МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ

**«Дальневосточный федеральный университет»**

**ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**

**Департамент программной инженерии и искусственного интеллекта**

**РАЗРАБОТКА КОНСТРУКТОРА ЗАДАЧ ПО ОПРЕДЕЛЕНИЮ РЕАКЦИЙ ОПОР**КУРСОВАЯ РАБОТА  
по дисциплине «Технологии коллективной промышленной разработки информационных систем» по образовательной программе подготовки бакалавров по направлению 09.03.04 «Программная инженерия»

Выполнили:

студенты гр. Б9121-09.03.04

Булах А.А.

Ермак В.М.

Лазарев А.Е.

Лебединский И.В.

Пасько У.Н.  
Руководитель:

Ассистент ДПИиИИ

(учёная степень, должность)

Логачев Е.М.

(подпись) (ФИО)

Регистрационный № Защищён с оценкой

  2025 г.

(подпись) (ФИО)

2025 г.

**Оглавление**

[Введение 3](#_Toc196843685)

[1 Декомпозиция задач. План проекта. 4](#_Toc196843686)

[2 Разработка регламента инспекции рабочих продуктов 7](#_Toc196843687)

[3 Разработка модели состояний задач 11](#_Toc196843688)

[4 Разработка презентации проекта 14](#_Toc196843689)

[5 Разработка требований к проекту 20](#_Toc196843690)

[6 Разработка архитектуры проекта 25](#_Toc196843691)

[7 Система метрик для оценки эффективности процесса разработки 28](#_Toc196843692)

[8 Разработка перечня задач проекта 30](#_Toc196843693)

[9 Разработка рекомендаций по кодированию 33](#_Toc196843694)

[10 Разработка плана тестирования проекта 36](#_Toc196843695)

[11 Тестирование проекта 41](#_Toc196843696)

[Заключение 66](#_Toc196843697)

[Список литературы 67](#_Toc196843698)

# Введение

Промышленная разработка информационных систем включает в себя множество этапов, начиная от разработки плана проекта, заканчивая тестированием проекта для чего, очевидно, необходимо множество специалистов различных профилей, а также унифицированные методы коммуникации между ними, с помощью которых можно разделить обязанности членов команды по их специализации.

Исходя из описанного выше необходимо использовать определенные технологии коллективной разработки для повышения эффективности работы в группе и соответствия конечного продукта заявленным требованиям.

В данной курсовой работе рассматривается задача коллективной разработки проекта «Конструктор задач по определению реакции опор» и составление технической документации к нему.

Таким образом, целью курсовой работы является разработка разработки проекта «Конструктор задач по определению реакции опор» с использованием подходов коллективной промышленной разработки.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать план проекта;
2. Разработать регламент проведения инспекции;
3. Разработать модель состояний задач;
4. Разработать презентацию проекта;
5. Разработать требования к проекту;
6. Разработать архитектуру проекта;
7. Разработать измерения проекта;
8. Разработать перечень задач проекта;
9. Разработать рекомендации по кодированию;
10. Разработать план тестирования проекта;
11. Протестировать проект.

# Декомпозиция задач. План проекта.

В данном разделе представлен анализ основных этапов разработки проекта, а также распределение задач между участниками команды. Разработка детализированного плана необходима для эффективной организации работы и своевременного достижения намеченных целей.

1. Разработка плана проекта
   1. Составление календарного плана:
      1. Определение основных этапов и сроков.
      2. Учет зависимостей между задачами.
   2. Организация работы команды:
      1. Распределение ролей (тимлид, разработчик, тестировщик и др.).
      2. Разработка документа с ролями и обязанностями.
2. Разработка регламента инспекций
   1. Определение критериев инспекций:
      1. Критерии проверки кода, требований, документации.
   2. Создание ролей и этапов инспекций:
      1. Формализация ролей (модератор, автор, рецензент).
      2. Определение этапов (подготовка, рецензирование, обсуждение).
3. Разработка модели состояний задач
   1. Определение возможных состояний задач:
      1. Backlog, To Do, In Progress, Review, Done, Blocked.
   2. Разработка регламента переходов:
      1. Правила перехода задач между состояниями.
4. Разработка требований к проекту
   1. Сбор требований:
      1. Определение функциональных и нефункциональных требований.
   2. Разделение требований по подсистемам:
      1. Пользовательский интерфейс, модуль расчётов, визуализация.
5. Разработка архитектуры проекта
   1. Создание архитектурных диаграмм:
      1. Диаграммы потоков данных, классов, вызовов.
   2. Определение взаимодействий подсистем:
      1. Взаимодействие UI, расчётов и визуализации.
6. Разработка программы измерений проекта
   1. Определение метрик эффективности разработки:
      1. Время цикла, эффективность потока.
   2. Разработка метрик качества:
      1. Надёжность и точность расчётов.
   3. Публикация программы измерений:
      1. Размещение документа с метриками в GitHub.
7. Реализация проекта
   1. Разработка пользовательского интерфейса:
      1. Реализация функций редактирования конструкций и визуализации.
   2. Разработка модуля расчётов:
      1. Расчёт реакций опор и проверка данных.
   3. Реализация подсистемы визуализации:
      1. Визуализация расчётов, экспорт изображений.
   4. Интеграция подсистем:
      1. Обеспечение взаимодействия UI, расчётов и визуализации.
8. Выпуск альфа-версии
   1. Сборка альфа-версии:
      1. Интеграция основных функций в рабочую версию.
   2. Размещение альфа-версии в репозитории:
      1. Публикация версии для начального тестирования.
9. Составление расписания тестирования
   1. Разработка расписания тестирования:
      1. Планирование модульного, интеграционного и пользовательского тестирования.
10. Тестирование
    1. Модульное тестирование:
       1. Проверка подсистем на соответствие требованиям.
    2. Интеграционное тестирование:
       1. Тестирование взаимодействия подсистем.
    3. Пользовательское тестирование:
       1. Проверка удобства работы системы студентами и преподавателями.
    4. Исправление ошибок и повторное тестирование:
       1. Устранение замечаний и повторная проверка.

Проектная команда:

1. Тимлид – Ермак В.М.
2. Программист 1 – Лебединский И.В.
3. Программист 2 – Лазарев А.Е.
4. Проектировщик программного продукта – Булах А.А.
5. Проектный менеджер – Пасько У.Н.

Сроки выполнения:

1. Составление плана разработки – 02.10.24 – 23.10.24
2. Разработка регламента инспекций – 23.10.24 – 06.11.24
3. Разработка модели состояний задач– 30.10.24 – 13.11.24
4. Разработка требований к проекту – 06.11.24 – 20.11.24
5. Разработка архитектуры проекта – 20.11.24 – 11.12.24
6. Разработка программы измерений проекта – 27.11.24 – 25.12.24
7. Реализация проекта– 11.12.24 – 15.01.25
8. Выпуск альфа-версии – 25.12.24 – 15.01.25
9. Составление расписания тестирования – 15.01.25 – 22.01.25
10. Тестирование – 22.01.25 – 28.02.25

Изображение выглядит как текст, линия, Шрифт, График

Автоматически созданное описание

Рисунок 1 – Сроки выполнения задач проекта

# Разработка регламента инспекции рабочих продуктов

Верификация рабочих продуктов является неотъемлемой частью процесса по обеспечению их качества. Современной технологией программирования выработаны специальные стандарты, подходы и механизмы проведения верификаций рабочих продуктов в формате так называемых инспекций.

Инспекция – это мероприятие по обеспечению качества рабочих продуктов проектов по разработке ПО и иной деятельности, которая проводится разработчиками, возможно – с участием представителей заказчика. Концептуально инспекция имеет следующие цели:

1. Обнаружить ошибки в функциях, логике, содержании или реализации рабочих продуктов на ранних этапах их разработки и предотвратить их наследование;
2. Рационально донести замысел или реализацию продукта до всех заинтересованных лиц (через их участие);
3. Оптимизировать, оценить или улучшить рабочий продукт.

Регламент проведения инспекции рабочих продуктов для студенческих проектов, выполняемых по дисциплине «Технологии коллективной промышленной разработки информационных систем».

## 2.1 Критерии отнесения к формальной инспекции. Проводится для:

1. Документов требований с объемом более 20 страниц.
2. Документов дизайна, если они влияют на критические функции системы.
3. Кода с высоким уровнем риска (например, связанного с вычислительными модулями, оптимизацией или безопасностью).
4. Тестов, предназначенных для проверки критических путей.

## 2.2 Неформальная инспекция допускается для:

1. Промежуточных версий требований, дизайна и тестов.
2. Кода, который не содержит критических функций или является экспериментальным.

## 2.3 Роли участников инспекции и их обязанности

1. Автор (создатель рабочего продукта) – подготавливает материалы к инспекции и объясняет сложные моменты участникам.
2. Модератор – организует инспекцию, контролирует соблюдение процесса и сроков. Составляет итоговый отчет.
3. Рецензент: анализирует рабочий продукт, выявляет недостатки.
4. Записывающий (секретарь): фиксирует замечания и комментарии во время обсуждения.
5. Эксперт: оценивает узкоспециализированные аспекты продукта.

Рекомендации по числу участников:

1. Для продуктов объемом до 10 страниц – 3-4 участника.
2. Для продуктов от 10 до 50 страниц – 4-6 участников.
3. Для более крупных продуктов – от 6 участников.

## 2.4 Этапы инспекции

1. Подготовка: сбор материалов, назначение участников, определение сроков.
2. Рецензирование: участники анализируют продукт индивидуально.
3. Обсуждение: проводится собрание для обсуждения замечаний.
4. Заключение: фиксация замечаний, определение статусов.
5. Верификация: тестирование исправлений выявленных ошибок.

## 2.5 Порядок организации

* 1. Автор выгружает материалы в корпоративное хранилище.
  2. Модератор организует встречи, отправляет ссылки на материалы, и приглашения участникам.
  3. Для критических продуктов используется инструмент отслеживания задач.

## 2.6 Порядок подготовки к инспекции

* 1. Материалы предоставляются за 3 рабочих дня до инспекции.
  2. Время обсуждения согласовывается с учётом доступности участников.
  3. Участники получают четкие инструкции по анализу продукта.

## 2.7 Порядок проведения инспекции

1. Автор: представляет продукт, отвечает на вопросы.
2. Рецензенты и эксперты: озвучивают замечания.
3. Модератор: фиксирует итоги и управляет дискуссией.
4. Время проведения: до 1 часа на каждые 10 страниц продукта.

## 2.8 Перечень статусов и степени важности замечаний

Статусы:

1. «К исправлению» (несоответствие или неполнота выполнения задачи).
2. «Информационное» (уточнение по реализации).
3. «Отклонено» (критическое несоответствие).

Степени важности:

1. Высокая (критическое влияние на проект) – компоненты, участвующие в основном функционале, вызывающие критические ошибки.
2. Средняя (может повлиять на сроки или качество) – дополнительный функционал, удобство интерфейса и широта его функционирования. Советы и подсказки для пользователя.
3. Низкая (косметические замечания) – оптимизация отдельных частей программы, интуитивная понятность программы, интерактивное обучение использования программы.

## 2.9 Порядок верификации учёта замечаний

1. Исправления выполняются автором в течение 5 рабочих дней.
2. Модератор проверяет устранение замечаний и сообщает участникам.

## 2.10 Метрики эффективности инспекций

1. Процент устраненных замечаний (≥ 90%).
2. Среднее время исправления замечаний.
3. Соотношение замечаний по важности (высокие ≤ 10% от общего числа).
4. Количество выявленных дефектов на единицу объёма продукта.

## 2.11 Inspection Fault Density (IFD)

В качестве метрики, характеризующей эффективность инспекции, была выбрана Inspection Fault Density (IFD). IFD характеризует эффективность инспекции, а также качество инспектируемого продукта:

Чем больше IFD, тем эффективнее инспекция при неизменном качестве рабочего продукта и наоборот. Это показано в формуле 1:

(1)

1. Стратегическая цель метрики – повышение качества разрабатываемого программного обеспечения.
2. Объект измерения – инспекция.
3. Измеряемый атрибут – плотность найденных ошибок в ходе инспекции.
4. Единица измерения – ошибка / <страница, требование, LOC (lines of code), тест>.
5. Целью предприятия является снижение IFD, чем меньше IFD, тем лучше.

# Разработка модели состояний задач

Множество состояний задач, принятых на итоговом проекте. Разработка регламента создания задач и перевода их из состояния в состояние.

1. Перечень возможных состояний задачи и их интерпретация
   1. Backlog
      1. Задача добавлена в проект, но ещё не утверждена к работе.
      2. Интерпретация: задача требует уточнения требований, приоритезации или проверки.
   2. To Do
      1. Задача утверждена для выполнения и ожидает начала работы.
      2. Интерпретация: приоритет задачи определён, она включена в ближайший спринт.
   3. In Progress
      1. Над задачей ведётся активная работа.
      2. Интерпретация: задача выполняется исполнителем согласно требованиям.
   4. Review
      1. Работа по задаче завершена и передана на проверку.
      2. Интерпретация: ожидается подтверждение выполнения требований или исправление замечаний.
   5. Done
      1. Задача полностью выполнена, проверена и утверждена.
      2. Интерпретация: результат интегрирован в проект.
   6. Blocked
      1. Задача приостановлена из-за внешних факторов (зависимость от другой задачи, отсутствие ресурсов и т.д.).
      2. Интерпретация: задача требует разрешения проблемных вопросов.
   7. Closed
      1. Задача закрыта (выполнена или отклонена).
      2. Интерпретация: дальнейшие действия по задаче не требуются.
2. Правила создания новой задачи
   1. Кто создает:
      1. Руководитель проекта – для постановки целей и ключевых задач.
      2. Разработчики – при выявлении технических, уточняющих или вспомогательных задач.
      3. Участники инспекций – при обнаружении замечаний в ходе проверки.
   2. Когда создается:
      1. При выявлении новой функциональной потребности.
      2. В случае обнаружения дефекта или проблемы.
      3. При необходимости уточнить существующую задачу.
   3. Требования к задаче:
      1. Чёткое название, отражающее суть задачи.
      2. Описание с критериями выполнения (Acceptance Criteria).
      3. Метки (Labels), указывающие тип задачи (bug, feature, improvement, documentation).
      4. Назначение исполнителя (Assignee).
      5. Привязка к вехе (Milestone).
3. Правила перехода задачи из состояния в состояние
   1. Backlog ↔ To Do:
      1. Требования уточнены, задача согласована руководителем проекта.
      2. Задача приоритезированы.
   2. To Do → In Progress:
      1. Исполнитель готов приступить к задаче.
      2. Все блокирующие зависимости сняты.
   3. In Progress → Review:
      1. Работа завершена, выполнены все Acceptance Criteria.
      2. Результат доступен для проверки.
   4. Review → Done:
      1. Задача проверена, все замечания устранены.
      2. Финальная версия подтверждена руководителем или рецензентом.
   5. Review ↔ In Progress:
      1. Замечания не устранены, требуется доработка.
   6. Blocked → To Do:
      1. Проблемы, блокирующие выполнение задачи, решены.

Критерии перевода в состояние:

Каждое состояние требует выполнения описанных выше условий.

Переходы допускаются только между логически связанными состояниями.

Возможные переходы:

1. Backlog ↔ To Do
2. To Do → In Progress
3. In Progress → Review
4. Review ↔ In Progress
5. Review → Done
6. Any → Blocked
7. Blocked → To Do

Результаты переходов:

1. При каждом переходе обновляется статус задачи.
2. Исполнители уведомляются об изменениях через комментарии в GitHub Issues.

# Разработка презентации проекта

Этот раздел включает описание создания презентации проекта, которая будет использована для демонстрации всех ключевых аспектов работы над проектом. Презентация позволяет наглядно представить результаты работы и цели, которых удалось достичь.



