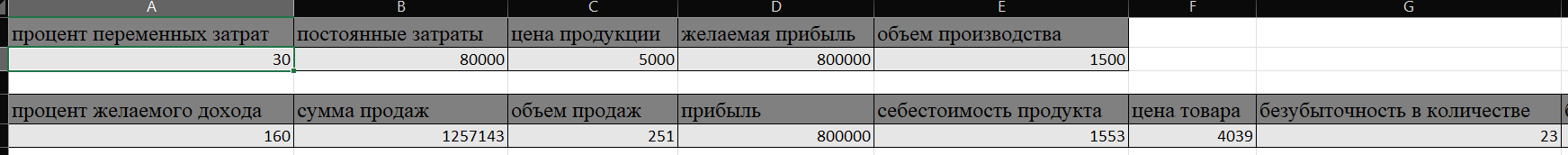


Рисунок 1.1 – Тестовый вариант расчета в Microsoft Excel

## 1.4 Постановки задач для выполнения отдельных этапов проекта в системе Azure DevOps

Azure DevOps Server (ранее Team Foundation Server, сокр. TFS) — продукт корпорации. Microsoft, представляющий собой комплексное решение, объединяющее в себе систему управления версиями, сбор данных, построение отчётов, отслеживание статусов и изменений по проекту и предназначенное для совместной работы над проектами по разработке программного обеспечения. Продукт доступен в виде отдельного приложения, сходного по функциям с облачным сервисом.

Был создан новый проект «CP\_VTRPO» (Рисунок 1.2).

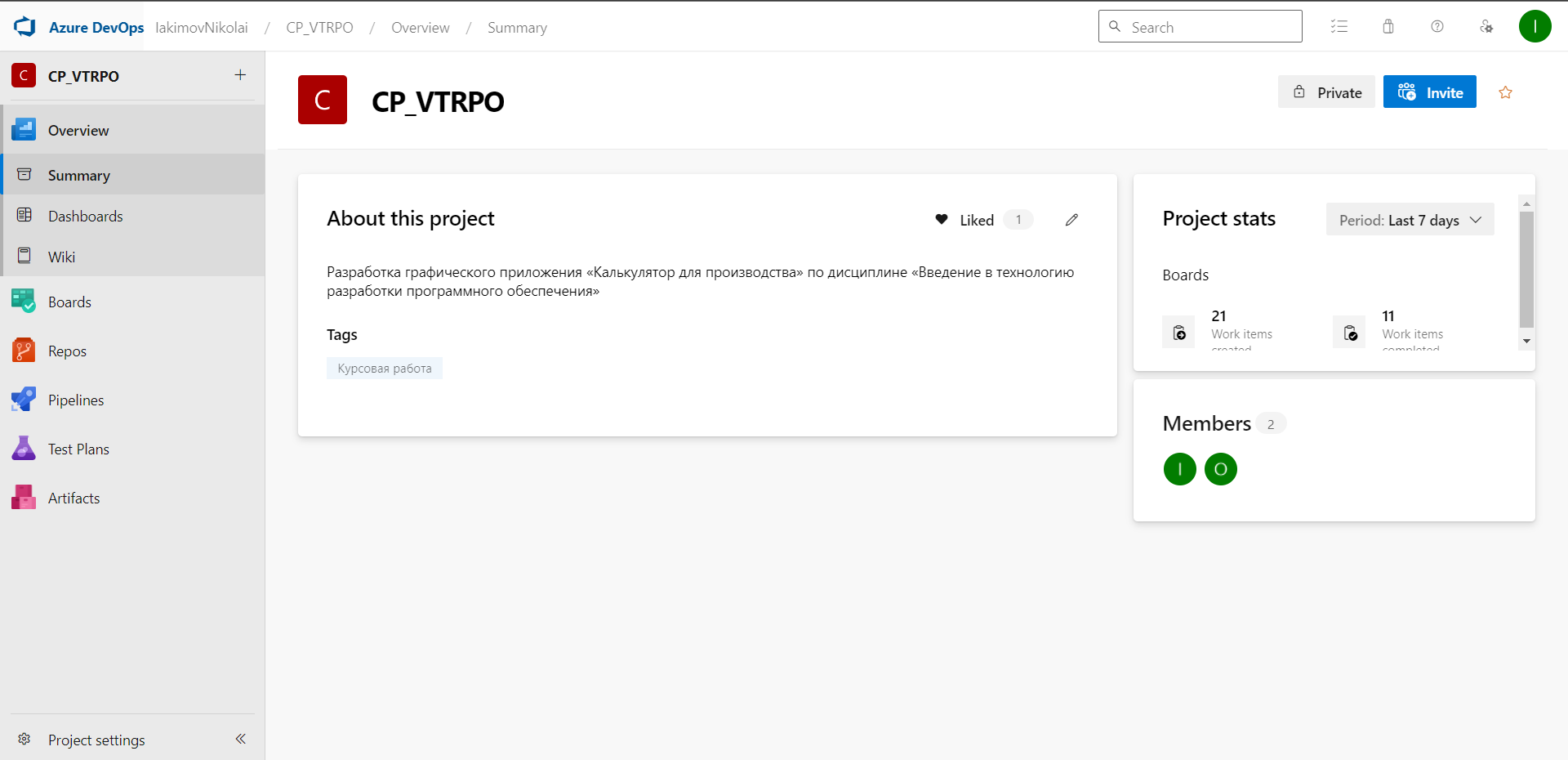


Рисунок 1.2 – главная страница проекта

В настройках была создана команда «CP\_VTRPO Team» (рисунок 1.3).

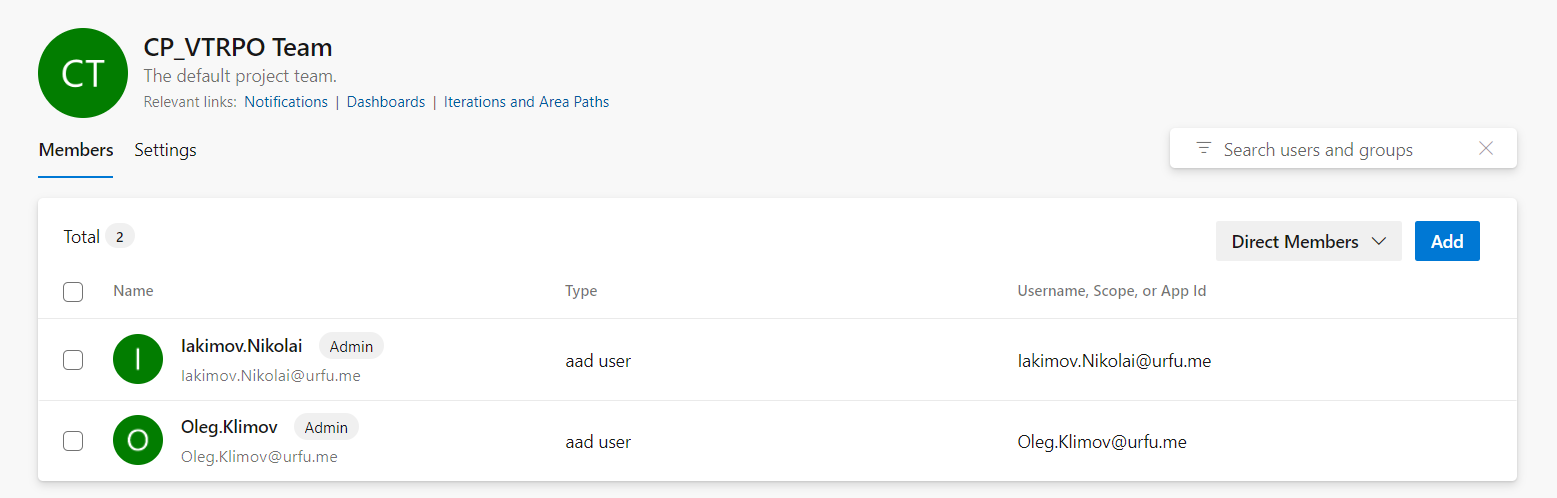


Рисунок 1.3 – участники команды CP\_VTRPO Team

В настройках команды были созданы 4 итерации на каждый спринт (рисунок 1.4). После чего были заполнены каждый из этих спринтов. В каждом спринте есть задачи, которые присвоены к определенному участнику (рисунок 1.5). На главной доске (рисунок 1.6) можно увидеть состояние спринтов (to do, doing, done).

