[Техническое задание 2](#_Toc202302804)

[Блок-схемы 4](#_Toc202302805)

[Отладка программного обеспечения 7](#_Toc202302806)

[Тест-кейсы 8](#_Toc202302807)

[Unit-тесты 12](#_Toc202302808)

[Инспектирование кода 14](#_Toc202302809)

# Техническое задание

Программный продукт под названием «Расчет бюджета на выбранный период» разрабатывается с целью автоматизации процесса ведения личного финансового учёта. Он будет использоваться частным пользователем для фиксации и анализа своих доходов и расходов за определённый временной промежуток. Основная задача программы — предоставить удобный и наглядный способ определить финансовый результат за выбранный период, а также облегчить контроль над расходами.

Работа программы основывается на принципе регистрации операций, относящихся к категории «доход» или «расход». Каждая операция содержит информацию о дате, сумме и категории. Пользователь сможет вводить новые записи, просматривать существующие, а также при необходимости изменять их. Интерфейс программы будет включать форму с таблицами, где отображаются все введённые данные. Также пользователь сможет выбрать интересующий его период времени при помощи календарных элементов и выполнить расчёт бюджета. Результатом расчёта станет отображение общей суммы доходов, расходов и итогового баланса — то есть разницы между доходами и расходами. Вывод этой информации будет осуществляться в виде отдельного окна или текстового поля, находящегося в нижней части формы.

Программа должна быть реализована на языке программирования C# с использованием технологии Windows Forms. Предполагается использование .NET Framework или .NET 6+ в зависимости от выбранной платформы. Разработка будет вестись в среде Visual Studio, что обеспечит удобную отладку и сопровождение кода. Программа должна обладать простым и интуитивно понятным интерфейсом, не требующим специальной подготовки со стороны пользователя.

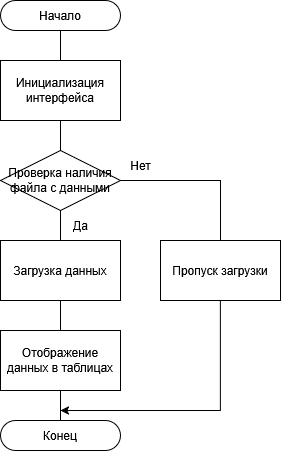
Особое внимание в процессе разработки будет уделяться корректности обработки пользовательского ввода. Программа должна обрабатывать некорректные или пустые значения, не допуская их сохранения в списке операций. Кроме того, необходимо предусмотреть возможность в будущем расширить функциональность — например, добавить построение графиков, экспорт данных или работу с базой данных.

Код программного продукта должен соответствовать базовым принципам чистоты и читаемости: имена переменных и методов должны быть понятными и отражать суть, структура кода — логичной и модульной. При реализации также потребуется обеспечить базовую обработку исключений, чтобы избежать критических сбоев при работе приложения.

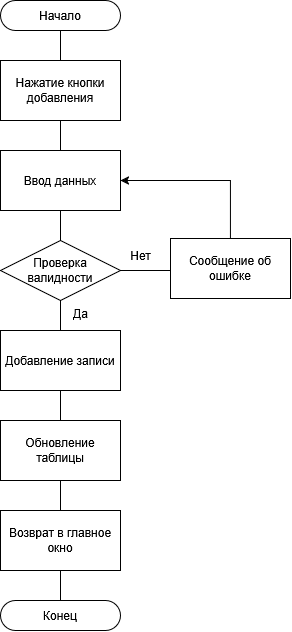
Таким образом, разработка программного продукта предполагает создание устойчивой и удобной в использовании системы, способной эффективно решать задачу анализа бюджета за любой выбранный пользователем период.

# Блок-схемы

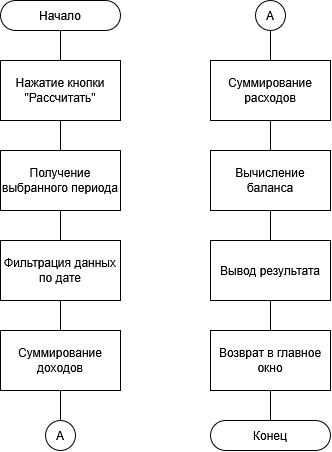
Блок-схема процесса запуска программы и загрузки данных



Блок-схема добавления новой записи (дохода или расхода)



Блок-схема расчёта бюджета за выбранный период



# Отладка программного обеспечения

На этапе отладки программного обеспечения использовались встроенные средства среды разработки Visual Studio, такие как точки останова и пошаговое выполнение. Основной задачей отладки было проверка корректности логики добавления доходов и расходов, а также корректного расчета бюджета за выбранный период.