Рисунок 2 – Слайд презентации «Приветствие»

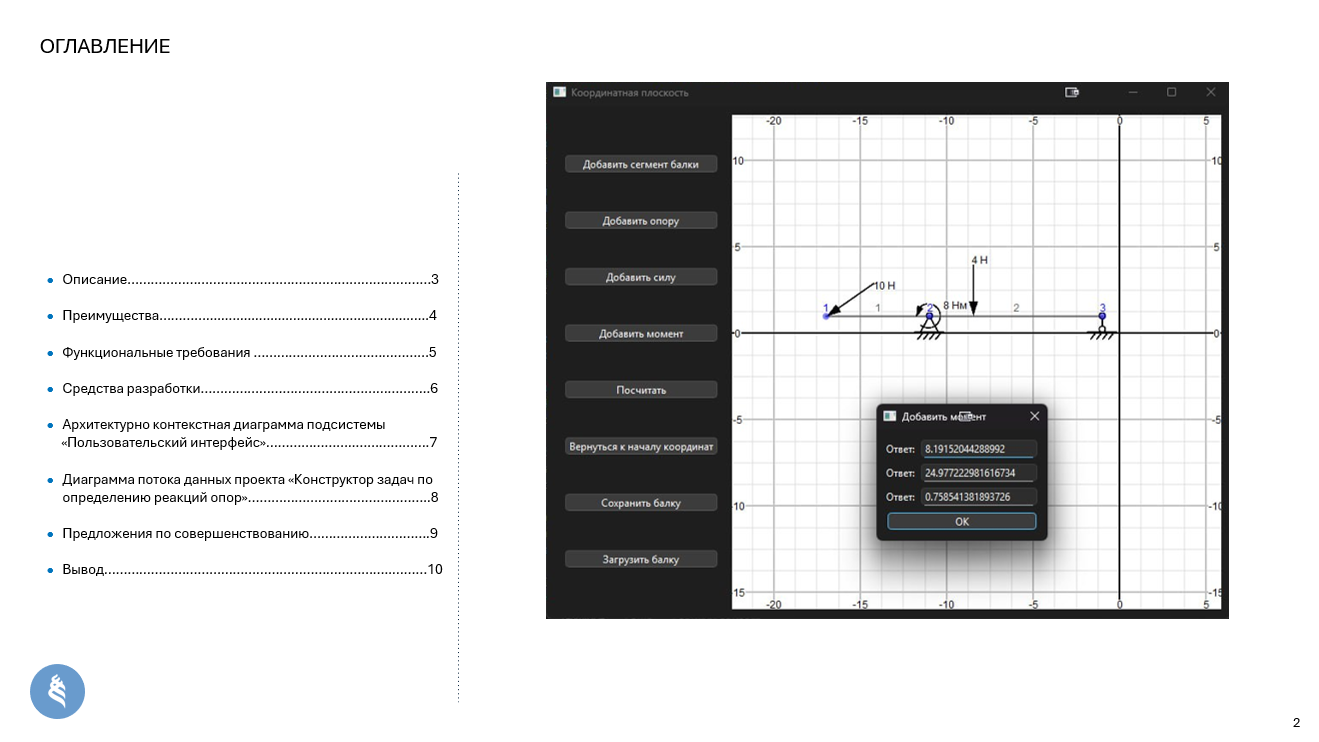


Рисунок 3 – Слайд презентации «Оглавление»

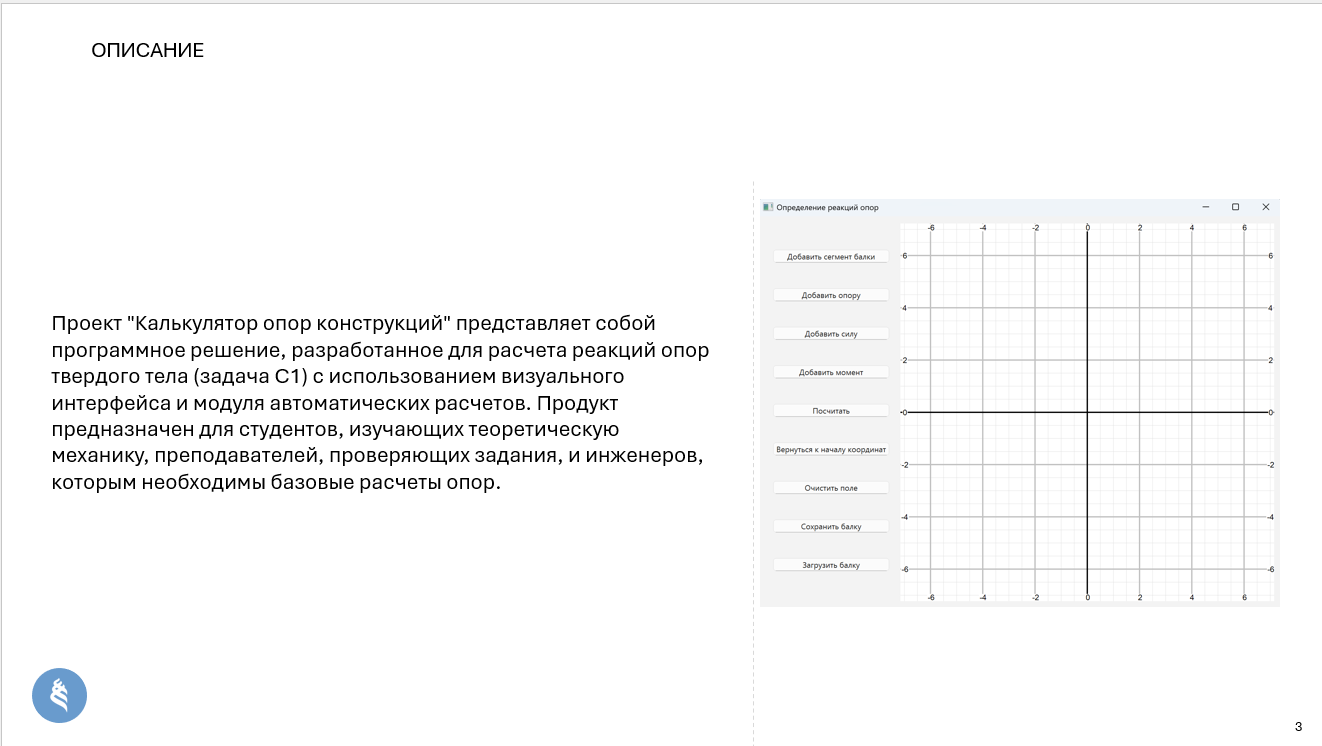


Рисунок 4 – Слайд презентации «Описание»



Рисунок 5 – Слайд презентации «Преимущества»

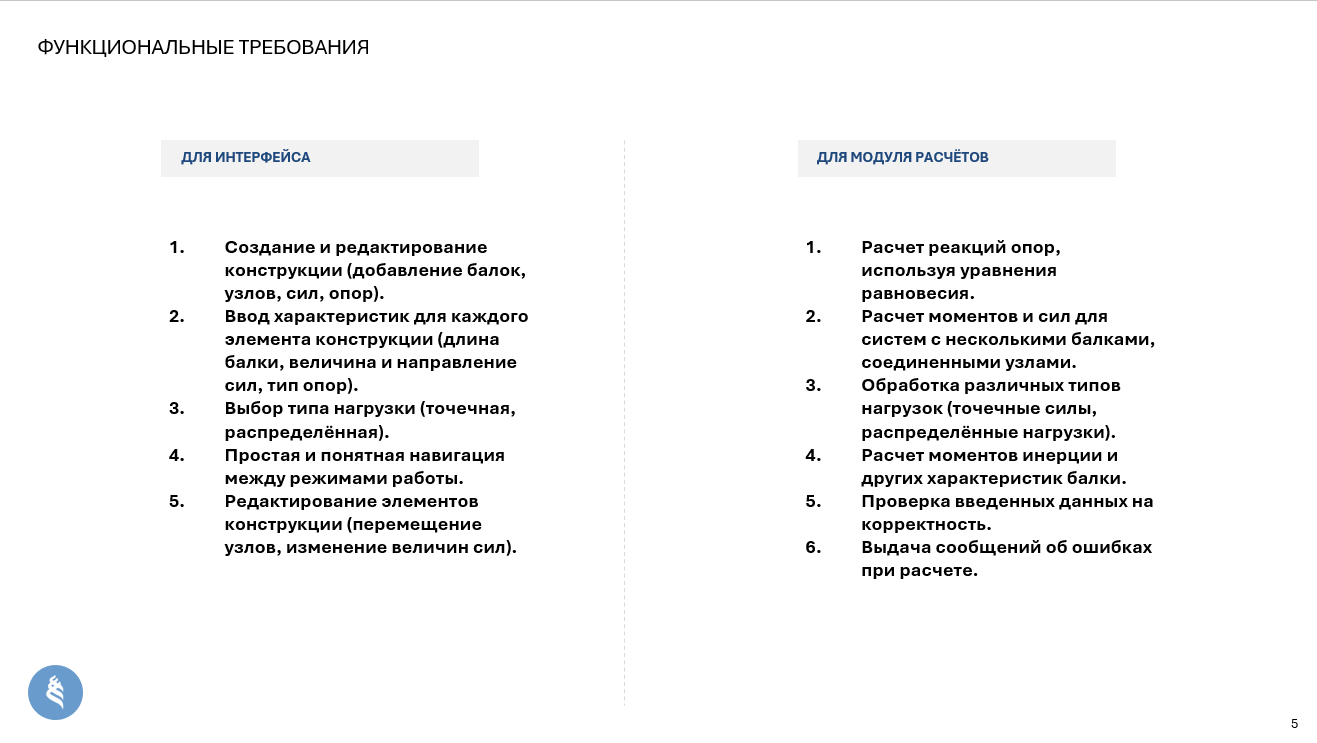


Рисунок 6 – Слайд презентации «Функциональные требования»

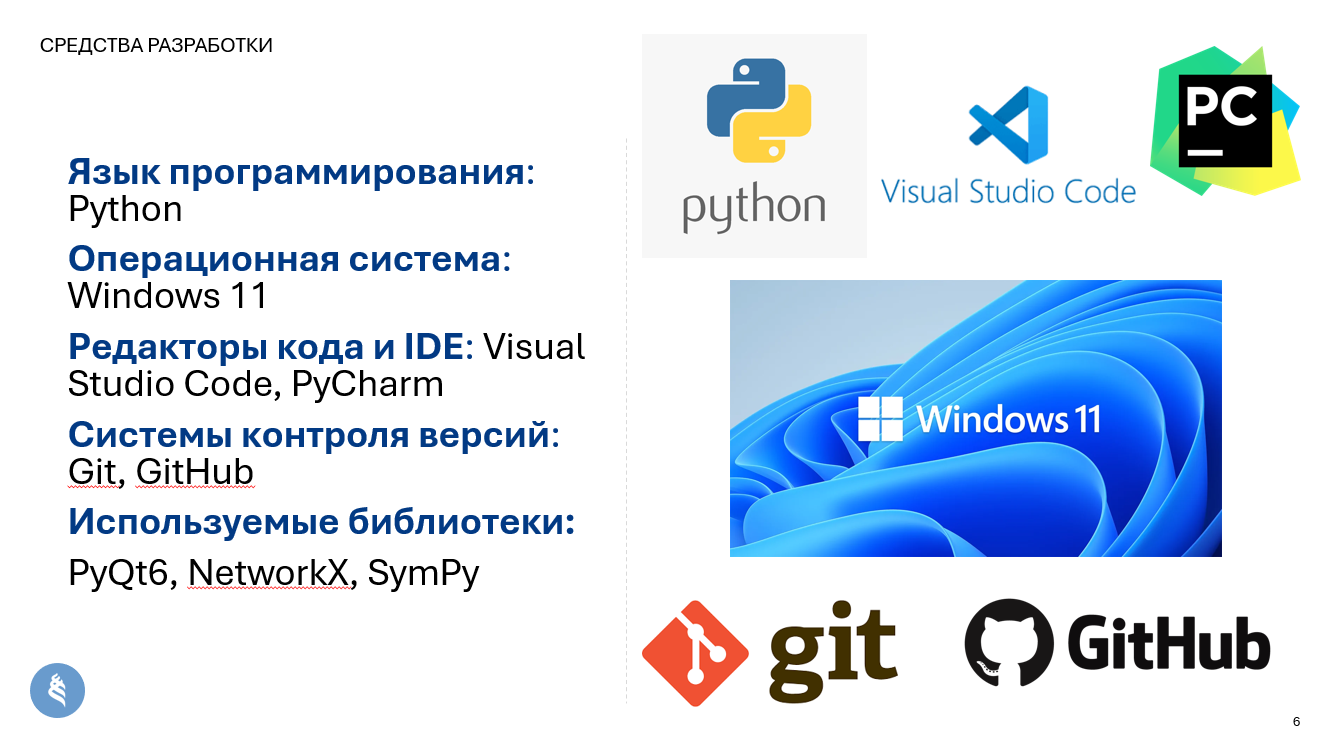


Рисунок 7 – Слайд презентации Средства разработки

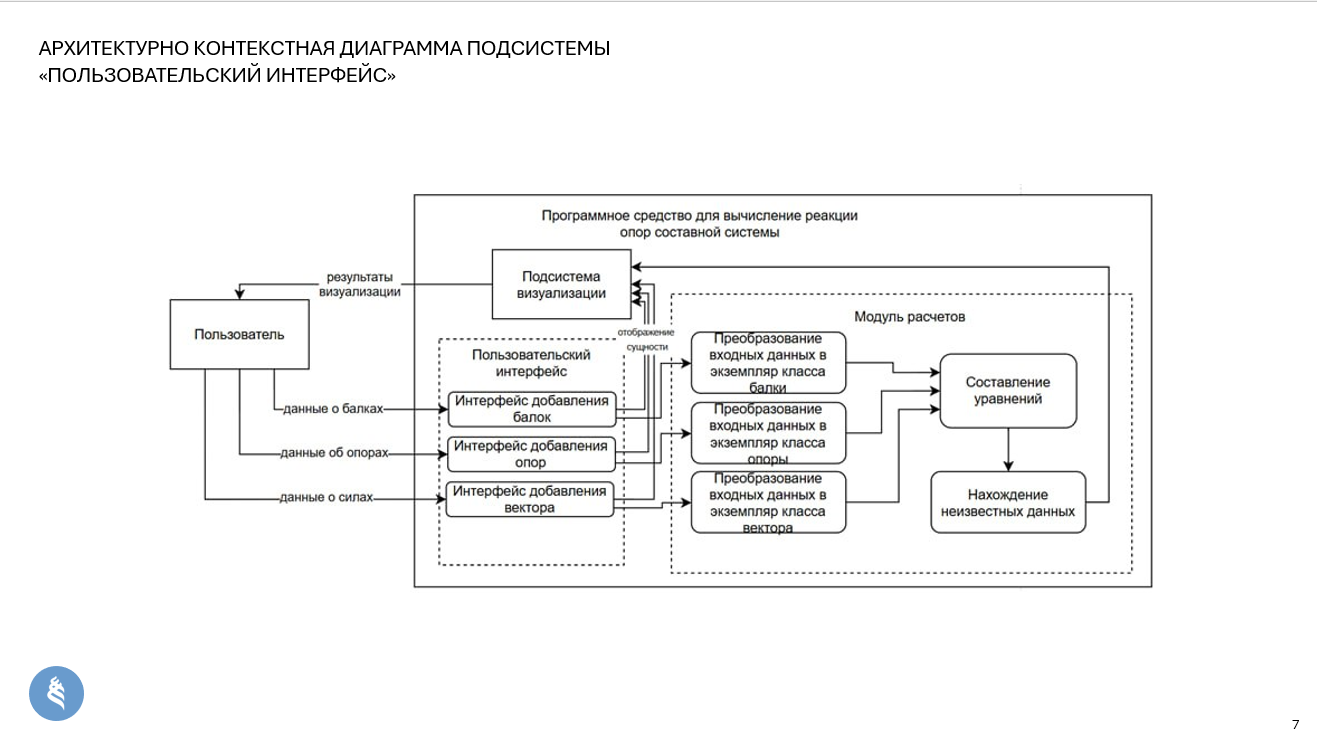


Рисунок 8 – Слайд презентации «Архитектурно контекстная диаграмма подсистемы Пользовательский интерфейс»