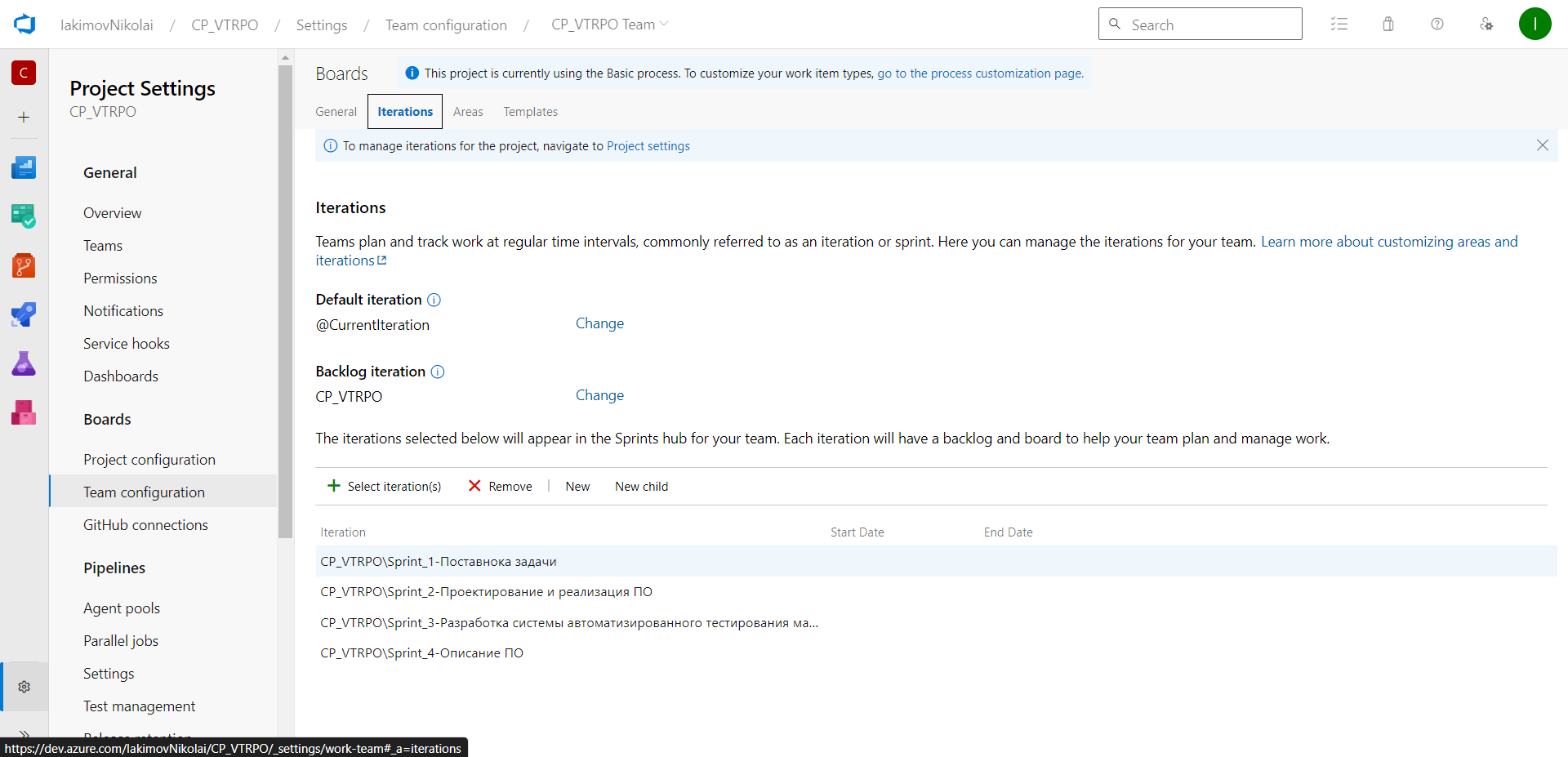


Рисунок 1.4 – создание спринтов

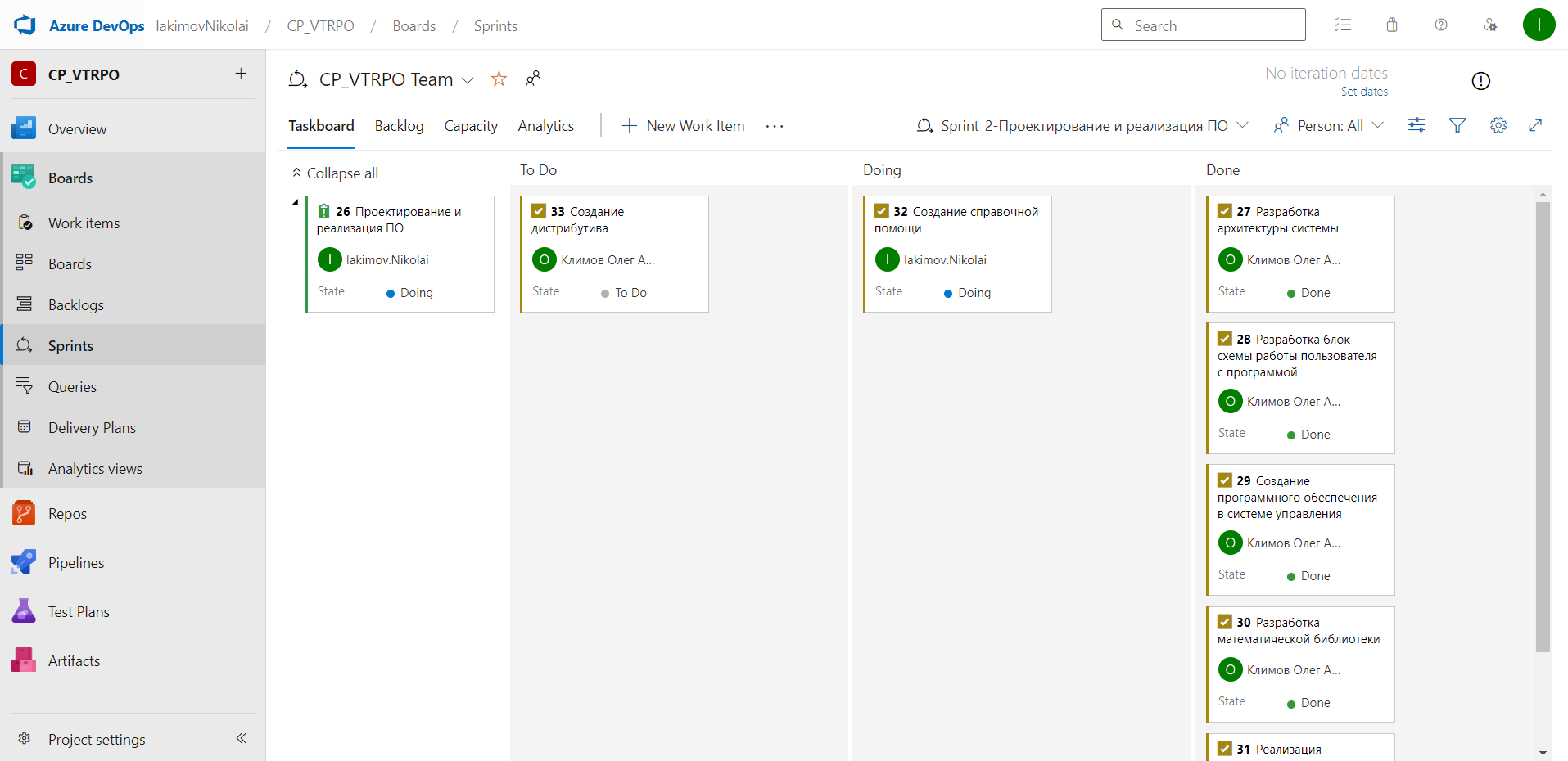


Рисунок 1.5 – Наполнение спринтов задачами

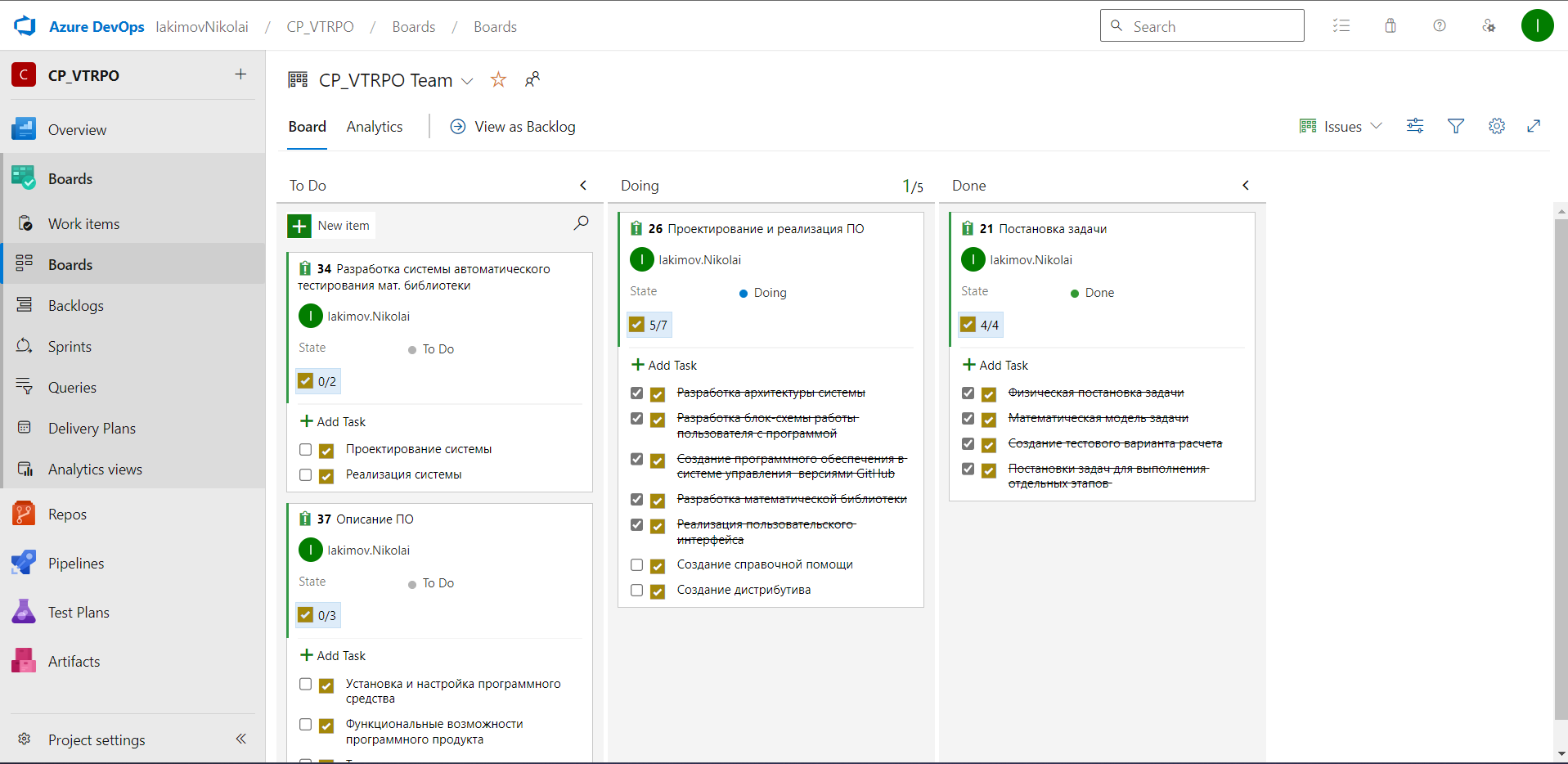


Рисунок 1.6 – главная доска со спринтами и их подзадачами

Также в Dashboards были добавлены виджеты для визуализации процесса работы участников в проекте (рисунок 1.7).

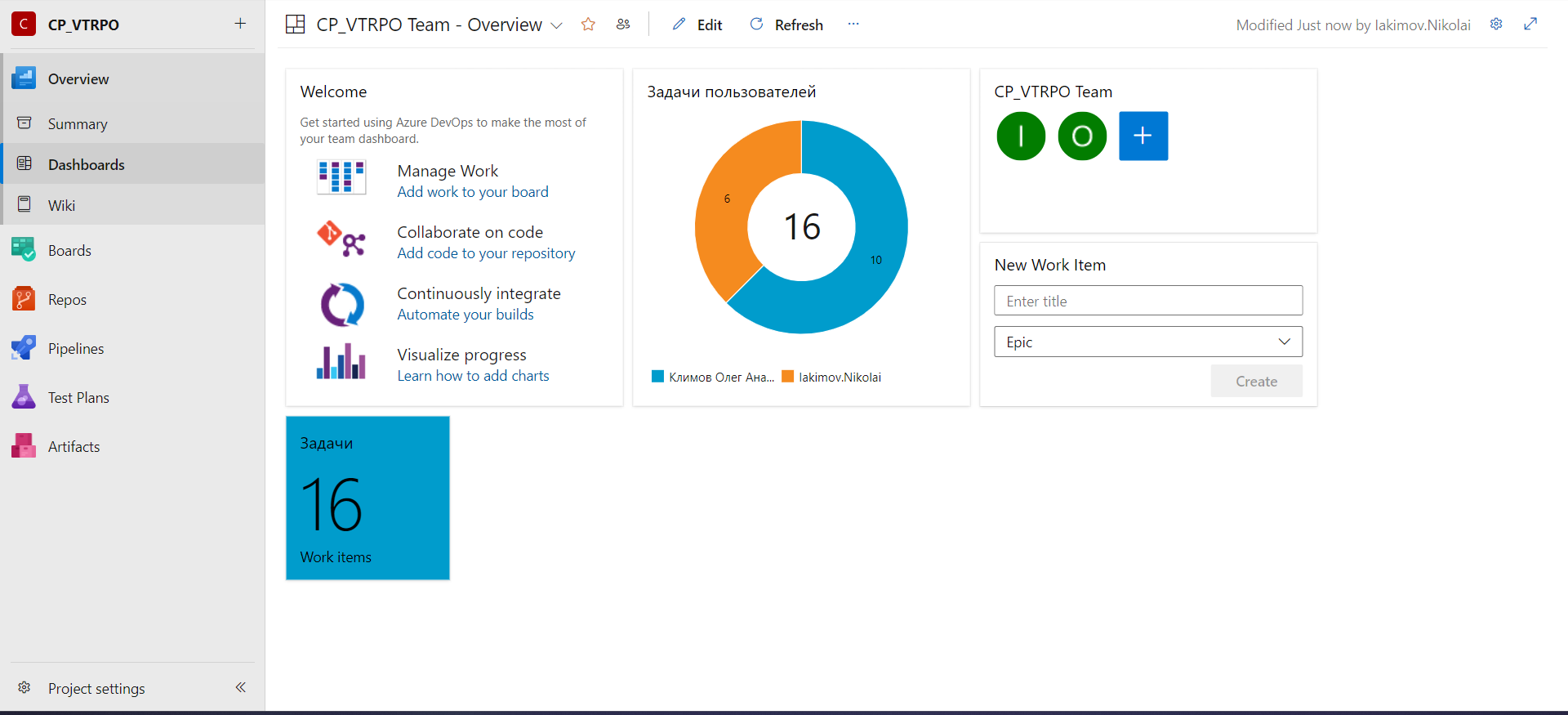


Рисунок 1.7 – Dashboards Overview

Ссылка на проект: [Summary - Overview (azure.com)](https://dev.azure.com/IakimovNikolai/CP_VTRPO)

# 2 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 2.1 Разработка архитектуры системы

На рисунке 2.1 показана архитектура системы работы пользователя с приложением. Пользователь взаимодействует с интерфейсом приложения, после этого с помощью view model выполняется взаимодействие с формулами из библиотеки классов. В итоге результат отображается пользователю на интерфейсе приложения.

