**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

**ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ (ШКОЛА)**

**Департамент программной инженерии и искусственного интеллекта**

РАЗРАБОТКА ТРАНСЛЯТОРА С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИЙ КОЛЛЕКТИВНОЙ ПРОМЫШЛЕННОЙ РАЗРАБОТКИ

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Технологии коллективной промышленной разработки информационных систем»

по образовательной программе подготовки бакалавров по направлению

09.03.04 «Программная инженерия», профиль «Программная инженерия»

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |  |  | Студенты группы Б9121-09.03.04 | | | | | | | |
|  |  |  | | | | Княжев А. К. | | | |
|  | | | | | | |  |  | (подпись) | | | |  | | | |
|  | | | | | | |  |  |  | | | | Княжева В. Л. | | | |
|  | | | | | | |  |  | (подпись) | | | |  | | | |
|  | | | | | | |  |  |  | | | | Барбаянов М. А. | | | |
|  | | | | | | |  |  | (подпись) | | | |  | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель: | | | | | | | |
|  | | | | | | |  |  | Старший преподаватель ДПиИИ | | | | | | | |
|  | | | | | | |  |  |  | | | | |  | Иваненко Ю. С. | |
|  | | | | | | |  |  | (подпись) | | | | |  | (ФИО) | |
|  | | | | | | | | | | | | | | | | |
| Регистрационный № | | | | | | |  |  | Защищен с оценкой | | | | | | | |
|  | | | | | | |  |  |  | | | | | | | |
|  | | | |  |  | |  |  | « |  | » |  |  | | | 2025 г. |
| (подпись) | | | |  | (ФИО) | |  |  |  | | | | | | | |
| « |  | » |  | | | 2025 г. |  |  |  | | | | | | | |

г. Владивосток

2025

Оглавление

[Введение 3](#_Toc197294929)

[1 Разработка плана проекта 5](#_Toc197294930)

[2 Разработка регламента проведения инспекции 8](#_Toc197294931)

[3 Разработка модели состояний задач 13](#_Toc197294932)

[4 Разработка презентации проекта 16](#_Toc197294936)

[5 Разработка требований к проекту 23](#_Toc197294937)

[6 Разработка архитектуры проекта 28](#_Toc197294946)

[7 Разработка измерений проекта 32](#_Toc197294947)

[8 Разработка перечня задач проекта 34](#_Toc197294950)

[9 Разработка рекомендаций по кодированию 35](#_Toc197294953)

[10 Разработка плана тестирования проекта 37](#_Toc197294954)

[11 Тестирование проекта 42](#_Toc197294957)

[Заключение 46](#_Toc197294958)

[Список литературы 48](#_Toc197294959)

# Введение

В современном мире компиляторы играют большую роль в программировании. Причём, это не только компиляторы языков высокого уровня в языки более низкого уровня, но и компиляторы из одного высокоуровневого языка в другой.

Компилятор – программа или техническое средство, выполняющее компиляцию.

Компиляция – трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком к машинному коду. Входной информацией для компилятора является описание алгоритма или программа на проблемно-ориентированном языке, а на выходе компилятора – эквивалентное описание алгоритма на машинно-ориентированном языке.

Трансляция – преобразование программы, представленной на одном из языков программирования, в программу на другом языке. Транслятор обычно выполняет также диагностику ошибок, формирует словари идентификаторов, выдаёт для печати текст программы и т. д.

Промышленная разработка информационных систем включает в себя множество этапов, начиная от разработки плана проекта и заканчивая его тестированием. Для этого, очевидно, требуется участие специалистов различных профилей, а также использование унифицированных методов коммуникации между ними, что позволяет эффективно распределять обязанности в команде в соответствии с профессиональной специализацией каждого участника.

Исходя из изложенного, возникает необходимость в использовании технологий коллективной разработки для повышения эффективности работы команды и обеспечения соответствия программного продукта заявленным требованиям.