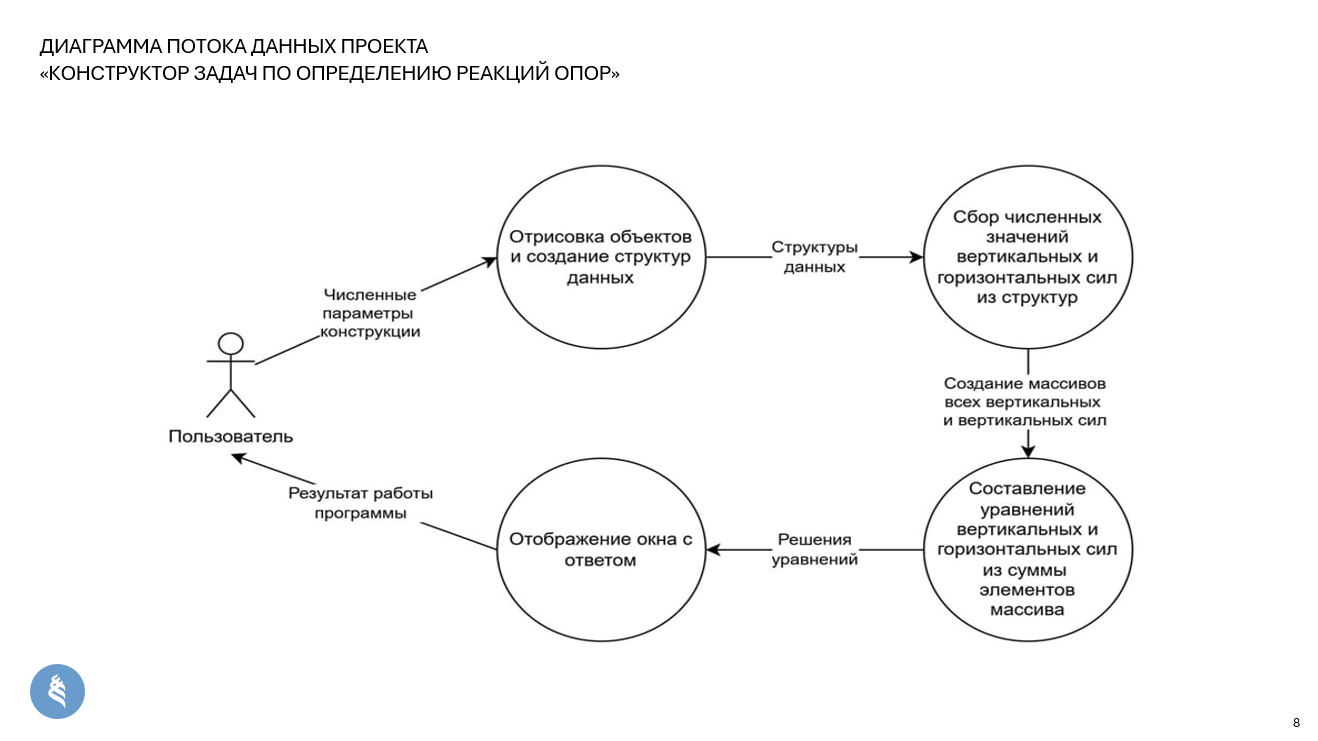


Рисунок 9 – Слайд презентации «Диаграмма потока данных проекта»

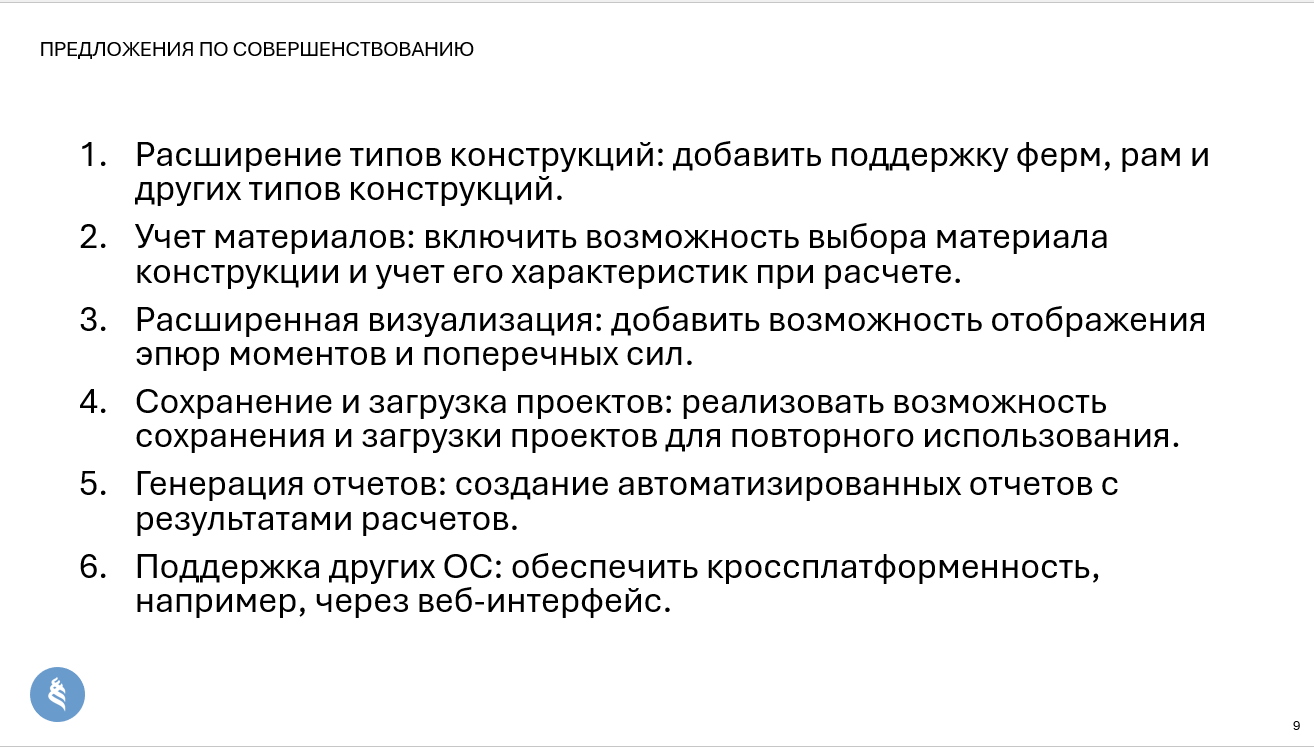


Рисунок 10 – Слайд презентации «Предложение по совершенствованию»

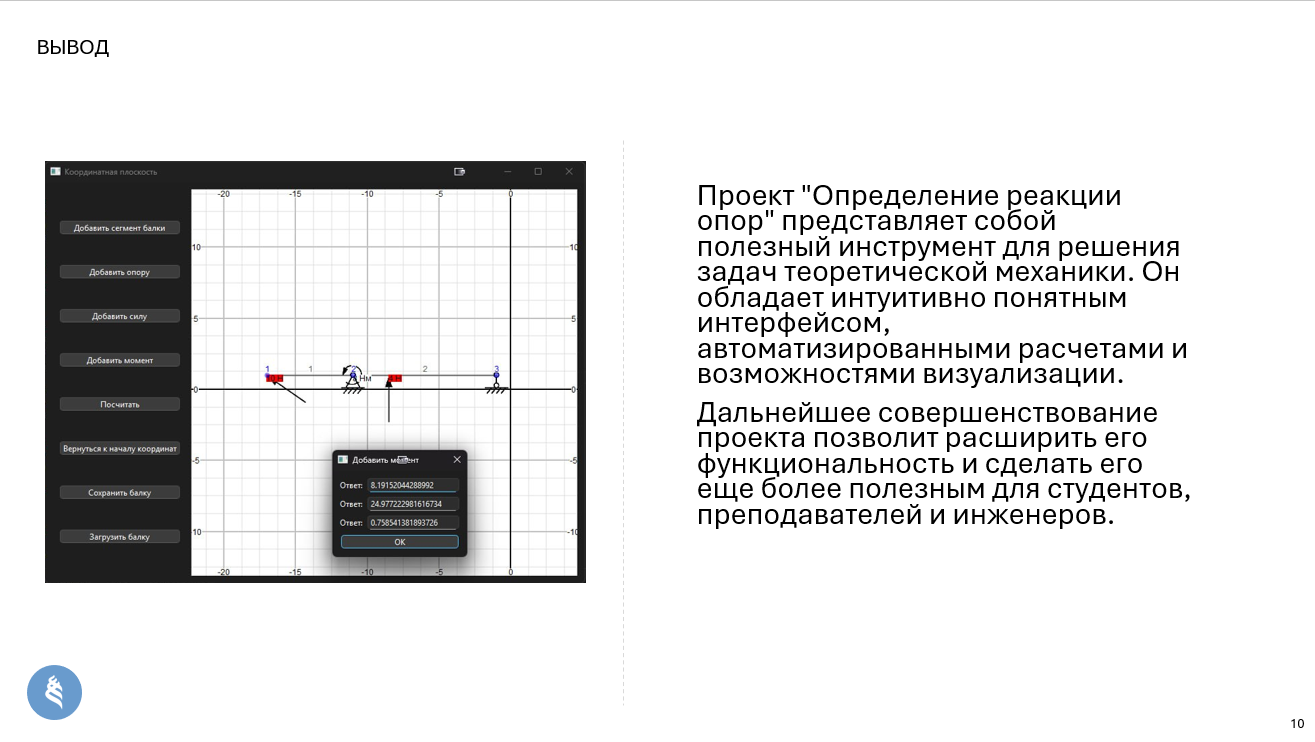


Рисунок 11 – Слайд презентации «Вывод»



Рисунок 12 – Слайд презентации «Конец»

# Разработка требований к проекту

Продукт позволяет рассчитывать реакции опор конструкций (задача C1) через визуальный интерфейс и модуль автоматических расчётов.

Программный продукт предназначен для:

1. Студентов, изучающих теоретическую механику.
2. Преподавателей, проверяющих задания студентов.
3. Инженеров, которым необходимы базовые расчеты опор.

Приложение работает на настольных системах с операционной системой Windows. Основной язык интерфейса: русский.

Определения и сокращения

C1: Задача на определение реакций опор твердого тела.

## 5.1 Общая характеристика

В данном разделе приводится описание общей характеристики программного продукта, включая основные функциональные возможности, ограничения и предположения. Это позволяет дать чёткое представление о том, какие задачи решает система, а также о возможных ограничениях и зависимостях от других технологий или компонентов.

### 5.2 Функциональность системы

1. Пользовательский интерфейс позволяет пользователю:
   1. Задавать конструкцию (опоры, силы, стержни) визуально.
   2. Запускать расчёты.
2. Модуль расчетов:
   1. Рассчитывает реакции опор на основе введенных данных.
   2. Выводит числовые результаты для задачи C1.
3. Подсистема визуализации:
   1. Отображает схему конструкции и направления сил.
   2. Позволяет взаимодействовать с элементами схемы (перемещать, редактировать).

### 5.3 Ограничения

1. Программа предназначена для работы на настольной системе Windows 11.
2. Ввод данных ограничивается двумя-тремя телами и фиксированным количеством сил (максимум 3).
3. Визуализация ограничена плоскими и пространственными схемами.

### 5.4 Предположения и зависимости

1. Пользователь должен обладать базовыми знаниями в теоретической механике.
2. Для работы требуется установленный Python (для версий без компиляции).

## 5.5 Требования к подсистемам

В этом разделе приводятся детализированные требования к различным подсистемам проекта, которые составляют основу для функционирования всего приложения. Каждая подсистема имеет свои функциональные и нефункциональные требования, которые обеспечивают работу системы в целом. Разделение на подсистемы позволяет четко распределить задачи и отвечает за выполнение определённых функций, таких как взаимодействие с пользователем, расчёт реакций опор и визуализация результатов.

### 5.5.1 Требования к подсистеме «Пользовательский интерфейс»

Функциональные требования:

1. REQ\_UI\_001

Приложение должно позволять пользователю создавать и редактировать конструкцию (добавлять балки, узлы, силы, опоры).

1. REQ\_UI\_002

Интерфейс должен поддерживать ввод характеристик для каждого элемента конструкции (длина балки, величина и направление сил, тип опор и т.д.).

1. REQ\_UI\_003

Должна быть возможность выбирать тип нагрузки (точечная, распределённая).

1. REQ\_UI\_004

Необходимо реализовать простую и понятную навигацию, чтобы пользователь мог легко переключаться между режимами работы (создание конструкции и расчёт).

1. REQ\_UI\_005

Пользователь может редактировать элементы конструкции (перемещать узлы, изменять величины сил).

Нефункциональные требования:

1. REQ\_UI\_006

Интерфейс должен быть интуитивно понятным и не требовать длительного обучения.

1. REQ\_UI\_007

Программа должна реагировать на действия пользователя без значительных задержек (менее 1 секунды для большинства операций).

1. REQ\_UI\_008

Интерфейс должен быть адаптивным для использования на экранах разного размера (от ноутбуков до больших мониторов).

### 5.5.2 Требования к подсистеме «Модуль расчетов»

Функциональные требования:

1. REQ\_CALC\_001

Модуль должен рассчитывать реакции опор, используя уравнения равновесия.

1. REQ\_CALC\_002

Модуль должен поддерживать расчёт моментов и сил для систем с несколькими балками, соединёнными узлами.

1. REQ\_CALC\_003

Должен быть реализован механизм обработки различных типов нагрузок (точечные силы, распределённые нагрузки).

1. REQ\_CALC\_004

Должна быть возможность расчёта моментов инерции и других характеристик балки.

1. REQ\_CALC\_005

Модуль должен проверять введённые данные на корректность.

1. REQ\_CALC\_006

При ошибках расчета программа должна выдавать пользователю сообщение с указанием причины.

Нефункциональные требования:

1. REQ\_CALC\_007

Алгоритм расчёта должен быть оптимизирован для конструкций с большим количеством узлов и сил (время выполнения не должно превышать 1 секунды для систем до 100 элементов).

1. REQ\_CALC\_008

Программа должна рассчитывать реакции опор с точностью до двух знаков после запятой.

1. REQ\_CALC\_009

Модуль должен быть гибким и легко масштабируемым для добавления новых функций (например, расчёт деформаций, 3D-расчёты).

### 5.5.3 Требования к подсистеме «Подсистема визуализации»

Функциональные требования:

1. REQ\_VIS\_001

Программа должна визуализировать конструкцию, включая узлы, балки, силы и опоры.

1. REQ\_VIS\_002

Должна быть возможность масштабировать и перемещать изображение конструкции.

1. REQ\_VIS\_003

Сила и её направление должны быть визуализированы с указанием величины (например, стрелка с числом рядом).

1. REQ\_VIS\_004

Результаты расчётов (реакции опор, распределение нагрузок) должны отображаться на графике или в текстовом формате, привязанном к конструкции.

1. REQ\_VIS\_005

Подсистема должна поддерживать экспорт изображения конструкции в виде файла (PNG, JPEG, SVG).

Нефункциональные требования:

1. REQ\_VIS\_006

Отображение должно быть плавным даже для сложных конструкций (до 100 элементов).

1. REQ\_VIS\_007

Визуализация должна быть чёткой и разборчивой, с использованием цветовой схемы для различения типов элементов (опоры, силы, реакции).

1. REQ\_VIS\_008

Визуализация должна автоматически обновляться при изменении параметров пользователем.

# Разработка архитектуры проекта

Здесь обсуждаются вопросы проектирования архитектуры системы, определяются основные компоненты, а также их взаимодействие. Важно создать такую архитектуру, которая обеспечит масштабируемость и гибкость проекта, позволяя легко добавлять новые функции.

## 6.1 Архитектурно-контекстная диаграмма системы

Архитектура проекта «Конструктор задач по определению реакции опор» основана на модульном подходе и включает три основные подсистемы: пользовательский интерфейс, модуль расчётов и подсистему визуализации. Пользовательский интерфейс обеспечивает удобное создание и редактирование конструкции, модуль расчётов выполняет автоматизированное определение реакций опор с использованием методов символьной математики, а подсистема визуализации отвечает за наглядное отображение схемы конструкции и результатов расчётов. Такое разделение упрощает развитие, поддержку и тестирование системы.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

Рисунок 13 – Архитектурно контекстная диаграмма 1-го уровня

Программный продукт состоит из следующих подсистем:

1. Пользовательский интерфейс
2. Модуль расчетов
3. Подсистема визуализации

## 6.2 Диаграмма потока данных проекта «Конструктор задач по определению реакции опор»

Диаграмма потока данных проекта «Конструктор задач по определению реакции опор» отражает основные этапы обработки информации внутри системы. Пользователь создаёт конструкцию через интерфейс, данные о конструкции передаются в модуль расчётов для построения уравнений равновесия и вычисления реакций опор, после чего результаты возвращаются в интерфейс для отображения пользователю и визуализации на графическом представлении конструкции.



Рисунок 14 – Диаграмма потока данных проекта  
«Конструктор задач по определению реакции опор»

## 6.3 Use-case диаграмма программной системы

Для описания взаимодействия пользователя с системой «Конструктор задач по определению реакции опор» была разработана диаграмма вариантов использования. Она отображает основные сценарии работы пользователя, включая создание и редактирование конструкции, выполнение расчётов, управление файлами и взаимодействие с визуальным представлением конструкции.