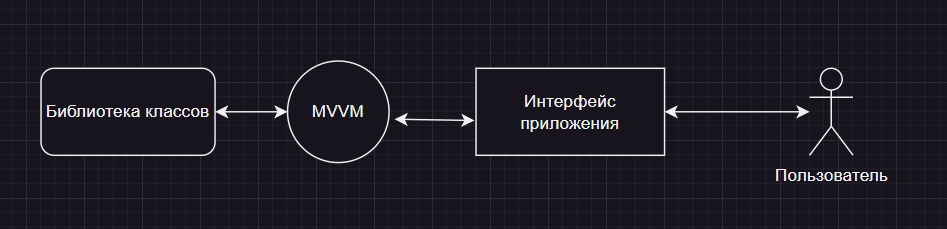


Рисунок 2.1 – блок-схема взаимодействия пользователя с программой

## 2.2 Создание программного обеспечения в системе управления версиями GitHub

GitHub — это служба размещения в Интернете репозиториев Git, которые используются для хранения содержимого docs.microsoft.com. Наш Прикладной программист загружает новейшие версии проекта(приложения) на GitHub, к которому имеет доступ вся команда.

На рисунке 2.2 представлен фрагмент окна сервиса GitHub со структурой папок, необходимых в рамках реализации проекта.

Для выгрузки проекта в GitHub использовалось приложение (рисунок 2.3) GitHub Desktop. Это программа, которая дублирует функциональность сайта github.com, но при этом работает локально на компьютере разработчика.

Ссылка на внешний репозиторий с проектом: <https://github.com/GoldFar228/formulas> .