В данной курсовой работе рассматривается задача разработки транслятора, переводящего подмножество языка Python в эквивалентное подмножество языка C++, и составление технической документации к данному средству.

Для достижения поставленной цели необходимо решить следующие задачи:

* Разработать план проекта;
* Разработать регламент проведения инспекции;
* Разработать модель состояний задач;
* Разработать презентацию проекта;
* Разработать требования к проекту;
* Разработать архитектуру проекта;
* Разработать измерения проекта;
* Разработать перечь задач проекта;
* Разработать рекомендации по кодированию;
* Разработать план тестирования проекта;
* Протестировать проект.

# Разработка плана проекта

План проекта — это документ, содержащий подробную информацию о проекте: исполнителях, задачах и сроках, помогающий организовать все процессы, нацеленные на конечный результат проекта. Документ является конечным результатом этапа планирования, утверждается до начала любых работ и становится самым главным и достоверным источником информации о грядущем проекте.

План включает в себя следующие главы:

1. Разработка плана проекта.

Описание ключевых этапов выполнения, сроков, необходимых ресурсов и задач, обеспечивающих успешную реализацию проекта.

1. Разработка регламента проведения инспекций

Установление процедур и правил для оценки и контроля качества выполнения работ на всех этапах проекта.

1. Разработка модели состояний задач.

Построение схемы, отображающей жизненный цикл задач с указанием их состояний, возможных переходов и правил обработки.

1. Разработка презентации проекта.

Подготовка текстовых и визуальных материалов, представляющих цели, структуру и ключевые особенности проекта для заинтересованных сторон.

1. Разработка требований к проекту.

Сбор, анализ и формализация функциональных и нефункциональных требований к продукту или системе.

1. Разработка архитектуры проекта.

Создание структуры системы или продукта, определение ключевых компонентов и их взаимодействий.

1. Разработка измерений проекта.

Определение и описание метрик для оценки прогресса, качества и эффективности проекта.

1. Разработка перечня задач проекта.

Подробное разбиение проекта на задачи с указанием приоритетов, сроков и ответственных.

1. Разработка рекомендаций по кодированию.

Создание стандартов и правил, направленных на улучшение качества и обеспечение удобства сопровождения программного кода.

1. Разработка плана тестирования проекта.

Определение подходов, методов и этапов тестирования для проверки соответствия проекта требованиям.

1. Тестирование проекта.

Практическая проверка работоспособности, функциональности и качества проекта на основе разработанного плана.

В нашем случае исполнителями являются следующие лица:

* Team Leader/Coder — Княжев А. К.
* Build Engineer/Coder — Княжева В. Л.
* Technical Writer/Tester — Барбаянов М.А.

Для перечня задач выше был разработан план с примерными сроками реализации каждого этапа.

Для реализации проекта была выбрана водопадная модель разработки, так как она предполагает последовательное выполнение этапов, что соответствует структуре работы над транслятором. Такой подход позволяет чётко определить границы между этапами — от планирования и проектирования до реализации и тестирования — и упростить контроль за выполнением задач.

Был разработан перечень задач для выполнения и примерные сроки их реализации (Рисунок 1).



Рисунок 1 — План проекта

# Разработка регламента проведения инспекции

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 (peer reviews).

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:

* Выявление ошибок, связанных с функциональностью, логикой, содержанием или реализацией продукта на ранних стадиях, чтобы избежать их дальнейшего распространения и наследования
* Рационально донести замысел или реализацию продукта до всех заинтересованных лиц (через их участие);
* Обеспечение понимания концепции или реализации продукта всеми заинтересованными участниками проекта (через их участие);
* Повышение качества, оптимизация или оценка текущего состояния продукта.

#### 2.1 Критерии формальности инспекции

**Неформальная инспекция** проводится:

* В случае изменения участка документа, содержащего не более 5 строк, для текстовых документов;
* В случае изменения не более 5 элементов для документов дизайна.

**Формальная инспекция** проводится в случае:

* Необходимости проверки изменений в сложных или критически важных участках продукта;
* Если затронуты ключевые компоненты или логика системы;
* В случае, если изменения требуют обсуждения с широким кругом участников (например, с представителями заказчика).