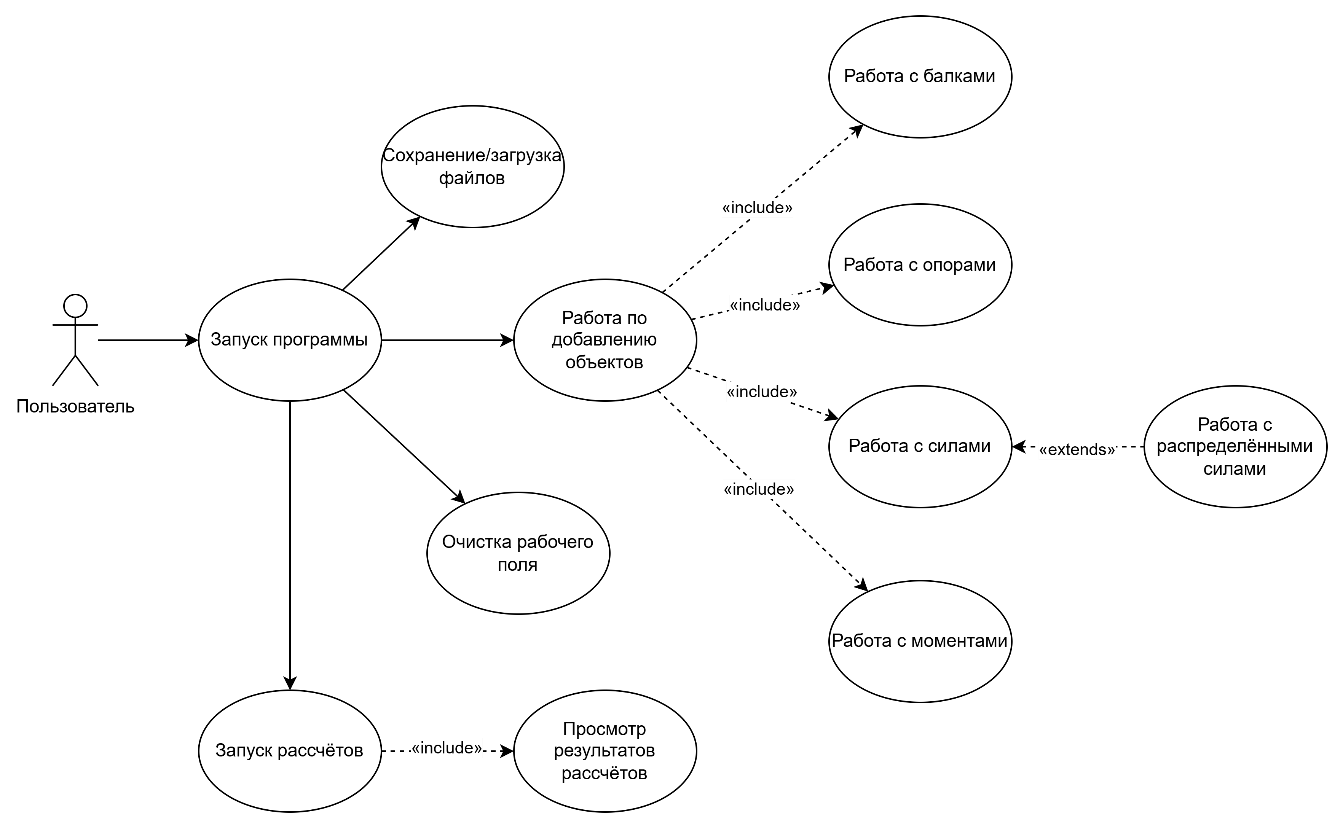


Рисунок 15 – Use-case диаграмма программной системы

# Система метрик для оценки эффективности процесса разработки

Этот раздел посвящён системе метрик, с помощью которых можно оценить эффективность процесса разработки. Оценка таких факторов, как время цикла и эффективность потока, позволяет своевременно вносить улучшения и повышать качество разработки.

## 7.1 Время цикла (Cycle Time)

Описание: эта метрика измеряет время, которое задача проводит в активной разработке, от момента начала работы до завершения. Она не учитывает время ожидания до начала работы над задачей.

Дата перехода задачи в статус «Завершено»‎‎ – дата перехода в статус «В работе»

Цель: позволяет оценить скорость работы команды и эффективность процесса разработки. Уменьшение времени цикла обычно указывает на повышение производительности команды

## 7.2 Эффективность потока (Flow Efficiency)

Описание: это метрика, которая показывает, какая часть времени задачи действительно затрачивается на активную работу, в сравнении с общим временем, проведенным в системе. Измеряется как процент активной работы от общего времени Lead Time (Формула 2).

Формула:

(2)

Цель: выявление «мёртвого времени» и узких мест в процессе разработки. Понимание этой метрики помогает оптимизировать процессы и улучшить планирование качества программного продукта.

Эта метрика качества оценивает надежность (Reliability) ПС.

Описание: надежность измеряет способность системы функционировать без сбоев в течение определенного времени или при выполнении определенных задач.

# Разработка перечня задач проекта

В данном разделе представлен подробный перечень задач, разбитых по подсистемам, с указанием наименования, описания и этапов реализации.

## 8.1 Задачи для подсистемы «Пользовательский интерфейс»

1. Задача UI\_001

Название: Проектирование пользовательского интерфейса.

Описание: Разработать графический интерфейс для взаимодействия пользователя с балкой: добавление сегментов, опор, сил, моментов, сохранение и загрузка балок. Использовать библиотеку PyQt6.

1. Задача UI\_002

Название: Реализация механизма отображения расчетов.

Описание: Реализовать окно для отображения результатов расчёта реакций опор в отдельном диалоге.

1. Задача UI\_003

Название: Реализация сохранения и загрузки балок.

Описание: Разработать механизм сохранения текущего состояния балки в файл и загрузки сохранённой балки.

## 8.2 Задачи для подсистемы «Отрисовка сетки и объектов»

1. Задача GR\_001

Название: Реализация отрисовки сетки координат.

Описание: Реализовать отображение координатной сетки с динамическим масштабированием и перемещением в окне приложения.

1. Задача GR\_002

Название: Отрисовка балки и ее сегментов.

Описание: Реализовать визуализацию сегментов балки на сетке.

1. Задача GR\_003

Название: Отрисовка сил, моментов и опор.

Описание: Отобразить приложенные силы, моменты и типы опор на графическом интерфейсе.

## 8.3 Задачи для подсистемы «Диалоги добавления объектов»

1. Задача DI\_001

Название: Диалог добавления сегмента балки.

Описание: Реализовать окно для ввода координат начальной и конечной точек сегмента балки.

1. Задача DI\_002

Название: Диалог добавления опоры.

Описание: Реализовать окно выбора узла и типа опоры.

1. Задача DI\_003

Название: Диалог добавления силы.

Описание: Реализовать окно добавления силы: сегмент, отступ, значение, угол и длина распределения.

1. Задача DI\_004

Название: Диалог добавления момента.

Описание: Реализовать окно добавления момента: сегмент, отступ и значение момента.

## 8.4 Задачи для подсистемы «Математический расчет реакций»

1. Задача MA\_001

Название: Формирование системы уравнений равновесия.

Описание: Разработать алгоритм составления уравнений равновесия по силам и моментам с учётом всех приложенных воздействий.

1. Задача MA\_002

Название: Решение системы уравнений.

Описание: Реализовать решение системы уравнений методом символьных вычислений (с использованием библиотеки sympy) для нахождения неизвестных реакций.

1. Задача MA\_003

Название: Формирование итогового отчета результатов.

Описание: Вывести расчетные значения реакций в удобной для пользователя форме.

## 8.5 Задачи для подсистемы «Обработка ошибок»

1. Задача ER\_001

Название: Обработка пользовательских ошибок ввода.

Описание: Обрабатывать ошибки ввода данных пользователем с выводом понятных сообщений.

1. Задача ER\_002

Название: Обработка ошибок построения балки.

Описание: Предусмотреть проверку на некорректные балки: несвязанные сегменты, отсутствие опор, нулевая длина балки и т.д.

## 8.6 Задачи для системы в целом

1. Задача SYS\_001

Название: Сборка и запуск проекта.

Описание: Реализовать старт основного окна программы через запуск скрипта main.py и корректную инициализацию всех компонентов.

# Разработка рекомендаций по кодированию

Для создания качественного кода на языке программирования Python и библиотеке PyQt6, обладающего такими свойствами, как удобочитаемость (readability) и понятность (understandability), необходимо следовать хорошо определённым стандартам. Особенно это актуально при коллективной разработке. Стандарты кодирования способствуют написанию единого стиля кода и упрощают его поддержку.

## 9.1 Основные правила кодирования

В данном разделе рассматриваются ключевые правила кодирования, которым следует придерживаться при разработке проекта. Соблюдение стандартов кодирования не только улучшает читаемость и поддерживаемость кода, но и повышает его качество, снижая вероятность возникновения ошибок.

### 9.1.1 Форматирование

1. Использовать четыре пробела для отступов. Не использовать табуляцию или два пробела.
2. Все import писать в начале файла отдельными строками, отсортированными по группам: стандартные библиотеки, внешние библиотеки, собственные модули.
3. Отделять блок импортов одной пустой строкой от остального кода.
4. Не использовать from module import \*.
5. Внутри скобок не ставить пробелы после открывающей и перед закрывающей скобками:

method(arg1, arg2) # Правильно

method( arg1, arg2 ) # Неправильно

1. Пробел ставится после, но не перед запятыми, двоеточиями и точками с запятой:

if x == 4:

print(x, y) # Правильно

if x == 4 :

print(x,y) # Неправильно

1. Бинарные операторы (=, +, -, \*, ==, !=, <, >, <=, >=, and, or) окружать ровно одним пробелом с каждой стороны.
2. Одна инструкция на строку:

x = 3

func(10)

# а не: x = 3; func(10)

1. После if, for, while, def всегда переносить тело на новую строку.

### 9.1.2 Комментарии

1. Комментарии должны соответствовать реальному коду.
2. Однострочные комментарии пишутся после кода или на отдельной строке, начинаются с # и одного пробела.
3. Для многострочных комментариев использовать тройные кавычки «»» комментарий «»», если это часть документации (docstring).
4. Комментарии должны быть информативными и лаконичными.

### 9.1.3 Имена

1. Имена переменных, функций и методов — в стиле snake\_case (example\_variable, calculate\_force).
2. Имена классов — в стиле CamelCase (BeamSegment, GridWidget).
3. Имена констант — заглавными буквами с подчёркиванием (MAX\_SCALE, DEFAULT\_MARGIN).
4. Использовать говорящие английские имена, избегать транслита и сокращений:

node\_count = 5 # Правильно

kolvo\_uzlov = 5 # Неправильно

### 9.1.4 Функции

1. Разделять определения функций и методов двумя пустыми строками.
2. Внутри функций разделять логические блоки одной пустой строкой для улучшения читаемости.
3. Стараться не делать функции длиннее 40–50 строк. Разбивать на вспомогательные функции при необходимости.

# Разработка плана тестирования проекта

Этот раздел включает в себя план тестирования, который охватывает все подсистемы проекта. Мы опишем, какие тесты будут выполнены, а также ожидаемые результаты, чтобы удостовериться, что каждая подсистема работает в соответствии с требованиями.

## 10.1 Тесты для тестирования подсистемы «Пользовательский интерфейс»

Тест TEST\_UI\_01

Тестируемые требования: REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_003, REQ\_UI\_004, REQ\_UI\_005, REQ\_UI\_006, REQ\_UI\_007, REQ\_UI\_008

Описание теста: Открыть приложение. Проверить возможность создания конструкции (добавление сегментов, опор, сил, моментов). Проверить ввод всех параметров (длина, угол, величина сил). Проверить смену режимов создания конструкции и расчета. Проверить редактирование элементов.

Ожидаемый результат: Все элементы создаются, редактируются и удаляются корректно. Интерфейс реагирует на действия без заметных задержек. Отображение корректно на экранах разных размеров.

## 10.2 Тесты для тестирования подсистемы «Модуль расчетов»

Тест TEST\_CALC\_01

Тестируемые требования: REQ\_CALC\_001, REQ\_CALC\_002, REQ\_CALC\_003, REQ\_CALC\_004, REQ\_CALC\_005, REQ\_CALC\_006, REQ\_CALC\_007, REQ\_CALC\_008, REQ\_CALC\_009

Описание теста: Создать различные конструкции с точечными и распределёнными нагрузками, различными типами опор и узлами. Выполнить расчёт реакций опор. Проверить правильность расчётов вручную. Проверить обработку некорректных данных (например, силы с отрицательной длиной).

Ожидаемый результат: Реакции опор рассчитываются корректно, результаты выводятся с точностью до двух знаков после запятой. При ошибках отображается соответствующее сообщение.

## 10.3 Тесты для тестирования подсистемы «Подсистема визуализации»

Тест TEST\_VIS\_01

Тестируемые требования: REQ\_VIS\_001, REQ\_VIS\_002, REQ\_VIS\_003, REQ\_VIS\_004, REQ\_VIS\_005, REQ\_VIS\_006, REQ\_VIS\_007, REQ\_VIS\_008

Описание теста: Нарисовать конструкцию с несколькими узлами, сегментами, силами и опорами. Проверить масштабирование, перемещение изображения. Проверить отображение стрелок с подписями значений сил и моментов. Проверить экспорт изображения конструкции.

Ожидаемый результат: Визуализация конструкции чёткая и плавная, масштабирование работает корректно, значения отображаются рядом с элементами. Экспорт изображения производится без ошибок.

## 10.4 Матрица покрытия тестами требований

Матрица покрытия требований тестами позволяет убедиться, что каждый функциональный и нефункциональный аспект проекта протестирован.

В соответствии с написанными требованиями и тестами на Рисунке 11 представлена матрица покрытия тестами требований.

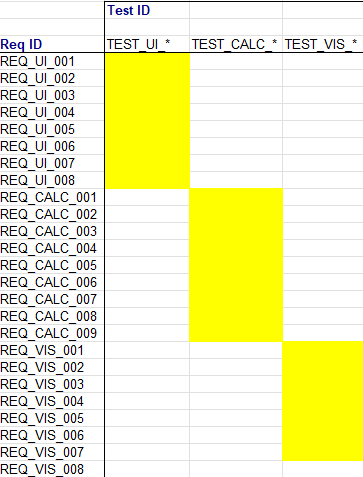


Рисунок 16 – Матрица покрытия тестами требований

## 10.5 Классы эквивалентности

Таблица 1 – Классы эквивалентности для тестирования конструктора задач

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Входные данные** | **Номер правильного класса эквивалентности** | **Правильный класс эквивалентности** | **Номер неправильного класса эквивалентности** | **Неправильный класс эквивалентности** |
| Добавление сегмента балки | (1) | Введены корректные координаты двух разных точек. Узлы не совпадают: координаты узла 1 ≠ координаты узла 2. Длина сегмента > 0. | (2) | Указаны одинаковые координаты узлов. Начальная и конечная точка совпадают. Длина сегмента = 0. |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Входные данные** | **Номер правильного класса эквивалентности** | **Правильный класс эквивалентности** | **Номер неправильного класса эквивалентности** | **Неправильный класс эквивалентности** |
|  |  |  | (3) | Введены координаты за пределами допустимого диапазона: значение по X или Y превышает ±10000. |
| Добавление силы | (4) | Введены положительное значение силы (> 0), угол в пределах 0–359°, отступ находится внутри длины сегмента. | (5) | Указана сила, равная нулю или отрицательная (≤ 0). |
|  |  |  | (6) | Введён некорректный угол (значение не является числом). |
|  |  |  | (7) | Отступ превышает длину сегмента. |
|  |  |  | (8) | Отступ отрицательный (< 0). |
|  |  |  | (9) | Значение силы задано некорректно: нечисловой ввод. |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Входные данные** | **Номер правильного класса эквивалентности** | **Правильный класс эквивалентности** | **Номер неправильного класса эквивалентности** | **Неправильный класс эквивалентности** |
| Добавление момента | (10) | Введено положительное или отрицательное значение момента, отступ находится в пределах длины сегмента. | (11) | Отступ за пределами сегмента или отрицательный. |
| Добавление опоры | (12) | Выбран существующий узел и введён валидный угол (0–359°). | (13) | Попытка добавить опору в несуществующий узел (номер узла отсутствует в модели). |
|  |  |  | (14) | Указан некорректный угол (не число или за пределами диапазона). |
| Сохранение/загрузка балки | (15) | Выбран файл корректного формата .bm, структура соответствует модели балки. | (16) | Выбран файл неверного формата (например, .txt, .docx). |
|  |  |  | (17) | Загружен пустой файл (нет данных о балке). |

Окончание таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Входные данные** | **Номер правильного класса эквивалентности** | **Правильный класс эквивалентности** | **Номер неправильного класса эквивалентности** | **Неправильный класс эквивалентности** |
| Расчёт реакций опор | (18) | Модель балки полная: есть как минимум две опоры, балки соединены, узлы связаны. | (19) | Конструкция без опор: расчёт невозможен. |
|  |  |  | (20) | Конструкция несвязная: существуют независимые сегменты. |

# Тестирование проекта

В данном разделе представлен процесс тестирования проекта, который охватывает тесты как для пользовательского интерфейса, так и для подсистем расчётов и визуализации. Этот процесс позволяет убедиться, что проект работает корректно и соответствует заявленным требованиям.