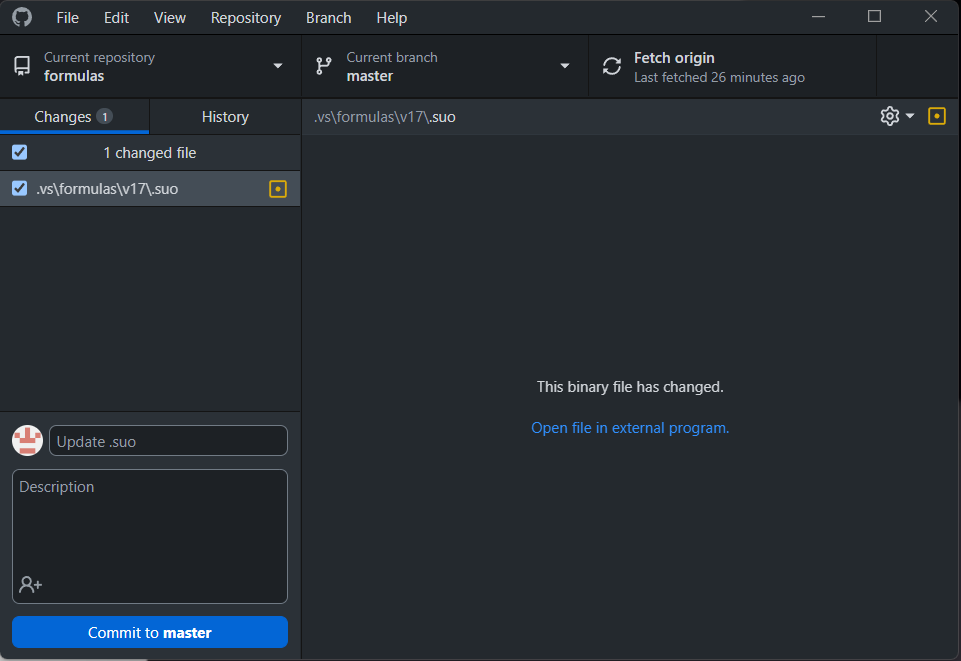


Рисунок 2.2 – окно приложения GitHub Desktop

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 2.2 - фрагмент окна cервиса GitHub

## 2.3 Разработка математической библиотеки

Была создана библиотека классов (ПРИЛОЖЕНИЕ А), в которой были расписаны все нужные формулы (рисунок 2.3).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

Рисунок 2.3 – фрагмент кода из библиотеки классов

## 2.4 Реализация пользовательского интерфейса

Для создания пользовательского интерфейса использовался метод MVVM – взаимодействие пользователя с приложением через view model. Были добавлены поля для ввода, поля для ответов, кнопка «создать отчет», графики «CartesianChart» (ПРИЛОЖЕНИЕ Б). Для удобства пользователя, рядом с полями вывода ответов, расположены подписи «label» (рисунки 2.4 – 2.6).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 2.4 - Интерфейс приложения

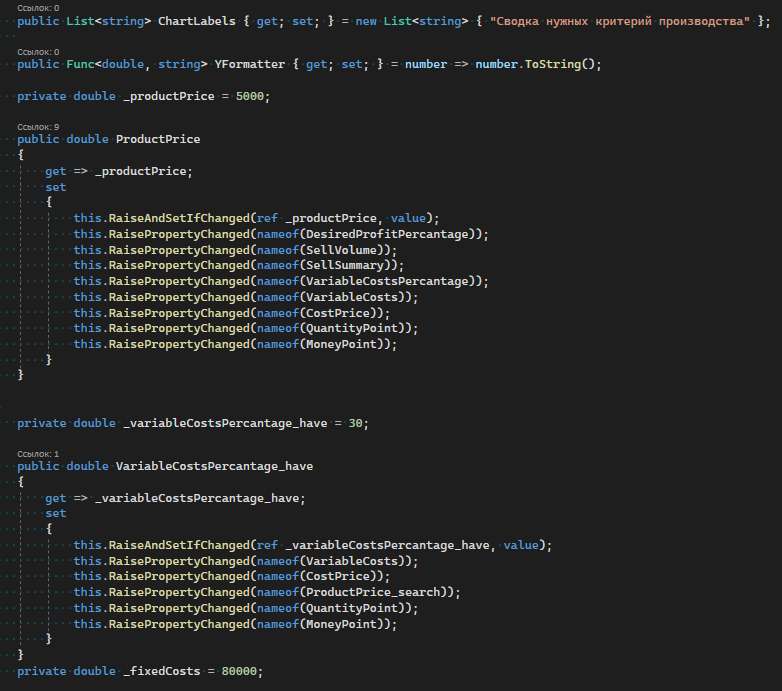


Рисунок 2.5 – фрагмент кода для создания интерфейса в классе TaskViewModel

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 2.6 – код для постройки диаграмм в «CartesianChart»

## 2.5 Обработка исключительных ситуаций

Была создана функция ThrowArgumentException, благодаря которой в ответе не будет отрицательных результатов, вместо этого, поля останутся пустыми (рисунки 2.7 – 2.8).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

Рисунок 2.7 – фрагмент кода с использованием функции ThrowArgumentException

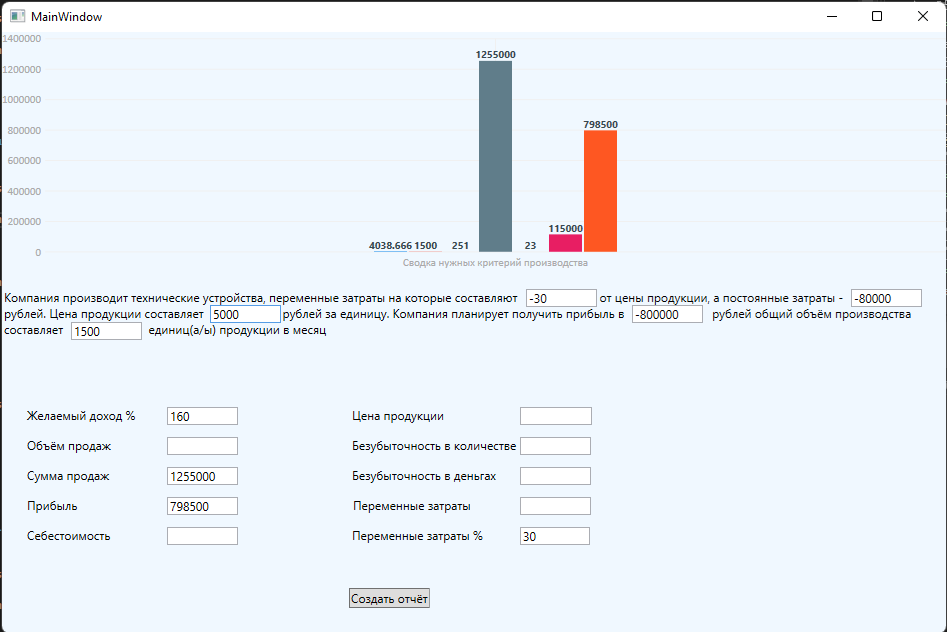


Рисунок 2.8 – демонстрация работы функции при отрицательных значениях в полях для ввода

## 2.6 Создание справочной помощи

После того, как пользователь установит приложение, на рабочем столе, вместе с приложением, появится ярлык на файл (рисунок 2.9) с расширением PDF. Это файл со справочной помощью, где кратко описывается то, как приложение работает и как им нужно пользоваться. Этот файл, нужен для того, чтобы пользователь не тратил время на освоение приложения.