Отладка проводилась вручную, с акцентом на анализ значений переменных во время исполнения. Для этого были установлены точки останова в методах обработки кнопок добавления данных и кнопки расчета. Во время выполнения приложение запускалось в режиме отладки, после чего значения переменных, таких как дата, сумма и результат вычислений, отслеживались через окно локальных переменных. Это позволило убедиться, что фильтрация по дате и суммирование значений работают корректно.

Также в процессе отладки было выявлено, что попытка добавления строк в таблицы DataGridView без предварительного определения столбцов вызывает исключение. Проблема была решена путём добавления нужных столбцов при загрузке формы.

В целом, отладка показала стабильную работу приложения при стандартных сценариях использования. Ошибки, возникавшие на начальном этапе, были оперативно устранены. Использование пошагового выполнения и точек останова позволило контролировать ход выполнения программы без применения дополнительных отладочных инструментов или логирования.

# Тест-кейсы

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | | | TC01 |
| Заголовок | | | Расчёт бюджета за период при наличии дохода и расхода |
| Предусловие | | | Программа запущена, добавлены как минимум одна запись дохода и расхода |
| Тестовые данные | | | Доход: "Зарплата", 15000 ₽, 01.06.2025; Расход: "Еда", 5000 ₽, 05.06.2025 |
| Ожидаемый результат | | | Вывод в Label: "Доходы: 15000 ₽ Расходы: 5000 ₽ Баланс: 10000 ₽" |
| Шаг | Действие | Предполагаемый результат | Pass / Fall |
| 1. | Выбрать период: с 01.06.2025 по 30.06.2025 | Период применён | Pass |
| 2. | Нажать "Рассчитать бюджет" | Отображается корректный баланс: 10000 ₽ | Pass |
| Постусловие | | | - |
| Результат | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | | | TC02 |
| Заголовок | | | Расчёт бюджета при пустом списке операций |
| Предусловие | | | Нет записей доходов и расходов в таблицах |
| Тестовые данные | | | Нет данных |
| Ожидаемый результат | | | Баланс равен 0, сообщение: "Доходы: 0 ₽ Расходы: 0 ₽ Баланс: 0 ₽" |
| Шаг | Действие | Предполагаемый результат | Pass / Fall |
| 1. | Убедиться, что таблицы пусты | Ни одной строки в DataGridView | Pass |
| 2. | Нажать "Рассчитать бюджет" | Итоговое значение — нули | Pass |
| Постусловие | | | - |
| Результат | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | | | TC03 |
| Заголовок | | | Расчёт бюджета при отсутствии операций в пределах выбранного периода |
| Предусловие | | | В таблицах имеются записи, но все они вне выбранного диапазона дат |
| Тестовые данные | | | Доход: 10.05.2025, 1000 ₽; Расход: 01.07.2025, 500 ₽. Период: с 01.06.2025 по 30.06.2025 |
| Ожидаемый результат | | | Отображается: Доходы: 0 ₽, Расходы: 0 ₽, Баланс: 0 ₽ |
| Шаг | Действие | Предполагаемый результат | Pass / Fall |
| 1. | Выбрать период: 01.06.2025 – 30.06.2025 | Применён диапазон | Pass |
| 2. | Нажать кнопку "Рассчитать бюджет" | Выведен нулевой результат | Pass |
| Постусловие | | | - |
| Результат | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | | | TC04 |
| Заголовок | | | Проверка учёта операций на границах выбранного периода |
| Предусловие | | | В таблицах есть запись с датой, совпадающей с границей диапазона |
| Тестовые данные | | | Доход: 30.06.2025, 2000 ₽. Период: с 01.06.2025 по 30.06.2025 |
| Ожидаемый результат | | | Доход учтён в балансе: Доходы: 2000 ₽, Расходы: 0 ₽, Баланс: 2000 ₽ |
| Шаг | Действие | Предполагаемый результат | Pass / Fall |
| 1. | Выбрать период с 01.06.2025 по 30.06.2025 | Дата установлена | Pass |
| 2. | Нажать "Рассчитать бюджет" | Запись учтена | Pass |
| Постусловие | | | - |
| Результат | | | Pass |