## Тесты для тестирования подсистемы «Пользовательский интерфейс»

## Тест TEST\_UI\_001

Покрываемые классы эквивалентности: (2)

Тестируемые требования: REQ\_UI\_001

Описание теста: Проверка обработки ошибки при попытке создать сегмент между одним и тем же узлом.

Вводимые данные:

Узел 1: (1, 1)

Узел 2: (1, 1)

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: Отображается всплывающее окно с ошибкой: «Ошибка: Длина сегмента должна быть положительной». Сегмент не создается, интерфейс остается в предыдущем состоянии.

Видимый результат:

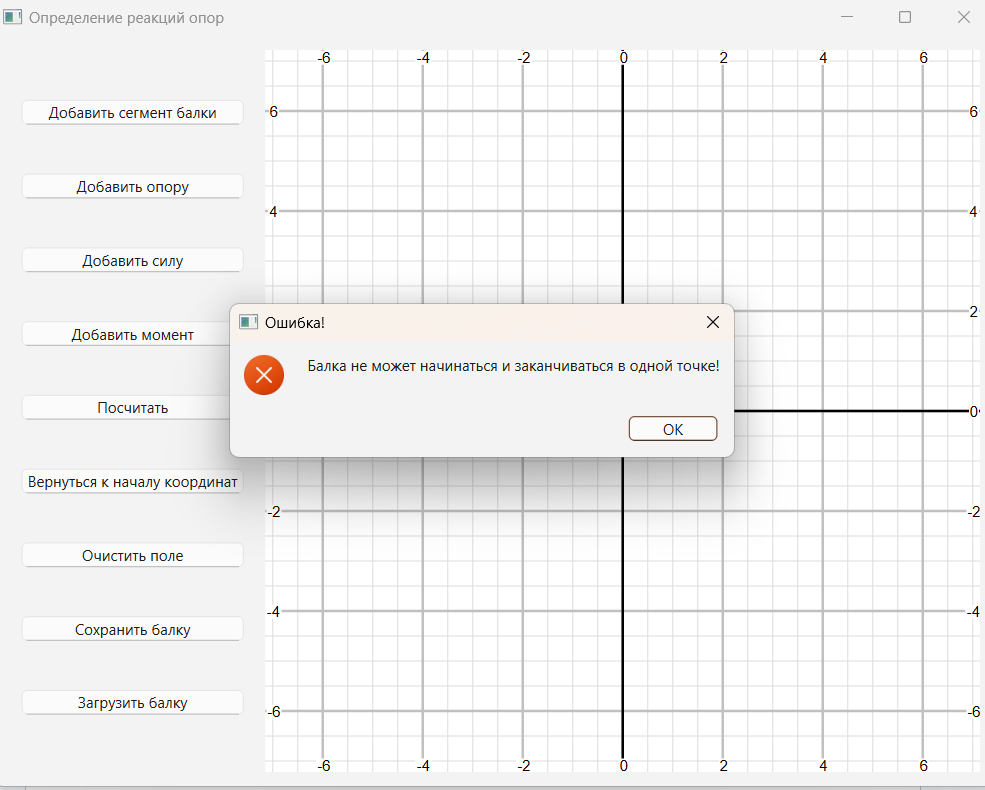


Рисунок 17 – Тест TEST\_UI\_001

Резюме: Тест пройден

## Тест TEST\_UI\_002

Покрываемые классы эквивалентности: (12)

Тестируемые требования: REQ\_UI\_002 (Добавление опоры к узлу)

Описание теста: Добавление жесткой заделки (Fixed Support) на существующий узел.

Вводимые данные:

Номер узла: 1

Тип опоры: Жесткая заделка

Угол: 0 градусов

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: На выбранном узле появляется графическое изображение жесткой заделки (иконка «support0.svg»). Опора визуально фиксируется на узле, пользователь видит её правильное положение.

Видимый результат:

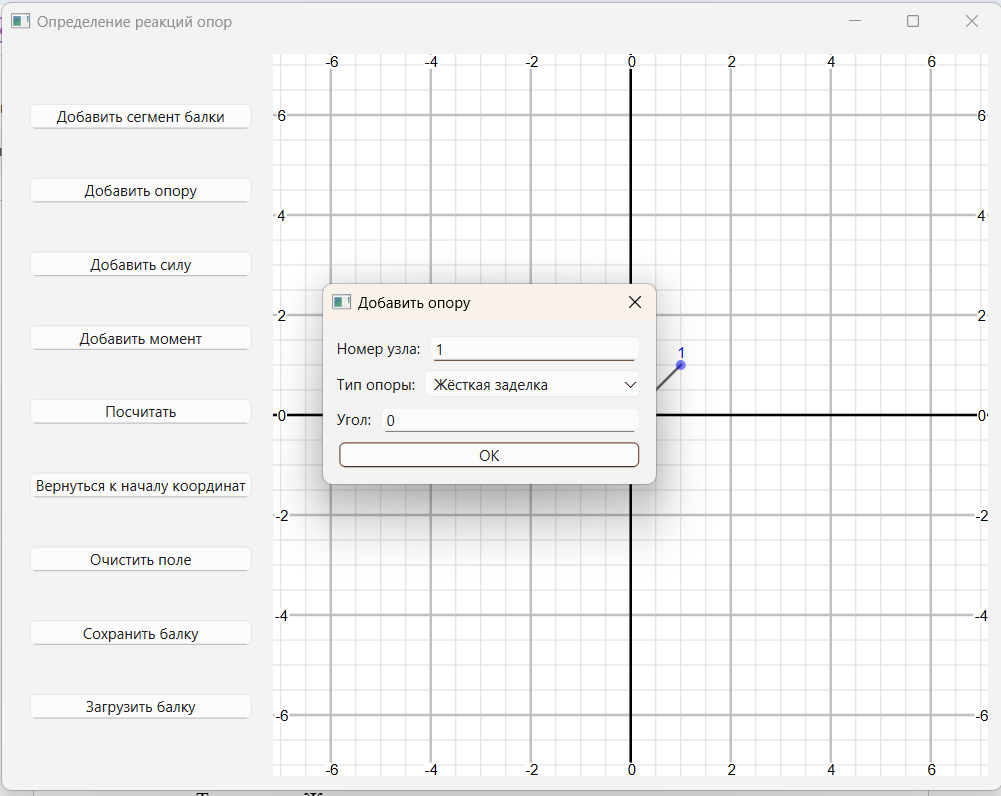


Рисунок 18 – Тест TEST\_UI\_002 (вводимые данные)

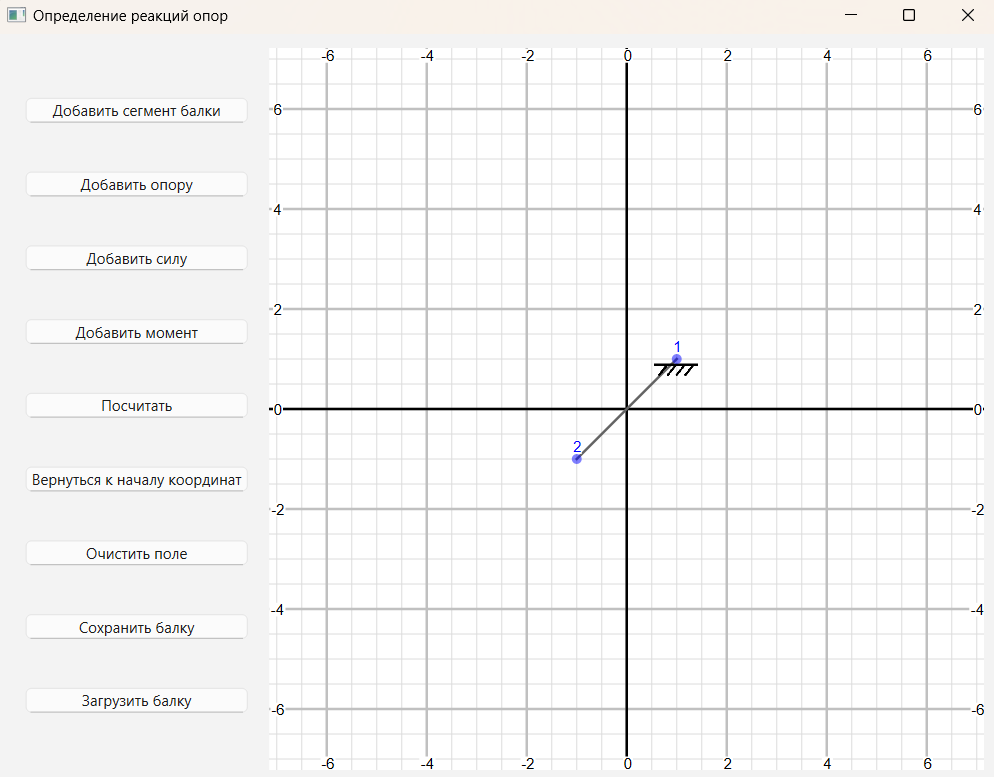
****

Рисунок 19 – Тест TEST\_UI\_002 (результат)

Резюме: Тест пройден

## Тест TEST\_UI\_003

Покрываемые классы эквивалентности: (13)

Тестируемые требования: REQ\_UI\_002 (Добавление опоры к узлу)

Описание теста: Попытка добавить опору на несуществующий узел.

Вводимые данные:

Номер узла: 999 (узел отсутствует в модели)

Тип опоры: Шарнирно-неподвижная

Угол: 0 градусов

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: Отображается всплывающее окно ошибки с текстом: «Ошибка: Узел 999 не существует». Операция отменяется, состояние модели остается неизменным.

Видимый результат:

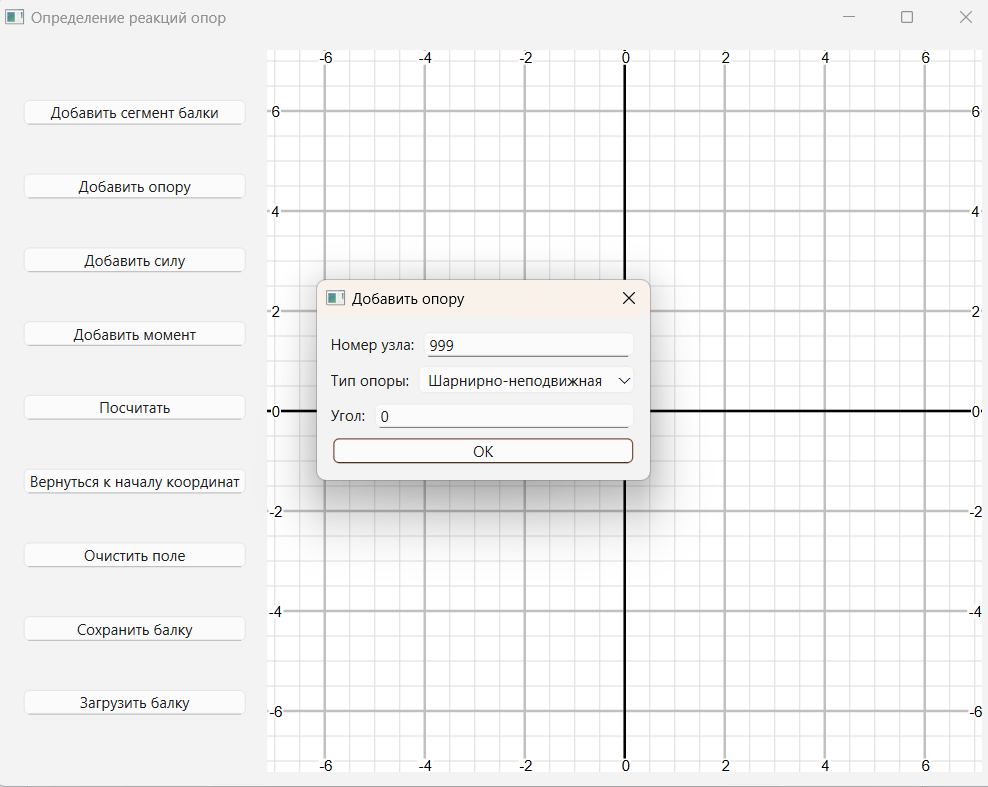


Рисунок 20 – Тест TEST\_UI\_003 (вводимые данные)

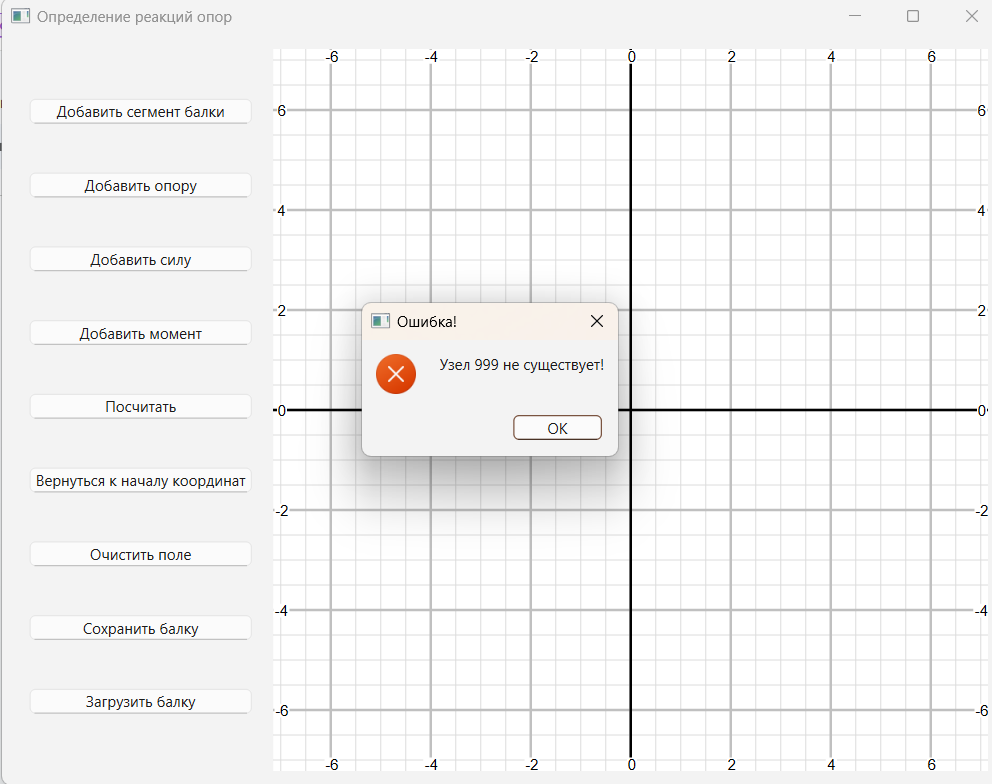


Рисунок 21 – Тест TEST\_UI\_003 (результат)

Резюме: Тест пройден

## Тест TEST\_UI\_004

Покрываемые классы эквивалентности: Нет отдельного класса (Тестирование интерфейса)

Тестируемые требования: REQ\_UI\_003 (Масштабирование и навигация по координатной плоскости)

Описание теста: Проверка увеличения масштаба координатной сетки с помощью колесика мыши.

Вводимые данные:

Операция: Прокрутка колесика мыши вверх 5 раз.

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: Все элементы координатной плоскости увеличиваются пропорционально без смещения. Подписи координат также адаптируются к новому масштабу. Центральная точка остается в центре экрана.