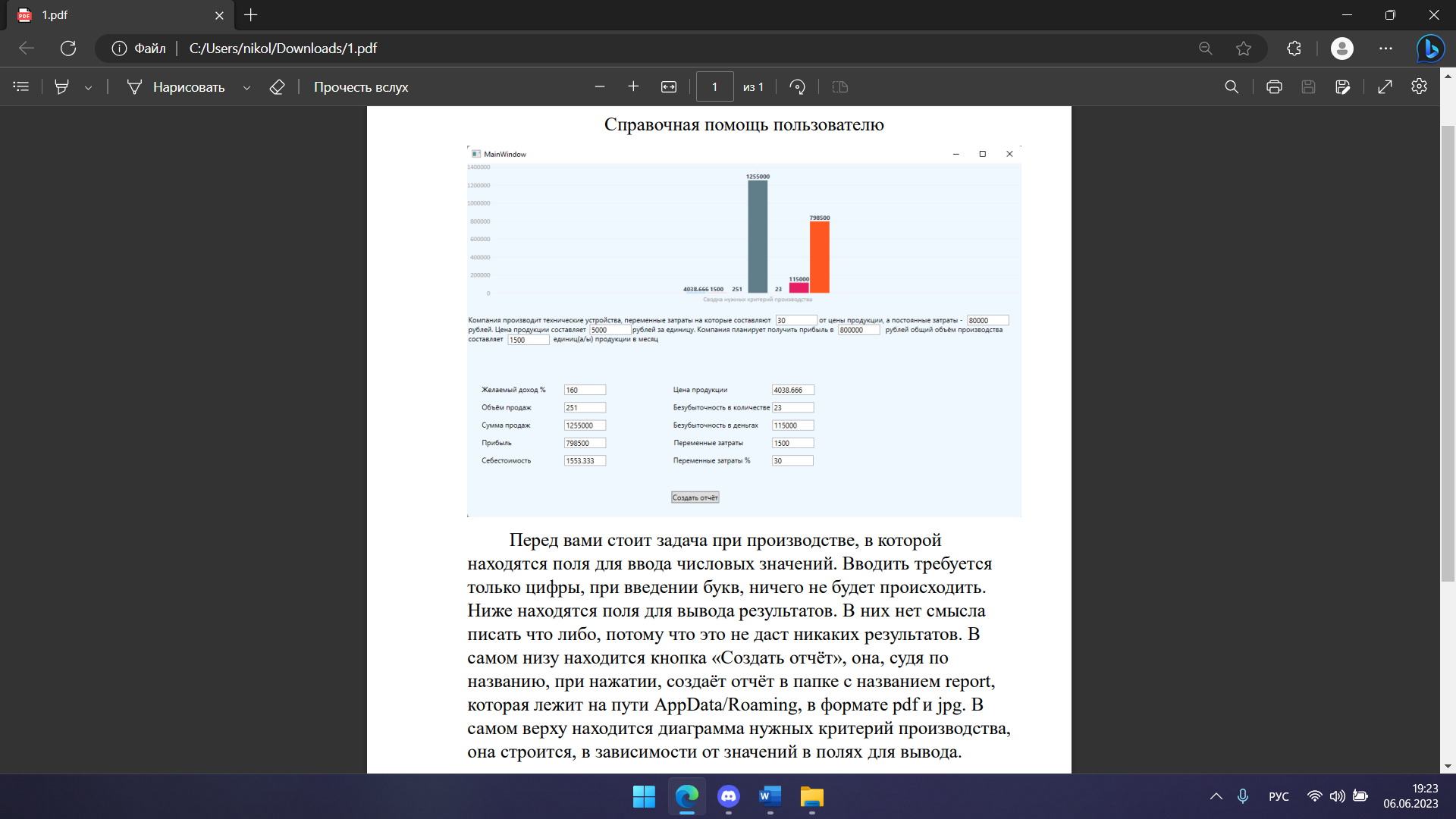


Рисунок 2.9 – справочный PDF-файл

## 2.7 Создание дистрибутива

На рисунке 2.10 показано, то, как выглядит установка приложения через дистрибутив. Для создания дистрибутива было использовано расширение Microsoft Visual Studio Instiller Projects 2022. В папке application folder хранятся файлы (рисунок 2.11) Table.frx, основные входные файлы from WPF, элементы для публикации from WPF. Они нужны для того, чтобы при установке приложения setup файл загрузил файлы, чтобы приложение корректно работало у любого пользователя.

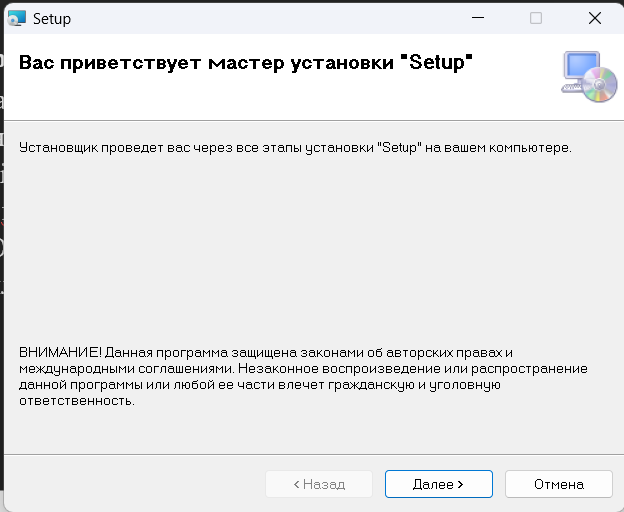


Рисунок 2.10 – установщик приложения



Рисунок 2.11 – нужные файлы для создания дистрибутива

# 3 РАЗРАБОТКА СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ МАТЕМАТИЧЕСКОЙ БИБЛИОТЕКИ

## 3.1 Проектирование системы

Так как в нашем приложении содержится большое количество связанных между собой формул, нужно было проверить корректность их работ в программе. Чтобы это сделать нужно было создать в VS 2022 «тестовый проект NUnit» (рисунок 3.1).

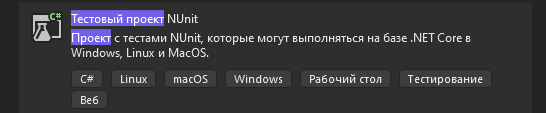
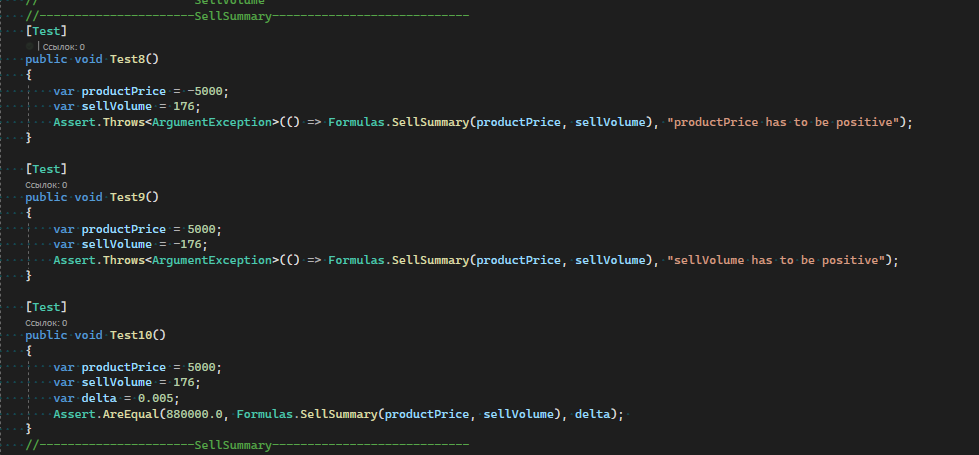


Рисунок 3.1 – создание тестового проекта в VS 2022

## 3.2 Реализация системы

Для проверки формул на корректность ввода данных и на правильность вычислений был создан проект «Tests» (ПРИЛОЖЕНИЕ В). В этом проекте происходит проверка на отрицательность и проверка равенство между планируемого числа и числа, которое находится при помощи формулы (рисунок 3.2). Благодаря этому тестовому проекту было доказано, что изначальные формулы работают корректно.

  
Рисунок 3.2 – фрагмент кода проекта «Tests»

# 4 ОПИСАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 4.1 Установка и настройка программного средства

Установка происходит через дистрибутив с расширением msi — это расширение файла, которое применяется к файлам базы данных, используемым установщиком Microsoft Windows (MSI). Они содержат информацию о приложении, разделенном на функции и компоненты, и каждый компонент может содержать файлы, данные реестра, ярлыки и так далее. В установщике можно выбрать расположение установки приложения (рисунок 4.1). Также можно установить для одного локального пользователя или для всех, которые есть в системе.

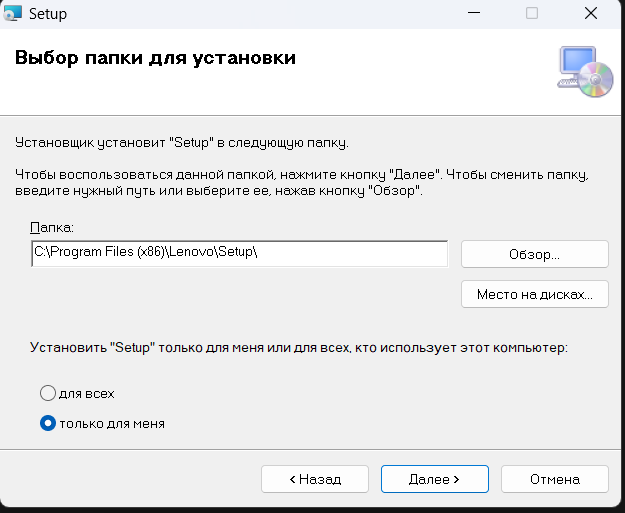


Рисунок 4.1 – Выбор папки для установки

## 4.2 Функциональные возможности программного продукта

В силу простого интерфейса, программы пользователь быстро сможет разобраться в том, как работает программа. Пользователю надо лишь правильно ввезти 5 значений, после чего программа автоматически выдаст значения, интересующие пользователя, в полях для вывода. Программа быстро вычисляет значения по формулам, тем самым пользователь экономит большое количество времени, нежели бы он считал вручную.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было создано графическое приложение, которое позволяет автоматизировать процесс постройки плана для дальнейшей оптимизации доходов и расходов на предприятии, что значительно облегчает работу главным бухгалтерам. Были изучены различные подходы к созданию: WPF-приложений на основе MVVM, диаграмм, дистрибутивов. Была изучена работа тестового проекта NUnit.