#### 2.2 Участники инспекции

Участники могут иметь следующие роли:

1. **Автор** — участник, внёсший изменения в рабочий продукт. Обязанности:
   * Загружает изменения в систему контроля версий;
   * Инициирует инспекцию;
   * Вносит исправления по замечаниям инспектора.
2. **Председатель** — лидер команды (Team Leader), контролирующий процесс инспекции. Обязанности:
   * Назначает инспектора;
   * Одобряет внесение ключевых изменений в основную версию продукта;
   * Участвует в обсуждении сложных вопросов, если инспектор обращается за помощью.
3. **Инспектор** — участник, проводящий проверку внесённых изменений. Обязанности:
   * Анализирует изменения;
   * Оставляет замечания;
   * Выносит вердикт о внесении изменений в продукт.

Дополнительно отмечается целесообразность включения в состав инспекции **секретаря** — участника, ответственного за фиксацию обсуждений, замечаний и принятых решений. Это особенно важно при формальных инспекциях, где требуется точная документация результатов для последующего анализа и контроля устранения выявленных недостатков.

#### 2.3 Этапы проведения инспекции

1. **Инициация:**
   * Автор создаёт запрос на внесение изменений в продукт;
   * Назначается инспектор и, при необходимости, председатель.
2. **Подготовка материалов:**
   * Автор подготавливает сопроводительную документацию к изменениям, включая описание целей изменений, затронутые модули, сценарии тестирования;
   * Инспектор изучает предоставленные материалы.
3. **Проведение инспекции:**
   * Инспектор проводит анализ изменений и оставляет замечания;
   * При необходимости проводится обсуждение с участием председателя.
4. **Доработка:**
   * Автор вносит исправления по замечаниям инспектора.
5. **Завершение:**
   * Инспектор проверяет исправления и выносит окончательный вердикт.

#### 2.4 Порядок организации инспекции

Работа над продуктом ведётся в системе контроля версий GIT. Автор изменений:

* Оформляет Pull Request (Merge Request);
* Назначает инспектора;
* Уведомляет инспектора в рабочем чате (например, ВК).

Если изменения касаются дизайна проекта, инспектор:

* Обращается к председателю инспекции через рабочий чат;
* После завершения работы одобряет Pull Request или отправляет на доработку автору, уведомляя его.

#### 2.5 Порядок подготовки и проведения инспекции

Инспекция должна быть проведена в течение 7 дней с момента её инициации. Этапы:

1. Автор загружает изменения и создаёт Pull Request.
2. Инспектор анализирует изменения, оставляет замечания в системе контроля версий, обозначая степень их важности.
3. При наличии замечаний работа передаётся на доработку автору.
4. При отсутствии замечаний инспекция считается завершённой, и изменения вступают в силу.

#### 2.6 Перечень статусов и степени важности замечаний

1. **Комментарий** — это предложение по улучшению качества продукта, но его реализация остаётся на усмотрение команды и не является обязательной.
2. **Ошибка** — это выявленный недостаток, который требует обязательного устранения для соответствия продукту требованиям или стандартам.

#### 2.7 Порядок верификации учёта замечаний

После внесения исправлений инспектор:

* Проверяет учёт всех замечаний;
* При необходимости инициирует новую инспекцию или завершает текущую;
* Вносит итоговый вердикт в систему контроля версий.

#### 2.8 Метрики, характеризующие эффективность инспекций

**Inspection Rate (IR):**

Inspection Rate (IR) — метрика, отражающая скорость проведения инспекции и характеризующая, сколько единиц проверяемого продукта проходит инспекцию за единицу времени. Она используется для оценки эффективности процесса анализа артефактов разработки и выявления потенциальных узких мест в процессе инспектирования.

* Формула:

(1)

* Стратегическая цель: повысить качество разрабатываемого ПО.
* Единица измерения: <страница, требование, LOC, тест>/час.