Видимый результат:

****

Рисунок 22 – Тест TEST\_UI\_004 (начальный масштаб)

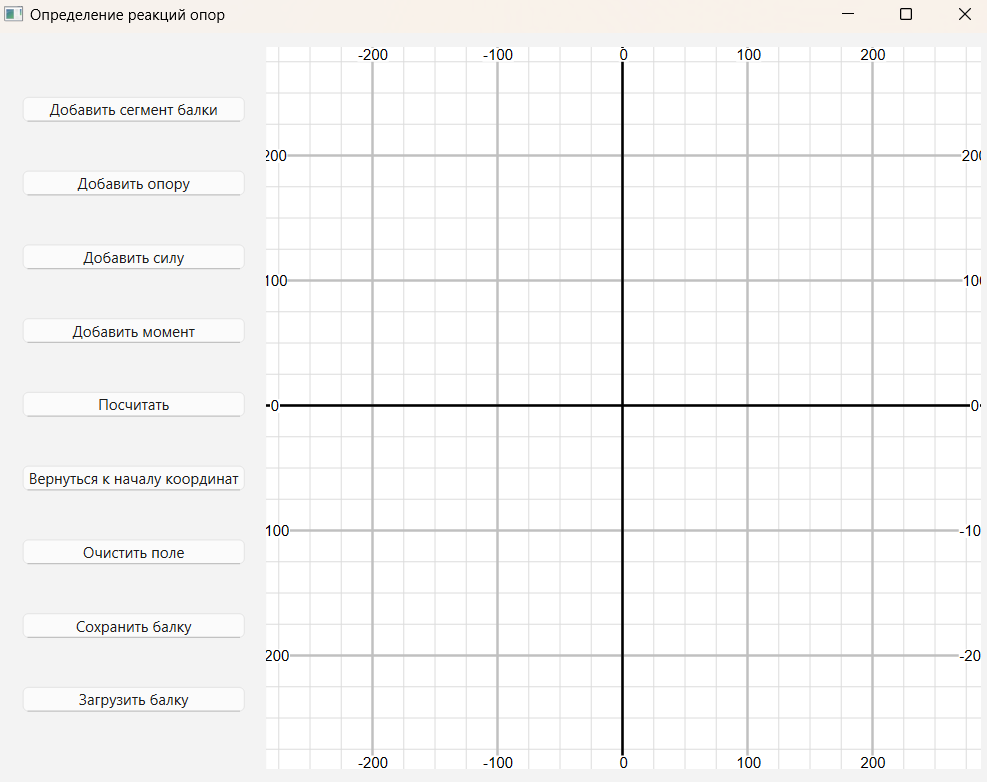


Рисунок 23 – Тест TEST\_UI\_004 (изменённый масштаб)

Резюме: Тест пройден

## Тест TEST\_UI\_005

Покрываемые классы эквивалентности: Нет отдельного класса (Тестирование интерфейса)

Тестируемые требования: REQ\_UI\_003

Описание теста: Проверка перемещения координатной плоскости с помощью зажатой левой кнопки мыши.

Вводимые данные:

Операция: Зажать левую кнопку мыши на пустом пространстве и перетащить координатную плоскость вниз и вправо.

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: Плоскость перемещается в направлении движения мыши. Элементы модели (балки, узлы, опоры) остаются на своих относительных местах, сетка корректно адаптируется.

Видимый результат:

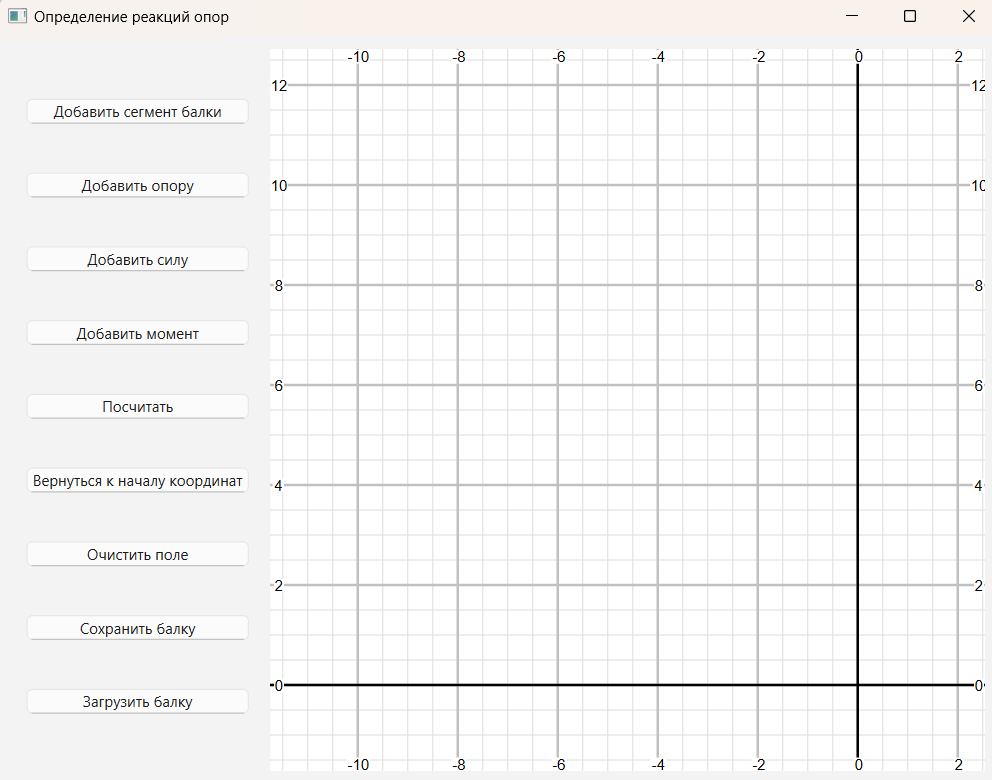


Рисунок 24 – Тест TEST\_UI\_005

Резюме: Тест пройден

## Тест TEST\_UI\_006

Покрываемые классы эквивалентности: (4)

Тестируемые требования: REQ\_UI\_004 (Добавление силы к сегменту балки)

Описание теста: Добавление сосредоточенной силы величиной 100 Н под углом 90° к середине сегмента балки.

Вводимые данные:

Номер сегмента: 1

Отступ от начального узла: 2 м

Значение силы: 100 Н

Угол приложения: 90 градусов

Распределение: Нет

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: На выбранном сегменте появляется стрелка, направленная вверх под углом 90°, с подписью «100 Н». Стрелка отображается на соответствующем расстоянии от начала сегмента.

Видимый результат:

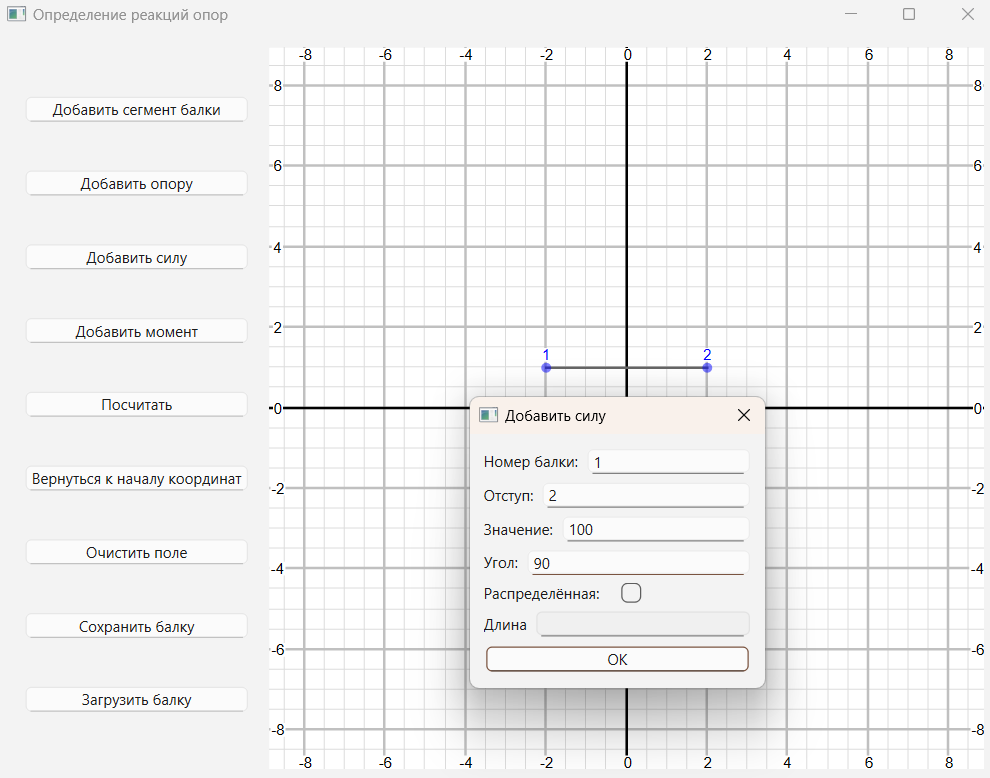


Рисунок 25 – Тест TEST\_UI\_006 (вводимые данные)

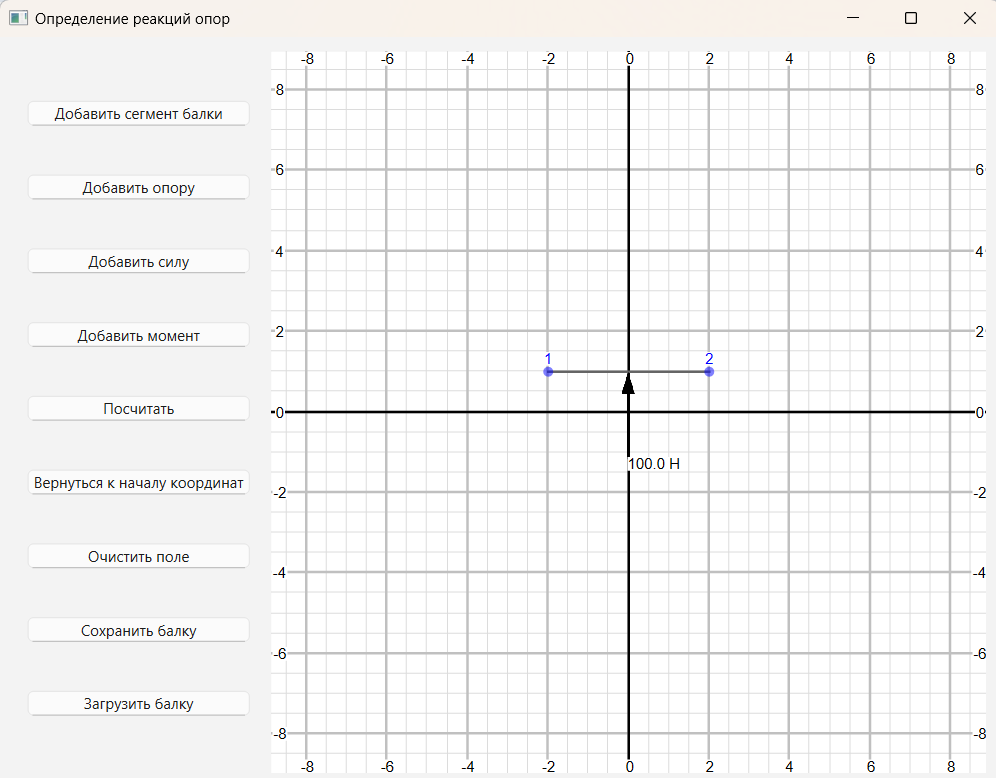


Рисунок 26 – Тест TEST\_UI\_006 (результат)

Резюме: Тест пройден

## Тест TEST\_UI\_007

Покрываемые классы эквивалентности: (5), (9)

Тестируемые требования: REQ\_UI\_004

Описание теста: Проверка обработки ошибки при добавлении силы с некорректным значением.

Вводимые данные:

Номер сегмента: 1

Отступ от начального узла: -5 м (отрицательный)

Значение силы: «сто» (не число)

Угол приложения: 45 градусов

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: При попытке добавления выводится всплывающее окно ошибки с текстом: «Ошибка: Введены некорректные данные» или «Отступ должен быть положительным числом». Операция отменяется.

Видимый результат:



Рисунок 27 – Тест TEST\_UI\_007 (вводимые данные)

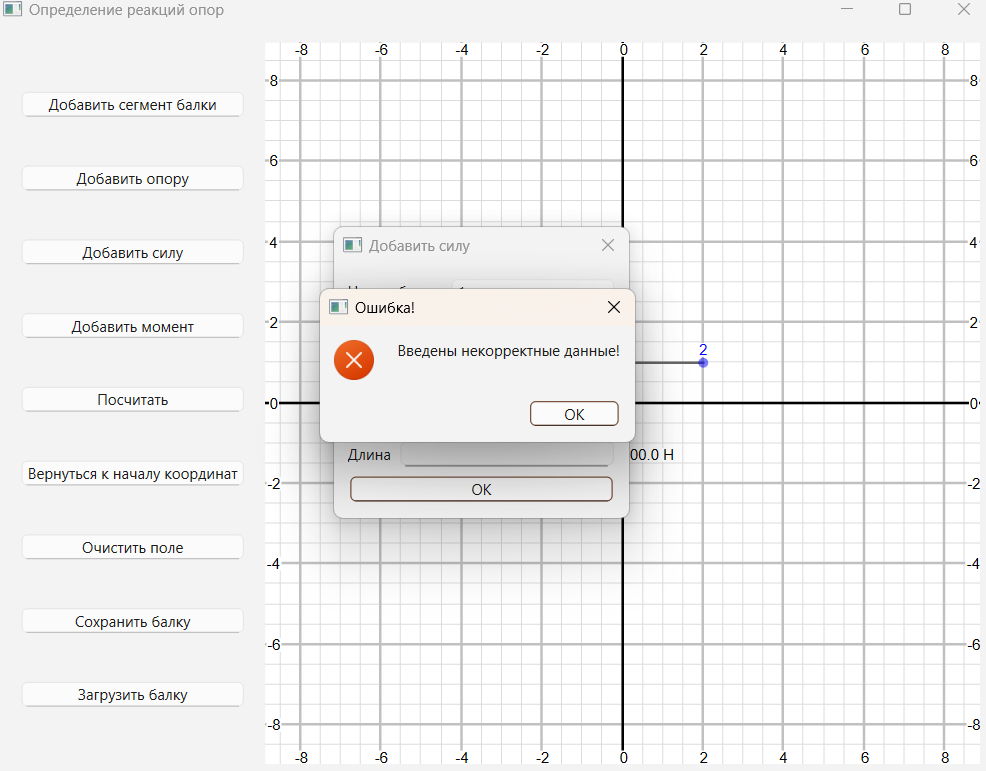


Рисунок 28 – Тест TEST\_UI\_007 (результат)

Резюме: Тест пройден

## Тест TEST\_UI\_008

Покрываемые классы эквивалентности: Нет отдельного класса (Тестирование интерфейса)

Тестируемые требования: REQ\_UI\_005 (Отображение подписей узлов и сегментов)

Описание теста: Проверка отображения номеров узлов и балок после построения конструкции.

Вводимые данные:

Создать сегмент между узлами (0,0) и (5,0).

Добавить еще один сегмент между (5,0) и (5,5).

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: На плоскости около каждого узла отображаются их уникальные номера (например, 1, 2, 3). Номера балок отображаются над или рядом с соответствующим сегментом.

Видимый результат:



Рисунок 29 – Тест TEST\_UI\_008

Резюме: Тест пройден

## Тест TEST\_UI\_009

Покрываемые классы эквивалентности: Нет отдельного класса (Тестирование интерфейса)

Тестируемые требования: REQ\_UI\_005

Описание теста: Проверка обновления подписей при масштабировании координатной плоскости.

Вводимые данные:

После построения балки прокрутить колесо мыши вверх (увеличение масштаба).

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: Номера узлов и балок остаются четко читаемыми, автоматически масштабируются пропорционально увеличению без наложения или смещения.