Система имеет следующие преимущества:

* упрощает работу главных бухгалтеров;
* минимизирует количество ошибок при расчете;
* позволяет быстро получать информацию об оптимизации финансов;

Таким образом, созданное приложение является эффективным инструментом для главных бухгалтеров. Она может быть использована на любых предприятиях для оптимизации финансового результата.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Коцюба И.Ю., Чунаев А.В., Шиков А.Н. Основы проектирования информационных систем. Учебное пособие. – СПб: Университет ИТМО, 2015. – 206 с.
2. Горлушкина Н.Н. Системный анализ и моделирование информационных процессов и систем. Учебное пособие. – СПб: Университет ИТМО, 2016. – 120 с.
3. Грекул В.И., Денищенко Г.Н., Коровкина Н.Л. – Проектирование информационных систем: учебное пособие / 2-е изд., испр. – М.: Интернет-Университет информационных технологий (ИНТУИТ.РУ): БИНОМ. Лаборатория знаний, 2010. – 299 c.
4. Троелсен Э. Язык программирования C# 5.0 и платформа .NET 4.5. 6-е изд. – М.: ООО «И.Д. Вильямс», 2015. – 1312 с.
5. Флёнов М.Е. Библия С#. 2-е изд. – СПб.: БХВ-Петербург, 2011. – 560 с.
6. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C#. Практическое руководство - Санкт-Петербург, 2012. – 928 с.
7. Шилдт Г. Полное руководство С# 4.0 [Текст]: учебное пособие / Г. Шилдт — пер. с англ. Берштейн И. В. — Москва: Вильямс, 2012.— 1051 с.
8. Хайруллин Р.С. Программирование на C#: учебное пособие / Р.С. Хайруллин. – Казань: Изд-во Казанск. гос. архитект.-строит. ун-та, 2017. – 153 с.
9. Фаронов В. В. Создание приложений с помощью C#: Руководство программиста [Текст]: учебник / В. В. Фаронов — Москва: Эксмо, 2008. — 576 с.
10. Лавров В.В. Методические указания к выполнению курсовой работы для студентов бакалавриата, обучающихся по направлению «Информационные системы и технологии» и студентов магистратуры, обучающихся по программе «Информационные системы в металлургии» / В.В. Лавров, К.А. Щипанов, А.А. Бурыкин – Екатеринбург: УрФУ, 2014. – 49 с.
11. Лошкарев Н.Б. Указания к оформлению дипломных и курсовых проектов и работ: методические указания / Н.Б. Лошкарев, А.Н. Лошкарев, Л.А. Зайнуллин. – Екатеринбург: УГТУ–УПИ, 2007. – 49 с.

# ПРИЛОЖЕНИЕ А

Фрагменты листинга программного кода для создания библиотеки классов

namespace formulas

{

public static class Formulas

{

private static ArgumentException ThrowArgumentException(string paramName, string message) =>

new(message, paramName);

public static double DesiredProfitPercantage(double desiredProfit, double productPrice)

{

if (productPrice < 0)

throw ThrowArgumentException(nameof(productPrice), "productPrice has to be positive");

if (desiredProfit < 0)

throw ThrowArgumentException(nameof(desiredProfit), "desiredProfit has to be positive");

return desiredProfit / productPrice;

}

public static double SellVolume(double fixedCosts, double desiredProfit, double productPrice, double variableCosts)

{

if (productPrice < 0)

throw ThrowArgumentException(nameof(productPrice), "productPrice has to be positive");

if (desiredProfit < 0)

throw ThrowArgumentException(nameof(desiredProfit), "desiredProfit has to be positive");

if (fixedCosts < 0)

throw ThrowArgumentException(nameof(fixedCosts), "fixedCosts has to be positive");

if (variableCosts < 0)

throw ThrowArgumentException(nameof(variableCosts), "variableCosts has to be positive");

return (fixedCosts + desiredProfit) / (productPrice - variableCosts);

}

public static double SellSummary(double productPrice, double sellVolume)

{

if (productPrice < 0)

throw ThrowArgumentException(nameof(productPrice), "productPrice has to be positive");

if (sellVolume < 0)

throw ThrowArgumentException(nameof(sellVolume), "sellVolume has to be positive");

return productPrice \* sellVolume;

}

public static double Profit(double productPrice, double variableCosts, double sellVolume, double fixedCosts)

{

if (productPrice < 0)

throw ThrowArgumentException(nameof(productPrice), "productPrice has to be positive");

if (variableCosts < 0)

throw ThrowArgumentException(nameof(variableCosts), "variableCosts has to be positive");

if (sellVolume < 0)

throw ThrowArgumentException(nameof(sellVolume), "fixedCosts has to be positive");

if (fixedCosts < 0)

throw ThrowArgumentException(nameof(fixedCosts), "fixedCosts has to be positive");

return (productPrice - variableCosts) \* sellVolume - fixedCosts;

}

public static double CostPrice(double variableCosts, double fixedCosts, double volumeOfProduction)

{

if (variableCosts < 0)

throw ThrowArgumentException(nameof(variableCosts), "variableCosts has to be positive");

if (fixedCosts < 0)

throw ThrowArgumentException(nameof(fixedCosts), "fixedCosts has to be positive");

if (volumeOfProduction < 0)

throw ThrowArgumentException(nameof(volumeOfProduction), "variableCosts has to be positive");

return variableCosts + (fixedCosts / volumeOfProduction);

}

public static double ProductPrice(double costPrice, double desiredProfitPercentage)

{

if (costPrice < 0)

throw ThrowArgumentException(nameof(costPrice), "costPrice has to be positive");

if (desiredProfitPercentage < 0)

throw ThrowArgumentException(nameof(desiredProfitPercentage), "desiredProfitPercentage has to be positive");

return costPrice \* (1 + (desiredProfitPercentage / 100));

}

public static double QuantityPoint(double fixedCosts, double productPrice, double variableCosts)

{

if (fixedCosts < 0)

throw ThrowArgumentException(nameof(fixedCosts), "fixedCosts has to be positive");

if (variableCosts < 0)

throw ThrowArgumentException(nameof(variableCosts), "variableCosts has to be positive");

if (productPrice < 0)

throw ThrowArgumentException(nameof(productPrice), "productPrice has to be positive");

return fixedCosts / (productPrice - variableCosts);

}

public static double MoneyPoint(double productPrice, double quantityPoint)

{

if (productPrice < 0)

throw ThrowArgumentException(nameof(productPrice), "productPrice has to be positive");

if (quantityPoint < 0)

throw ThrowArgumentException(nameof(quantityPoint), "quantityPoint has to be positive");

return productPrice \* quantityPoint;

}

public static double VariableCosts(double productPrice, double variableCostsPercantage)

{

if (productPrice < 0)

throw ThrowArgumentException(nameof(productPrice), "productPrice has to be positive");

if (variableCostsPercantage < 0 || variableCostsPercantage > 100)

throw ThrowArgumentException(nameof(variableCostsPercantage), "variableCostsPercantage has to be positive and less than 100");

return variableCostsPercantage / 100 \* productPrice;

}

public static double VariableCostsPercantage(double productPrice, double variableCosts)

{

if (productPrice < 0)

throw ThrowArgumentException(nameof(productPrice), "productPrice has to be positive");

if (variableCosts < 0)

throw ThrowArgumentException(nameof(variableCosts), "variableCosts has to be positive");

return variableCosts \* 100 / productPrice;

}

}

}

# ПРИЛОЖЕНИЕ Б

Фрагменты листинга программного кода для создания интерфейса в классе TaskViewModel

using formulas;

using LiveCharts;

using LiveCharts.Helpers;

using LiveCharts.Wpf;

using ReactiveUI;

using System;

using System.Collections.Generic;

using System.Reactive;

using FastReport;

using FastReport.Export.Image;

using FastReport.Export.PdfSimple;

namespace WPF

{

public class TaskViewModel : ReactiveObject

{

public TaskViewModel()

{

this.WhenAnyValue(vm => vm.ProductPrice\_search,

vm => vm.VariableCosts,

vm => vm.SellVolume,

vm => vm.SellSummary,

vm => vm.QuantityPoint,

vm => vm.MoneyPoint,

vm => vm.Profit

).Subscribe(value =>

{

SeriesCollection = new SeriesCollection {

new ColumnSeries

{

Title = "Цена продукции в рублях",

PointGeometry = DefaultGeometries.Circle,

Values = new List<double> { value.Item1 }.AsChartValues(),

DataLabels = true

},

new ColumnSeries

{

Title = "Переменные затраты в рублях",

PointGeometry = DefaultGeometries.Circle,

Values = new List<double> { value.Item2 }.AsChartValues(),

DataLabels = true

},

new ColumnSeries

{

Title = "Объём продаж",

PointGeometry = DefaultGeometries.Circle,

Values = new List<double> { value.Item3 }.AsChartValues(),

DataLabels = true

},

new ColumnSeries

{

Title = "Сумма продаж в рублях",

PointGeometry = DefaultGeometries.Circle,

Values = new List<double> { value.Item4 }.AsChartValues(),

DataLabels = true

},

new ColumnSeries

{

Title = "Безубыточность в количестве",

PointGeometry = DefaultGeometries.Circle,

Values = new List<double> { value.Item5 }.AsChartValues(),

DataLabels = true

},

new ColumnSeries

{

Title = "Безубыточность в рублях",

PointGeometry = DefaultGeometries.Circle,

Values = new List<double> { value.Item6 }.AsChartValues(),

DataLabels = true

},

new ColumnSeries

{

Title = "Прибыль в рублях",

PointGeometry = DefaultGeometries.Circle,

Values = new List<double> { value.Item7 }.AsChartValues(),

DataLabels = true

}

};