# Разработка модели состояний задач

Каждая задача, являясь отражением делового процесса, проходит определенные состояния. Сначала идет создание задачи, потом идет выполнение работ по задаче, после выполнения задача завершается.

## Перечень возможных состояний задач и их интерпретация

* New (Новая) — задача создана, но еще не рассмотрена комитетом по контролю изменений.
* Approved (Утверждена) — задача принята к работе и назначена ответственному исполнителю.
* In Progress (В работе) — задача в процессе исполнения.
* Blocked (Заблокирована) — выполнение задачи временно остановлено в ожидании решения проблем или дополнительных ресурсов.
* Ready for Review (Готова к проверке) — задача завершена и готова для проверки комитетом или инспектором.
* Testing (Тестирование) — задача проходит проверку на соответствие требованиям и отсутствию ошибок.
* Done (Завершена) — задача успешно проверена и сдана.

## Правила создания новой задачи

1. Кто создает:

* Новые задачи создаются членами команды, выявившими проблему, либо инициированы руководителем проекта (PM).

1. Когда создает:

* Обнаружение дефекта в программном обеспечении.
* Новое функциональное требование.
* Улучшение существующего функционала.
* Задачи, направленные на оптимизацию.

1. Требования к созданию задачи:

* Указание четкого названия задачи.
* Описание проблемы или цели.
* Приложение доказательной базы (скриншоты, отчеты, ссылки и пр.).
* Указание предполагаемой категории задачи (например: баг, новая функция).

## Правила перехода задачи из состояния в состояние

* Переходы между состояниями фиксируются в системе управления проектами и задачами ClickUp
* Ответственные лица обязаны уведомлять заинтересованных сотрудников о смене состояния.

Детализация переходов

1. **New** ➔ **Approved**

Условия:

* Задача рассмотрена комитетом по контролю изменений.
* Назначен ответственный исполнитель.

Ответственный за перевод: Руководитель проекта (PM) или член комитета по контролю изменений.

1. **Approved** ➔ **In Progress**

Условия:

* Ответственный исполнитель приступил к выполнению задачи.
* Ответственный за перевод: Назначенный исполнитель.

1. **In Progress** ➔ **Blocked**

Условия:

* Возникли проблемы, требующие дополнительных ресурсов или решений.
* Исполнитель указал причину блокировки.

Ответственный за перевод: Исполнитель задачи.

Примечание: уведомить PM о блокировке.

1. **Blocked** ➔ **In Progress**

Условия:

* Проблемы решены.
* Исполнитель может продолжить выполнение задачи.

Ответственный за перевод: Исполнитель задачи.

1. **In Progress** ➔ **Ready for Review**

Условия:

* Задача выполнена в полном объеме.
* Задача соответствует первоначальным требованиям.

Ответственный за перевод: Исполнитель задачи.

1. **Ready for Review** ➔ **Testing**

Условия:

* Задача одобрена для тестирования.
* Ответственный за перевод: Комитет по контролю изменений или инспектор.

1. **Testing** ➔ **Done**

Условия:

* Задача успешно прошла тестирование.
* Все обнаруженные ошибки устранены.

Ответственный за перевод: Тестировщик или инспектор.

# Разработка презентации проекта

Была разработана презентация проекта, состоящая из двенадцати слайдов:

1. Титульный слайд (Рисунок 2)
2. Цель проекта и обоснование его актуальности (Рисунок 3)
3. Разработка плана проекта (Рисунок 4)
4. Разработка плана проведения инспекции (Рисунок 5)
5. Требования к транслятору (Рисунок 6)
6. Архитектура проекта (Рисунок 7)
7. Устройство системы (Рисунок 8)
8. Возможности и ограничения (Рисунок 9)
9. Измерения и метрики (Рисунок 10)
10. Тестирование (Рисунок 11)
11. Сравнение с аналогами (Рисунок 12)
12. Финальный слайд (Рисунок 13)