Видимый результат:

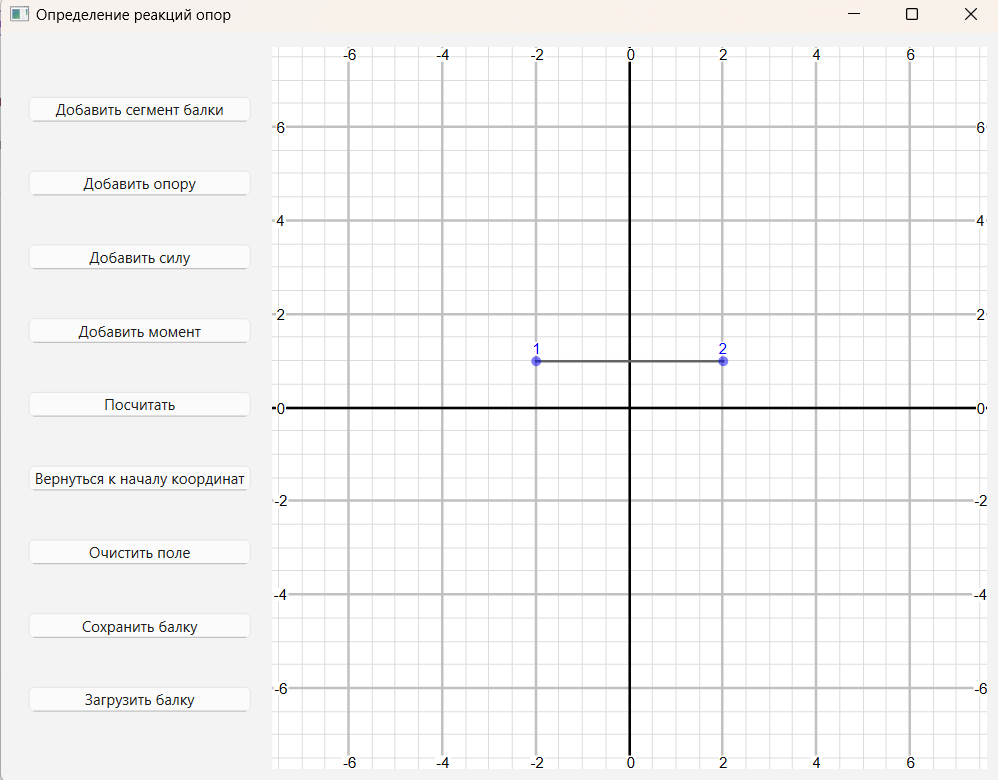


Рисунок 30 – Тест TEST\_UI\_009 (начальный масштаб)

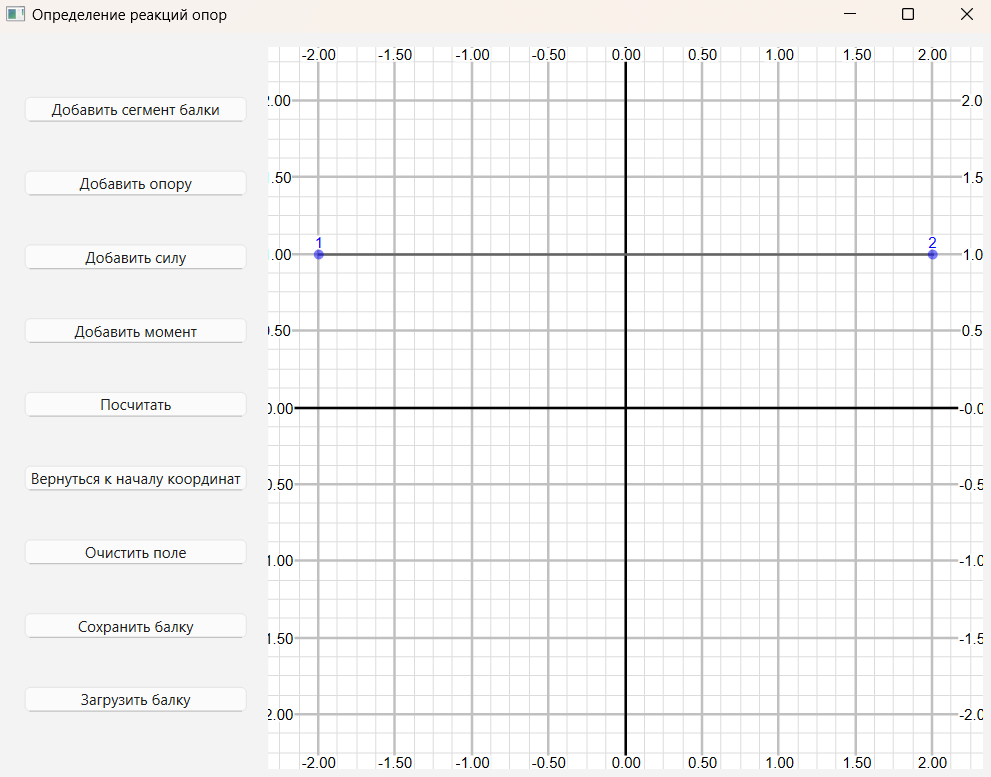


Рисунок 31 – Тест TEST\_UI\_009 (изменённый масштаб)

Резюме: Тест пройден

## Тест TEST\_UI\_010

Покрываемые классы эквивалентности: Нет отдельного класса (Тестирование интерфейса)

Тестируемые требования: REQ\_UI\_007 (Интерфейс должен быть адаптивным к разным размерам экранов)

Описание теста: Проверка адаптивности интерфейса на разных размерах окон.

Вводимые данные:

Операция: Изменить размер окна приложения:

Уменьшить размер окна до 800x600 пикселей.

Увеличить размер окна до 1920x1080 пикселей (Full HD).

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: Координатная плоскость и элементы конструкции (балки, узлы, силы, опоры) автоматически масштабируются и остаются полностью доступными без наложения или искажения. Все элементы интерфейса остаются работоспособными.

Видимый результат:

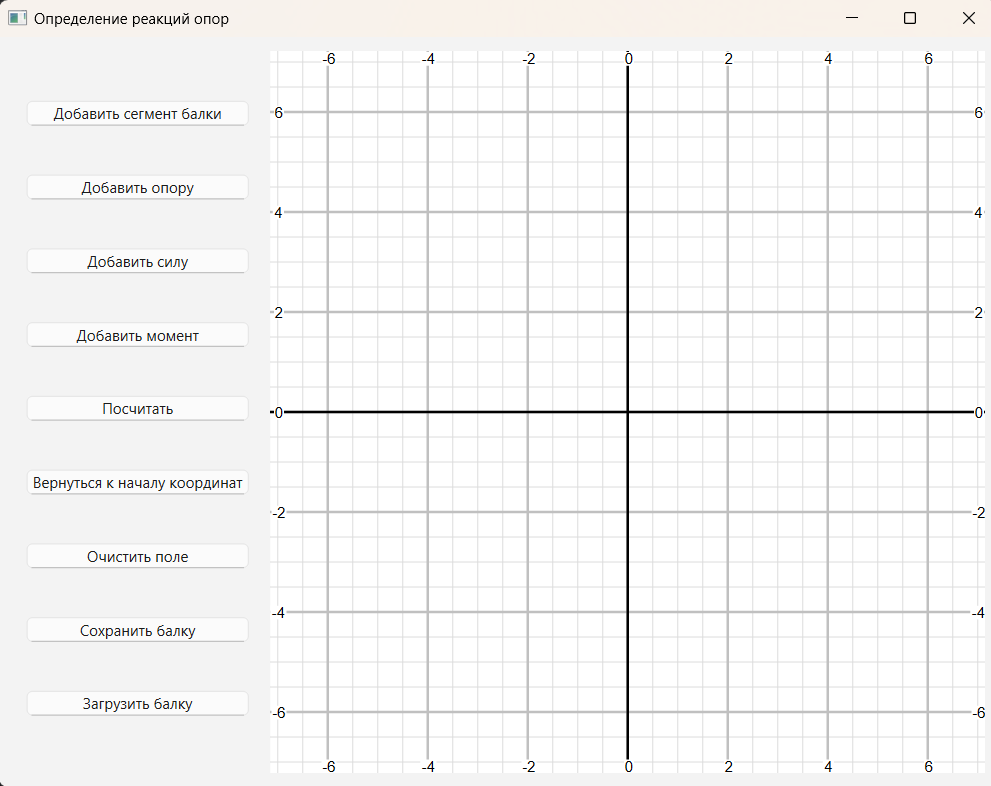


Рисунок 32 – Тест TEST\_UI\_010 (начальный масштаб)

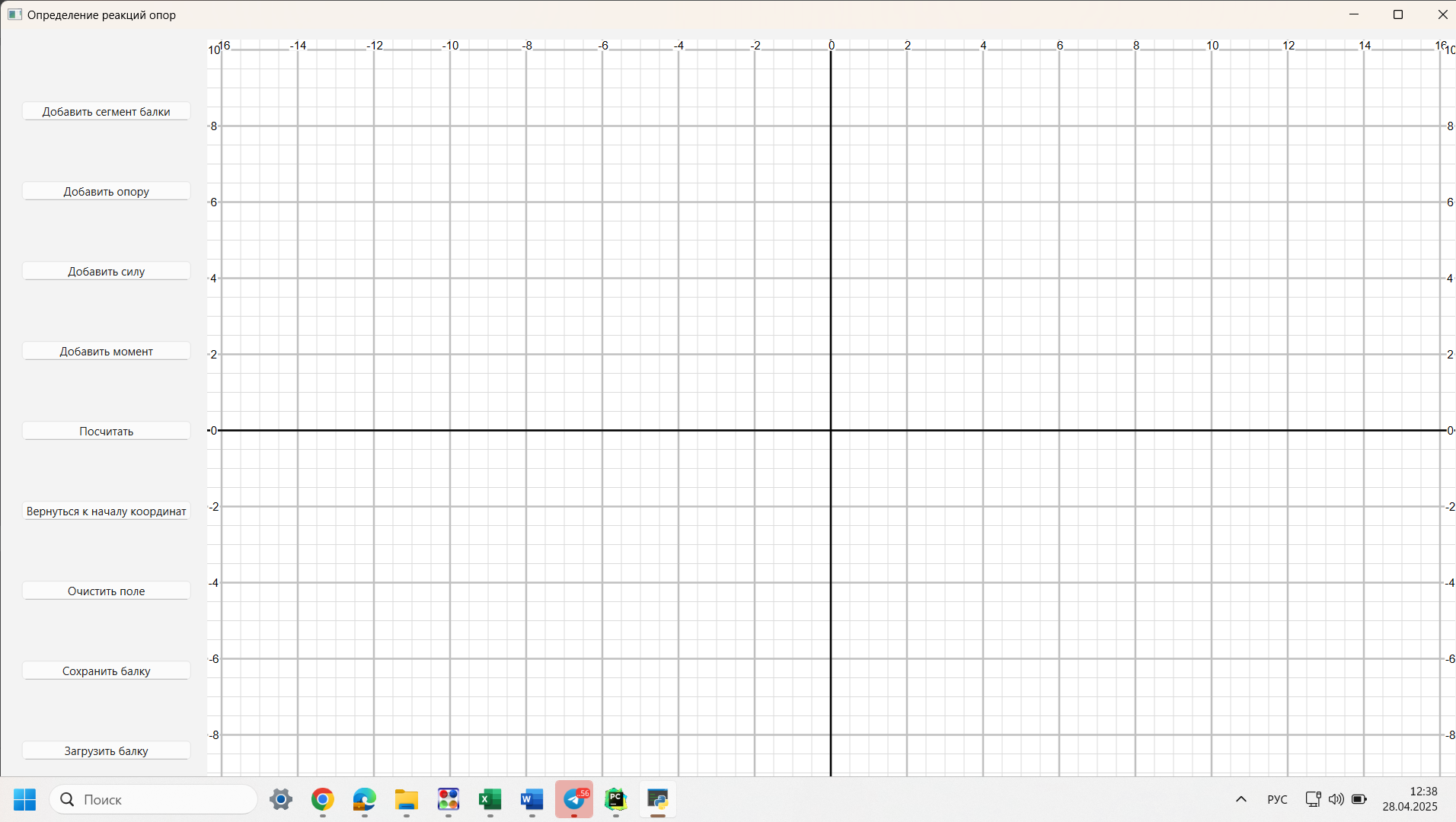


Рисунок 33 – Тест TEST\_UI\_010 (изменённый масштаб)

Резюме: Тест пройден

## Тесты для тестирования подсистемы «Модуль расчетов»

## Тест TEST\_CALC\_001

Покрываемые классы эквивалентности: (18)

Тестируемые требования: REQ\_CALC\_001 (Расчет реакций через уравнения равновесия)

Описание теста: Проверка расчета реакций опор для простой балки на двух опорах с одной сосредоточенной вертикальной силой посередине.

Вводимые данные:

Узел 1: (0,0), жесткая заделка.

Узел 2: (4,0), шарнирно-подвижная опора.

Сегмент между узлами 1 и 2.

Сила: 100 Н, приложена к середине балки, под углом 90°.

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: Модуль расчета успешно определяет реакции опор:

На жесткой заделке: горизонтальная и вертикальная реакции + момент.

На подвижной опоре: только вертикальная реакция. Сумма всех сил и моментов равна нулю (уравнения равновесия выполняются).

Видимый результат:

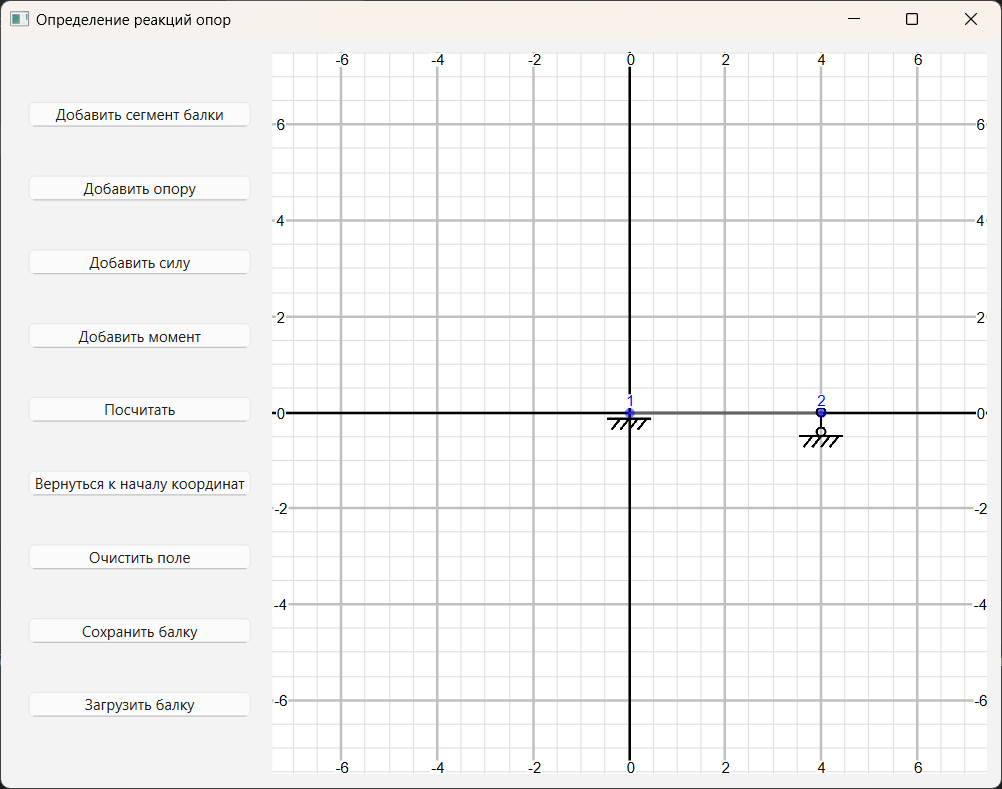


Рисунок 34 – Тест TEST\_CALC\_001

Резюме: Тест пройден

## Тест TEST\_CALC\_002

Покрываемые классы эквивалентности: (18)

Тестируемые требования: REQ\_CALC\_002 (Расчет сил)

Описание теста: Проверка корректного расчета распределения сил при наличии нескольких сосредоточенных сил на балке.

Вводимые данные:

Узел 1: (0,0), жесткая заделка.

Узел 2: (6,0), шарнирно-подвижная опора.

Сегмент между узлами 1 и 2.

Силы:

50 Н на 2 метрах от начала, угол -90°.

75 Н на 4 метрах от начала, угол -90°.

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: Модуль корректно рассчитывает реакции опор, учитывая обе приложенные силы, с учетом их положения и момента относительно базовой точки.