});

RecCom = ReactiveCommand.Create(PDFLoad);

}

public List<string> ChartLabels { get; set; } = new List<string> { "Сводка нужных критерий производства" };

public Func<double, string> YFormatter { get; set; } = number => number.ToString();

private double \_productPrice = 5000;

public double ProductPrice

{

get => \_productPrice;

set

{

this.RaiseAndSetIfChanged(ref \_productPrice, value);

this.RaisePropertyChanged(nameof(DesiredProfitPercantage));

this.RaisePropertyChanged(nameof(SellVolume));

this.RaisePropertyChanged(nameof(SellSummary));

this.RaisePropertyChanged(nameof(VariableCostsPercantage));

this.RaisePropertyChanged(nameof(VariableCosts));

this.RaisePropertyChanged(nameof(CostPrice));

this.RaisePropertyChanged(nameof(QuantityPoint));

this.RaisePropertyChanged(nameof(MoneyPoint));

}

}

private double \_variableCostsPercantage\_have = 30;

public double VariableCostsPercantage\_have

{

get => \_variableCostsPercantage\_have;

set

{

this.RaiseAndSetIfChanged(ref \_variableCostsPercantage\_have, value);

this.RaisePropertyChanged(nameof(VariableCosts));

this.RaisePropertyChanged(nameof(CostPrice));

this.RaisePropertyChanged(nameof(ProductPrice\_search));

this.RaisePropertyChanged(nameof(QuantityPoint));

this.RaisePropertyChanged(nameof(MoneyPoint));

}

}

private double \_fixedCosts = 80000;

public double FixedCosts

{

get => \_fixedCosts;

set

{

this.RaiseAndSetIfChanged(ref \_fixedCosts, value);

this.RaisePropertyChanged(nameof(SellVolume));

this.RaisePropertyChanged(nameof(CostPrice));

this.RaisePropertyChanged(nameof(ProductPrice\_search));

this.RaisePropertyChanged(nameof(QuantityPoint));

this.RaisePropertyChanged(nameof(MoneyPoint));

}

}

private double \_desiredProfit\_have = 800000;

public double DesiredProfit\_have

{

get => \_desiredProfit\_have;

set

{

this.RaiseAndSetIfChanged(ref \_desiredProfit\_have, value);

this.RaisePropertyChanged(nameof(SellVolume));

this.RaisePropertyChanged(nameof(SellSummary));

this.RaisePropertyChanged(nameof(DesiredProfitPercantage));

this.RaisePropertyChanged(nameof(ProductPrice\_search));

}

}

private double \_volumeOfProduction = 1500;

public double VolumeOfProduction

{

get => \_volumeOfProduction;

set

{

this.RaiseAndSetIfChanged(ref \_volumeOfProduction, value);

this.RaisePropertyChanged(nameof(CostPrice));

this.RaisePropertyChanged(nameof(ProductPrice\_search));

}

}

public double DesiredProfitPercantage

{

get => Math.Round(Formulas.DesiredProfitPercantage(DesiredProfit\_have, ProductPrice), 3);

set

{

}

}

public double SellVolume

{

get => Math.Round(Formulas.SellVolume(FixedCosts, DesiredProfit\_have, ProductPrice, VariableCosts), 0);

set

{

}

}

public double SellSummary

{

get => Math.Round(Formulas.SellSummary(ProductPrice, SellVolume), 3);

}

public double VariableCosts

{

get => Math.Round(Formulas.VariableCosts(ProductPrice, VariableCostsPercantage\_have), 3);

}

public double VariableCostsPercantage

{

get => Math.Round(Formulas.VariableCostsPercantage(ProductPrice, VariableCosts), 3);

set { }

}

public double Profit

{

get => Math.Round(Formulas.Profit(ProductPrice, VariableCosts, SellVolume, FixedCosts), 3);

}

public double CostPrice

{

get => Math.Round(Formulas.CostPrice(VariableCosts, FixedCosts, VolumeOfProduction), 3);

}

public double ProductPrice\_search

{

get => Math.Round(Formulas.ProductPrice(CostPrice, DesiredProfitPercantage), 3);

}

public double QuantityPoint

{

get => Math.Round(Formulas.QuantityPoint(FixedCosts, ProductPrice, VariableCosts), 0);

}

public double MoneyPoint

{

get => Math.Round(Formulas.MoneyPoint(ProductPrice, QuantityPoint), 3);

}

private SeriesCollection \_seriesCollection;

public SeriesCollection SeriesCollection

{

get => \_seriesCollection;

set => this.RaiseAndSetIfChanged(ref \_seriesCollection, value);

}

public ReactiveCommand<Unit, Unit> RecCom { get; }

private void PDFLoad()

{

List<Data> Dataa = new();

Dataa.Add(new Data { Name = "Желаемый доход %", Value = DesiredProfitPercantage });

Dataa.Add(new Data { Name = "Объём продаж", Value = SellVolume });

Dataa.Add(new Data { Name = "Сумма продаж", Value = SellSummary });

Dataa.Add(new Data { Name = "Прибыль", Value = Profit });

Dataa.Add(new Data { Name = "Себестоимость производства", Value = CostPrice });

Dataa.Add(new Data { Name = "Цена продукции", Value = ProductPrice });

Dataa.Add(new Data { Name = "Безубыточность в количестве", Value = QuantityPoint});

Dataa.Add(new Data { Name = "Безубыточность в деньгах", Value = MoneyPoint});

Dataa.Add(new Data { Name = "Переменные затраты", Value = VariableCosts });

Dataa.Add(new Data { Name = "Переменные затраты %", Value = VariableCostsPercantage });

Report report = new();

report.Load("Table.frx");

report.RegisterData(Dataa, "Data");

report.Prepare();

report.SavePrepared("Prepared\_Table.fpx");

ImageExport image = new();

image.ImageFormat = ImageExportFormat.Jpeg;

report.Export(image, "report.jpg");

PDFSimpleExport pdfExport = new();

pdfExport.Export(report, "report.pdf");

report.Dispose();

}

}

}

# ПРИЛОЖЕНИЕ В

Фрагменты листинга программного кода проекта «Tests»

using formulas;

using NUnit.Framework;

using System;

namespace Tests

{

public class Tests

{

[SetUp]

public void Setup()

{

}

[Test]

public void Test1()

{

var productPrice = 5000;

var desiredProfit = 800000;

var delta = 0.005;

Assert.AreEqual(160.0, Formulas.DesiredProfitPercantage(desiredProfit,productPrice), delta);

}

[Test]

public void Test2()

{

var productPrice = -5000;

var desiredProfit = 800000;

Assert.Throws<ArgumentException>(() => Formulas.DesiredProfitPercantage(desiredProfit, productPrice), "productPrice has to be positive");

}

//----------------------SellVolume-----------------------------

[Test]

public void Test3()

{

var fixedCosts = -50000;

var desiredProfit = 400000;

var productPrice = 5000;

var variableCosts = 1000;

Assert.Throws<ArgumentException>(() => Formulas.SellVolume(fixedCosts, desiredProfit, productPrice, variableCosts), "fixedCosts has to be positive");

}

[Test]

public void Test4()

{

var fixedCosts = 80000;

var desiredProfit = 800000;

var productPrice = 5000;

var variableCosts = 1500;

var delta = 0.005;

Assert.AreEqual(251.429, Formulas.SellVolume(fixedCosts, desiredProfit, productPrice, variableCosts), delta);

}

[Test]

public void Test5()

{

var fixedCosts = 50000;

var desiredProfit = -400000;

var productPrice = 5000;

var variableCosts = 1000;

Assert.Throws<ArgumentException>(() => Formulas.SellVolume(fixedCosts, desiredProfit, productPrice, variableCosts), "desiredProfit has to be positive");

}

[Test]

public void Test6()

{

var fixedCosts = 50000;

var desiredProfit = 400000;

var productPrice = -5000;

var variableCosts = 1000;

Assert.Throws<ArgumentException>(() => Formulas.SellVolume(fixedCosts, desiredProfit, productPrice, variableCosts), "productPrice has to be positive");