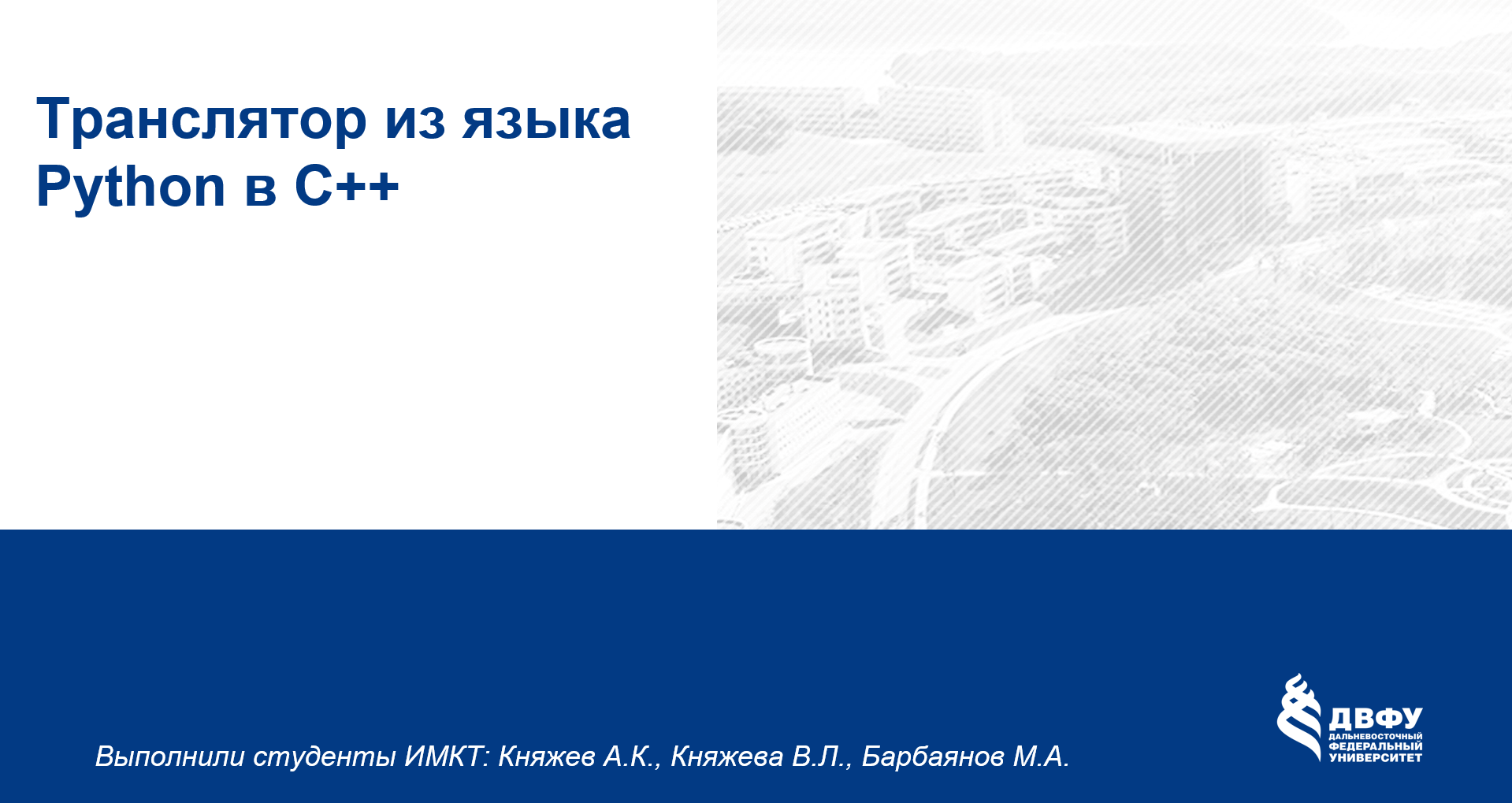


Рисунок 2 — Титульный слайд

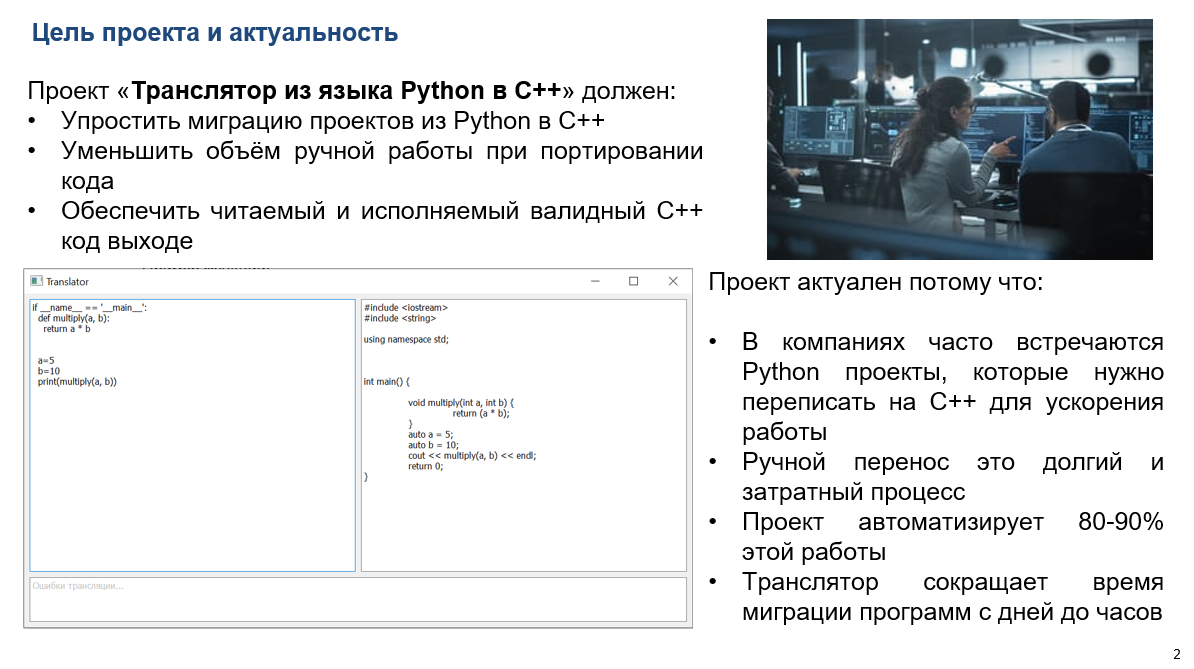


Рисунок 3 — Зачем он нужен?

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4 — Разработка плана проекта