Видимый результат:

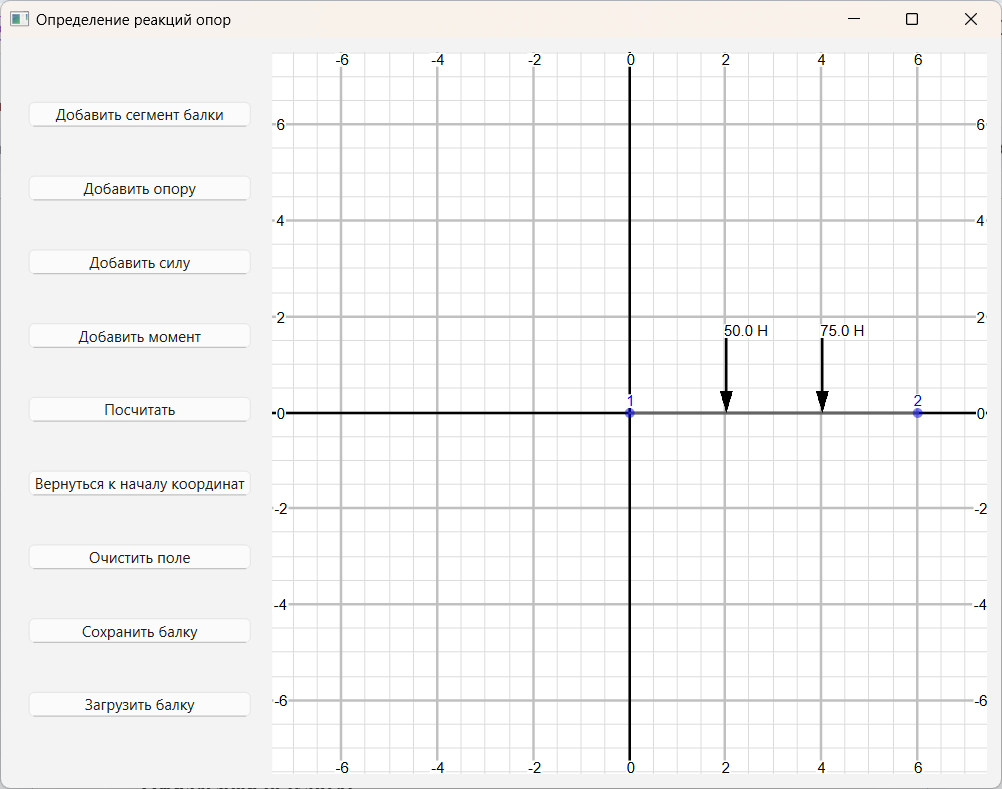


Рисунок 35 – Тест TEST\_CALC\_002

Резюме: Тест пройден

## Тест TEST\_CALC\_003

Покрываемые классы эквивалентности: (19)

Тестируемые требования: REQ\_CALC\_005 (Проверка введённых данных на корректность)

Описание теста: Попытка рассчитать реакции при создании балки без опор.

Вводимые данные:

Узел 1: (0,0)

Узел 2: (3,0)

Сегмент между узлами 1 и 2. (Опоры отсутствуют.)

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: При попытке расчета возникает ошибка: выводится сообщение об отсутствии опор. Процесс расчета останавливается, пользователь получает уведомление о некорректной постановке задачи.

Видимый результат:

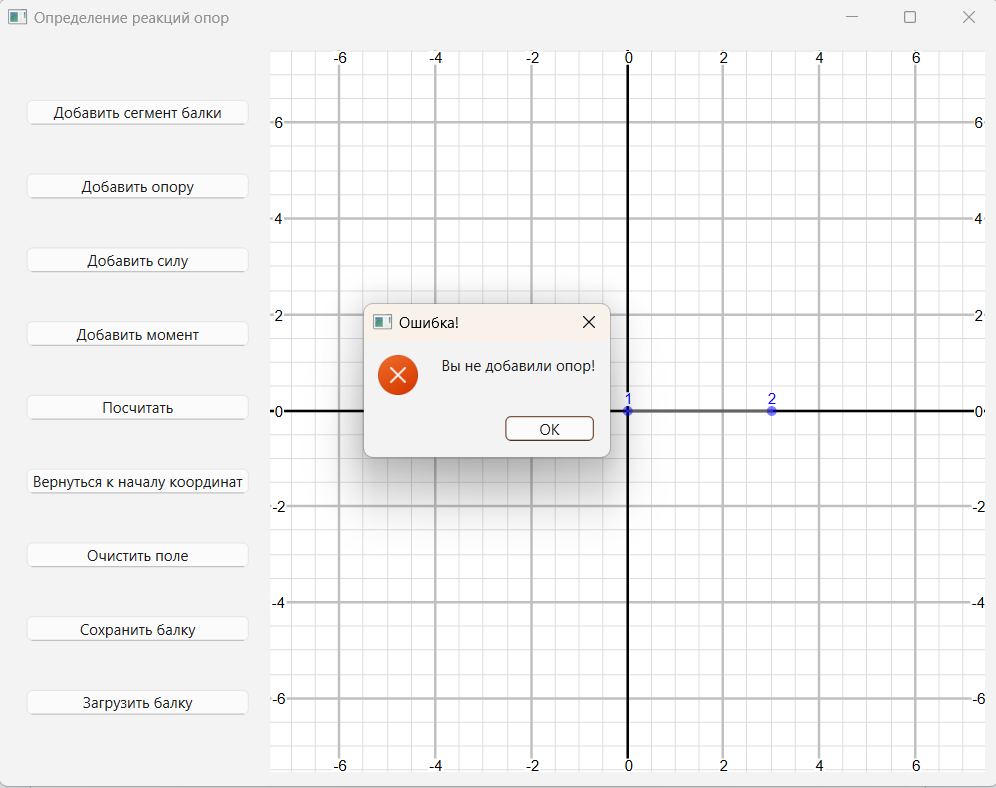


Рисунок 36 – Тест TEST\_CALC\_003

Резюме: Тест пройден

## Тест TEST\_CALC\_004

Покрываемые классы эквивалентности: (5)

Тестируемые требования: REQ\_CALC\_005

Описание теста: Попытка добавить силу с некорректным значением (например, отрицательная сила).

Вводимые данные:

Номер балки: 1

Отступ: 1 м

Значение силы: -100 Н

Угол приложения: 90 градусов

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: Появляется всплывающее окно ошибки: «Ошибка: Значение силы должно быть положительным числом». Операция добавления силы отменяется.

Видимый результат:

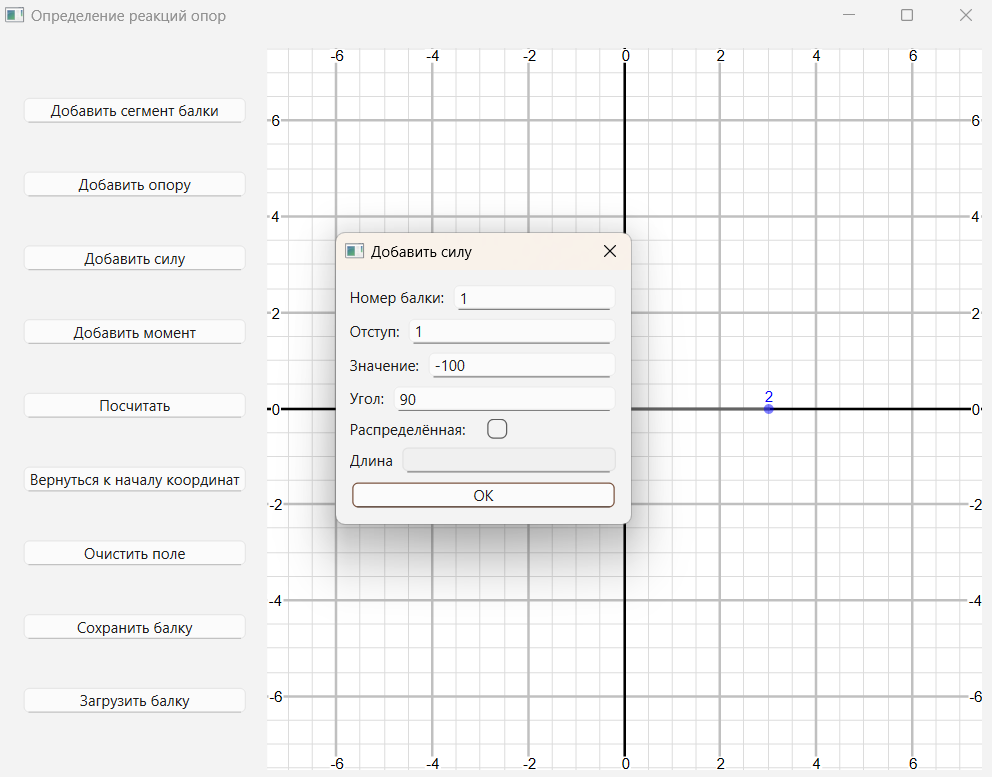


Рисунок 37 – Тест TEST\_CALC\_004 (вводимые данные)

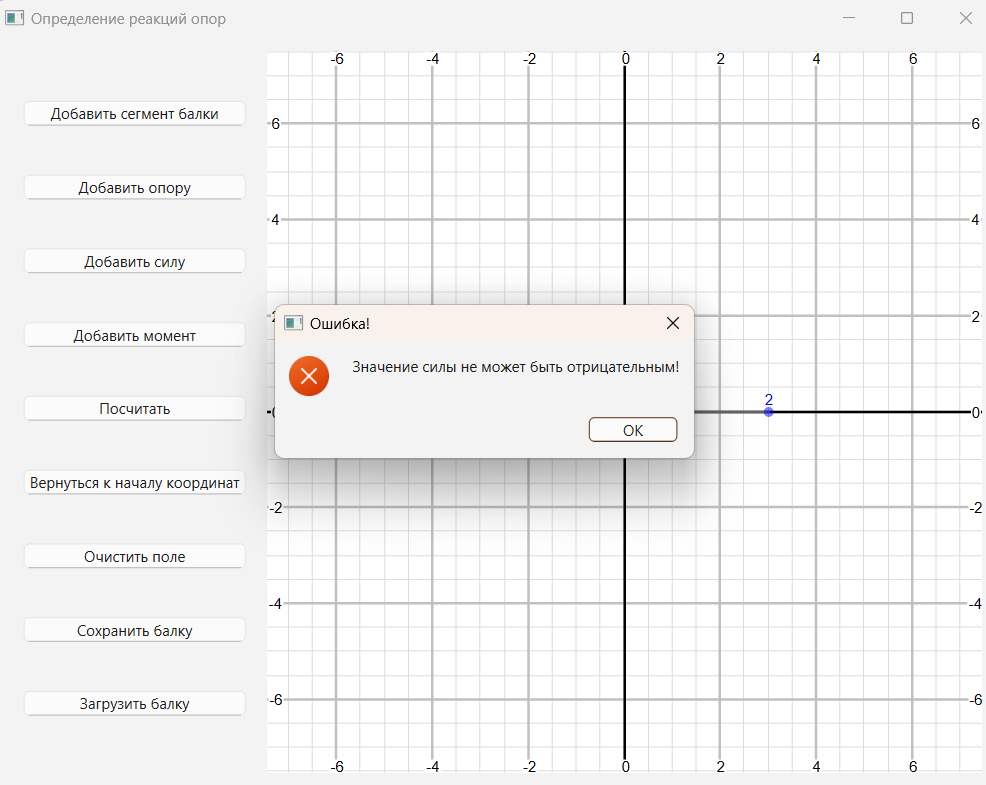


Рисунок 38 – Тест TEST\_CALC\_004 (результат)

Резюме: Тест пройден

## Тест TEST\_CALC\_005

Покрываемые классы эквивалентности: (19)

Тестируемые требования: REQ\_CALC\_005

Описание теста: Попытка рассчитать реакции на пустой конструкции (нет ни узлов, ни сегментов).

Вводимые данные:

Модель: пустая (без созданных элементов).

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: При попытке запустить расчет выводится сообщение об ошибке: «Ошибка: Конструкция не содержит элементов для расчета». Исключение обрабатывается корректно, программа не завершается аварийное.

Видимый результат:

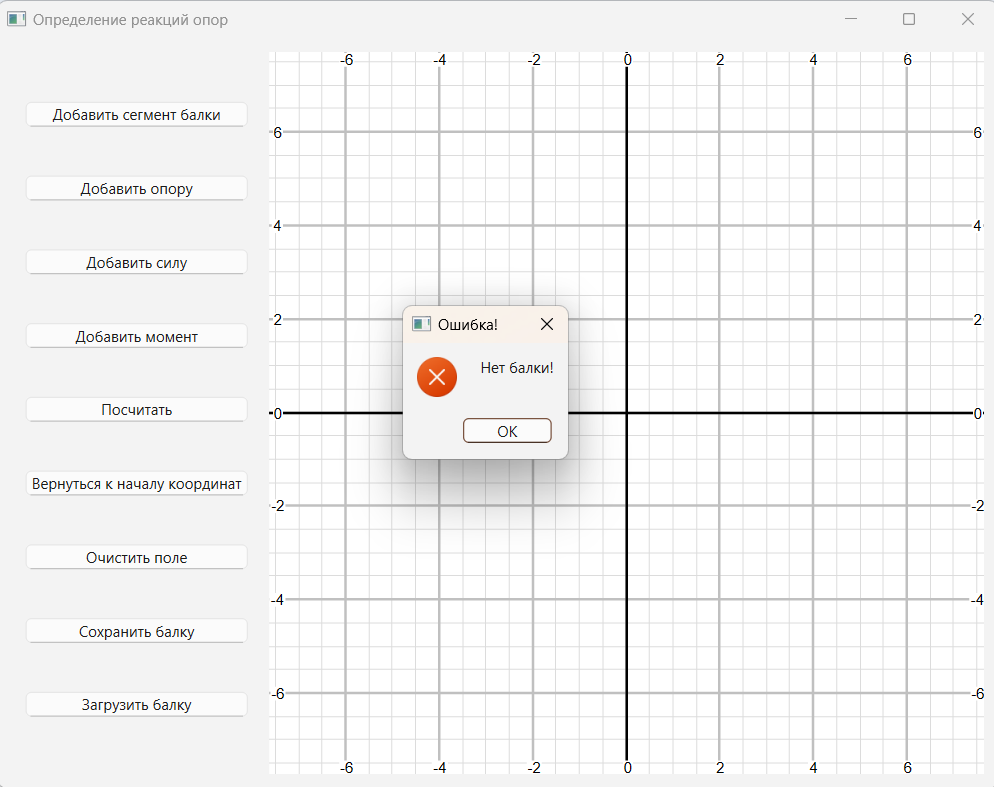


Рисунок 39 – Тест TEST\_CALC\_005

Резюме: Тест пройден

## Тест TEST\_CALC\_006

Покрываемые классы эквивалентности: Нет отдельного класса (Проверка точности)

Тестируемые требования: REQ\_CALC\_008 (Точность расчета до двух знаков после запятой)

Описание теста: Проверка точности вычисления реакций опор.

Вводимые данные:

Узел 1: (0,0), жесткая заделка.

Узел 2: (4,0), шарнирно-подвижная опора.

Сегмент между ними.

Сила: 33.333 Н вниз на 2 м от начала.

Тестируемая версия продукта: номер версии из GitHub

Ожидаемый результат: Все расчётные значения реакций на опорах выводятся с точностью до двух знаков после запятой (например, 66.66 Н, 33.33 Н), без лишних знаков.

# Заключение

В ходе выполнения курсовой работы был разработан проект «Конструктор задач по определению реакции опор», который охватывает весь цикл коллективной разработки, от планирования до тестирования. В рамках работы были решены ключевые задачи, такие как создание детализированного плана проекта, разработка архитектуры системы, а также составление требований и рекомендаций по кодированию. Также особое внимание было уделено разработке моделей состояний задач и проведению тестирования, что позволило удостовериться в функциональности и эффективности работы проекта.

Особо важным моментом работы является использование подходов коллективной разработки, что способствует улучшению координации и обмена знаниями между участниками команды, а также повышает качество конечного продукта. Успешная реализация всех поставленных задач позволила создать проект, который отвечает требованиям промышленной разработки информационных систем и может быть использован в качестве примера для дальнейших работ в данной области.

# Список литературы

1. Учебно-методический комплекс дисциплины «Технология коллективной разработки информационных систем» [Электронный ресурс]. URL: <https://bb.dvfu.ru/webapps/blackboard/content/listContent.jsp?course_id=_5025_1&content_id=_172365_1> (дата обращения: 21.01.2025)