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | | | TC05 |
| Заголовок | | | Расчёт бюджета с несколькими доходами и расходами |
| Предусловие | | | В таблице несколько записей по обеим категориям |
| Тестовые данные | | | Доходы: 01.06.2025 – 1000 ₽, 10.06.2025 – 500 ₽; Расходы: 15.06.2025 – 300 ₽. Период: 01.06.2025 – 30.06.2025 |
| Ожидаемый результат | | | Доходы: 1500 ₽, Расходы: 300 ₽, Баланс: 1200 ₽ |
| Шаг | Действие | Предполагаемый результат | Pass / Fall |
| 1. | Установить даты: 01.06.2025 – 30.06.2025 | Период выбран | Pass |
| 2. | Нажать "Рассчитать бюджет" | Результат рассчитан верно | Pass |
| Постусловие | | | - |
| Результат | | | Pass |

# Unit-тесты

Для проверки корректности работы вычислительной логики программного продукта был разработан отдельный модуль модульного тестирования с использованием библиотеки xUnit. Цель тестирования заключалась в проверке правильности работы метода расчёта итоговой суммы доходов и расходов за заданный временной промежуток.

В рамках тестирования ключевая часть логики — фильтрация по диапазону дат и суммирование значений — была вынесена в отдельный метод GetTotal(...) статического класса BudgetLogic. Это позволило протестировать вычисления в изоляции от пользовательского интерфейса и внешней среды выполнения.

В ходе тестирования были реализованы следующие модульные проверки:

* GetTotal\_ReturnsCorrectSum — проверка корректного суммирования значений, попадающих в указанный период;
* GetTotal\_EmptyList\_ReturnsZero — проверка работы метода при передаче пустого списка;
* GetTotal\_AllEntriesOutsidePeriod\_ReturnsZero — проверка, что при отсутствии подходящих по дате записей метод возвращает 0;
* GetTotal\_EntryExactlyOnBoundary\_IsIncluded — проверка включения записей, находящихся на границе диапазона (включительно);
* GetTotal\_PartialInPeriod\_ReturnsPartialSum — проверка корректности фильтрации, если часть записей находится в пределах диапазона, а часть — вне его.

Все тесты базировались на заранее подготовленных тестовых данных, имитирующих реальные случаи использования. Запуск тестов осуществлялся с помощью встроенной системы тестирования Visual Studio. В результате выполнения проверок ошибки в логике обнаружены не были.

Таким образом, модульная проверка показала, что основная вычислительная часть приложения функционирует стабильно и предсказуемо при различных входных данных, что подтверждает её готовность к использованию в составе оконного интерфейса.

# Инспектирование кода

В ходе разработки программного продукта было проведено инспектирование исходного кода на соответствие стандартам оформления и стилевым соглашениям языка C#. Целью инспектирования было выявление потенциальных несоответствий, влияющих на читаемость, поддержку и расширяемость кода.

Проверка включала анализ имён переменных и методов, структуру классов, форматирование, использование документации и других аспектов. Также была проверена логика именования с учётом практики camelCase и PascalCase, а также рекомендованных названий для методов, свойств и классов.

По результатам анализа была составлена таблица нарушений и рекомендаций.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Фрагмент кода | Несоответствие | Рекомендации |
| 1 | public String ConString = ... | Имя поля написано с заглавной буквы, не по стандарту для полей | Использовать camelCase: conString или connectionString |
| 2 | textBoxCategory.Text.Trim(); | Несколько повторяющихся вызовов .Text.Trim() без сохранения результата | Сохранить в переменную: var category = textBoxCategory.Text.Trim(); |
| 3 | DateTime.Parse(row.Cells[0].Value.ToString()) | Отсутствует проверка на null перед ToString() | Добавить проверку или использовать as string с TryParse |
| 4 | decimal.TryParse(..., out decimal amount) | Название переменной amount может быть перезаписано в другом методе | Использовать более локальные и описательные имена, например parsedAmount |
| 5 | Отсутствие XML-комментариев к методам | Методы и классы не документированы | Добавить ///-комментарии к публичным методам: что делают, какие параметры |
| 6 | Расчёт баланса напрямую в форме (Form1) | Логика не отделена от интерфейса | Вынести расчёты в отдельный класс (как сделано с BudgetLogic) |