}

[Test]

public void Test7()

{

var fixedCosts = 50000;

var desiredProfit = 400000;

var productPrice = 5000;

var variableCosts = -1000;

Assert.Throws<ArgumentException>(() => Formulas.SellVolume(fixedCosts, desiredProfit, productPrice, variableCosts), "variableCosts has to be positive");

}

//----------------------SellVolume-----------------------------

//----------------------SellSummary----------------------------

[Test]

public void Test8()

{

var productPrice = -5000;

var sellVolume = 176;

Assert.Throws<ArgumentException>(() => Formulas.SellSummary(productPrice, sellVolume), "productPrice has to be positive");

}

[Test]

public void Test9()

{

var productPrice = 5000;

var sellVolume = -176;

Assert.Throws<ArgumentException>(() => Formulas.SellSummary(productPrice, sellVolume), "sellVolume has to be positive");

}

[Test]

public void Test10()

{

var productPrice = 5000;

var sellVolume = 176;

var delta = 0.005;

Assert.AreEqual(880000.0, Formulas.SellSummary(productPrice, sellVolume), delta);

}

//----------------------SellSummary----------------------------

//----------------------Profit---------------------------------

[Test]

public void Test11()

{

var productPrice = -5000;

var variableCosts = 1500;

var sellVolume = 176;

var fixedCosts = 80000;

Assert.Throws<ArgumentException>(() => Formulas.Profit(productPrice, variableCosts ,sellVolume, fixedCosts), "productPrice has to be positive");

}

[Test]

public void Test12()

{

var productPrice = 5000;

var variableCosts = 1500;

var sellVolume = -176;

var fixedCosts = 80000;

Assert.Throws<ArgumentException>(() => Formulas.Profit(productPrice, variableCosts, sellVolume, fixedCosts), "sellVolume has to be positive");

}

[Test]

public void Test13()

{

var productPrice = 5000;

var variableCosts = -1500;

var sellVolume = 176;

var fixedCosts = 80000;

Assert.Throws<ArgumentException>(() => Formulas.Profit(productPrice, variableCosts, sellVolume, fixedCosts), "variableCosts has to be positive");

}

[Test]

public void Test14()

{

var productPrice = 5000;

var variableCosts = 1500;

var sellVolume = 176;

var fixedCosts = -80000;

Assert.Throws<ArgumentException>(() => Formulas.Profit(productPrice, variableCosts, sellVolume, fixedCosts), "fixedCosts has to be positive");

}

[Test]

public void Test15()

{

var productPrice = 5000;

var variableCosts = 1500;

var sellVolume = 176;

var fixedCosts = 80000;

var delta = 0.005;

Assert.AreEqual(536000.0, Formulas.Profit(productPrice, variableCosts, sellVolume, fixedCosts), delta);

}

//----------------------Profit---------------------------------

//----------------------CostPrice------------------------------

[Test]

public void Test16()

{

var variableCosts = -1500;

var fixedCosts = 80000;

var volumeOfProduction = 1500;

Assert.Throws<ArgumentException>(() => Formulas.CostPrice(variableCosts, fixedCosts, volumeOfProduction), "variableCosts has to be positive");

}

[Test]

public void Test17()

{

var variableCosts = 1500;

var fixedCosts = -80000;

var volumeOfProduction = 1500;

Assert.Throws<ArgumentException>(() => Formulas.CostPrice(variableCosts, fixedCosts, volumeOfProduction), "fixedCosts has to be positive");

}

[Test]

public void Test18()

{

var variableCosts = 1500;

var fixedCosts = 80000;

var volumeOfProduction = -1500;

Assert.Throws<ArgumentException>(() => Formulas.CostPrice(variableCosts, fixedCosts, volumeOfProduction), "volumeOfProduction has to be positive");

}

[Test]

public void Test19()

{

var variableCosts = 1500;

var fixedCosts = 80000;

var volumeOfProduction = 1500;

var delta = 0.005;

Assert.AreEqual(1553.333, Formulas.CostPrice(variableCosts, fixedCosts, volumeOfProduction), delta);

}

//----------------------CostPrice------------------------------

//----------------------ProductPrice---------------------------

[Test]

public void Test21()

{

var costPrice = -1553.333;

var desiredProfitPercentage = 30;

Assert.Throws<ArgumentException>(() => Formulas.ProductPrice(costPrice, desiredProfitPercentage), "costPrice has to be positive");

}

[Test]

public void Test22()

{

var costPrice = 1553.333;

var desiredProfitPercentage = -30;

Assert.Throws<ArgumentException>(() => Formulas.ProductPrice(costPrice, desiredProfitPercentage), "desiredProfitPercentage has to be positive");

}

[Test]

public void Test23()

{

var costPrice = 1553.333;

var desiredProfitPercentage = 160;

var delta = 0.005;

Assert.AreEqual(4038.666, Formulas.ProductPrice(costPrice, desiredProfitPercentage), delta);

}

//----------------------ProductPrice---------------------------

//----------------------QuantityPoint--------------------------

[Test]

public void Test24()

{

var fixedCosts = -80000;

var productPrice = 5000;

var variableCosts = 1500;

Assert.Throws<ArgumentException>(() => Formulas.QuantityPoint(fixedCosts, productPrice, variableCosts), "fixedCosts has to be positive");

}

[Test]

public void Test25()

{

var fixedCosts = 80000;

var productPrice = -5000;

var variableCosts = 1500;

Assert.Throws<ArgumentException>(() => Formulas.QuantityPoint(fixedCosts, productPrice, variableCosts), "productPrice has to be positive");

}

[Test]

public void Test26()

{

var fixedCosts = 80000;

var productPrice = 5000;

var variableCosts = -1500;

Assert.Throws<ArgumentException>(() => Formulas.QuantityPoint(fixedCosts, productPrice, variableCosts), "variableCosts has to be positive");

}

[Test]

public void Test27()

{

var fixedCosts = 80000;

var productPrice = 5000;

var variableCosts = 1500;

var delta = 0.005;

Assert.AreEqual(23, Formulas.QuantityPoint(fixedCosts, productPrice, variableCosts), delta);

}

//----------------------QuantityPoint--------------------------

//----------------------MoneyPoint-----------------------------

[Test]

public void Test28()

{

var productPrice = -5000;

var quantityPoint = 23;

Assert.Throws<ArgumentException>(() => Formulas.MoneyPoint(productPrice, quantityPoint), "productPrice has to be positive");

}

[Test]

public void Test29()

{

var productPrice = 5000;

var quantityPoint = -23;

Assert.Throws<ArgumentException>(() => Formulas.MoneyPoint(productPrice, quantityPoint), "quantityPoint has to be positive");

}

[Test]

public void Test30()

{

var productPrice = 5000;

var quantityPoint = 23;

var delta = 0.005;

Assert.AreEqual(115000, Formulas.MoneyPoint(productPrice, quantityPoint), delta);

}

//----------------------MoneyPoint-----------------------------

//----------------------VariableCosts--------------------------

[Test]

public void Test31()

{

var productPrice = -5000;

var variableCostsPercantage = 30;

Assert.Throws<ArgumentException>(() => Formulas.VariableCosts(productPrice, variableCostsPercantage), "productPrice has to be positive");

}

[Test]

public void Test32()

{

var productPrice = 5000;

var variableCostsPercantage = -30;

Assert.Throws<ArgumentException>(() => Formulas.VariableCosts(productPrice, variableCostsPercantage), "variableCostsPercantage has to be positive");

}

[Test]

public void Test37()

{

var productPrice = 5000;

var variableCostsPercantage = 130;

Assert.Throws<ArgumentException>(() => Formulas.VariableCosts(productPrice, variableCostsPercantage), "variableCostsPercantage has to be less than 100");

}

[Test]

public void Test33()

{

var productPrice = 5000;

var variableCostsPercantage = 30;

var delta = 0.005;

Assert.AreEqual(1500, Formulas.VariableCosts(productPrice, variableCostsPercantage), delta);

}

//----------------------VariableCosts--------------------------

//----------------------VariableCostsPercantage----------------

[Test]

public void Test34()

{

var productPrice = -5000;

var variableCosts = 1500;

Assert.Throws<ArgumentException>(() => Formulas.VariableCostsPercantage(productPrice, variableCosts), "productPrice has to be positive");

}

[Test]

public void Test35()

{

var productPrice = 5000;

var variableCosts = -1500;

Assert.Throws<ArgumentException>(() => Formulas.VariableCostsPercantage(productPrice, variableCosts), "variableCosts has to be positive");

}

[Test]

public void Test36()

{

var productPrice = 5000;

var variableCosts = 1500;

var delta = 0.005;

Assert.AreEqual(30, Formulas.VariableCostsPercantage(productPrice, variableCosts), delta);

}

//----------------------VariableCostsPercantage----------------

}

}