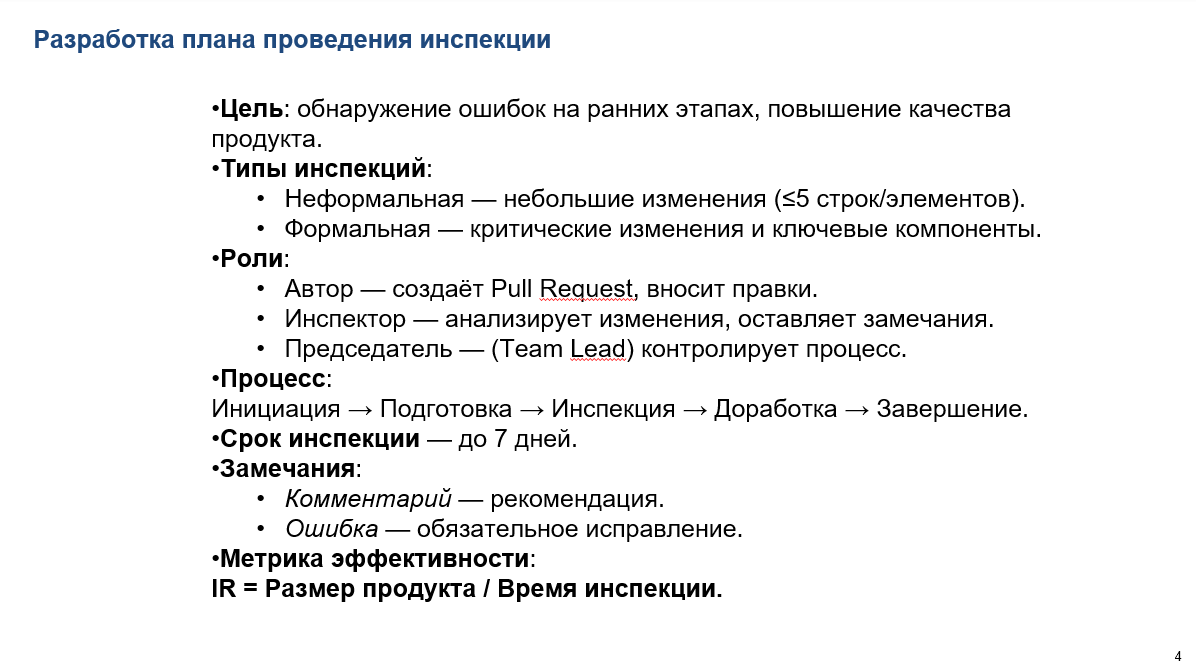


Рисунок 5 — Разработка плана проведения инспекции

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6 — Требования к транслятору

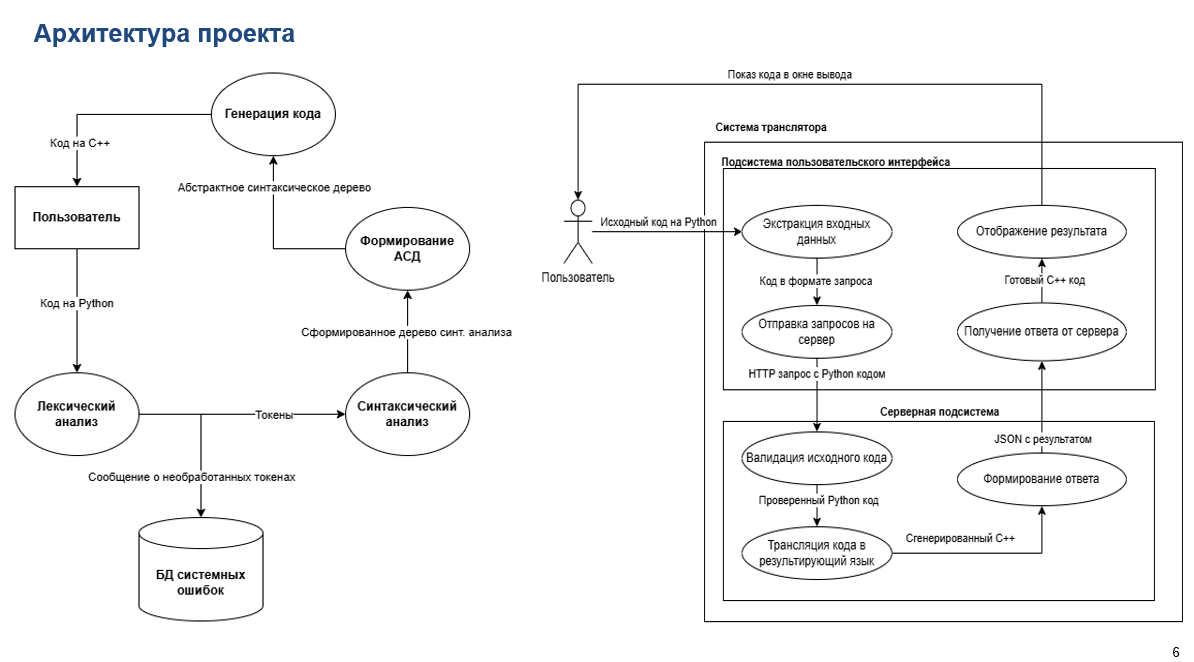


Рисунок 7 — Архитектура проекта

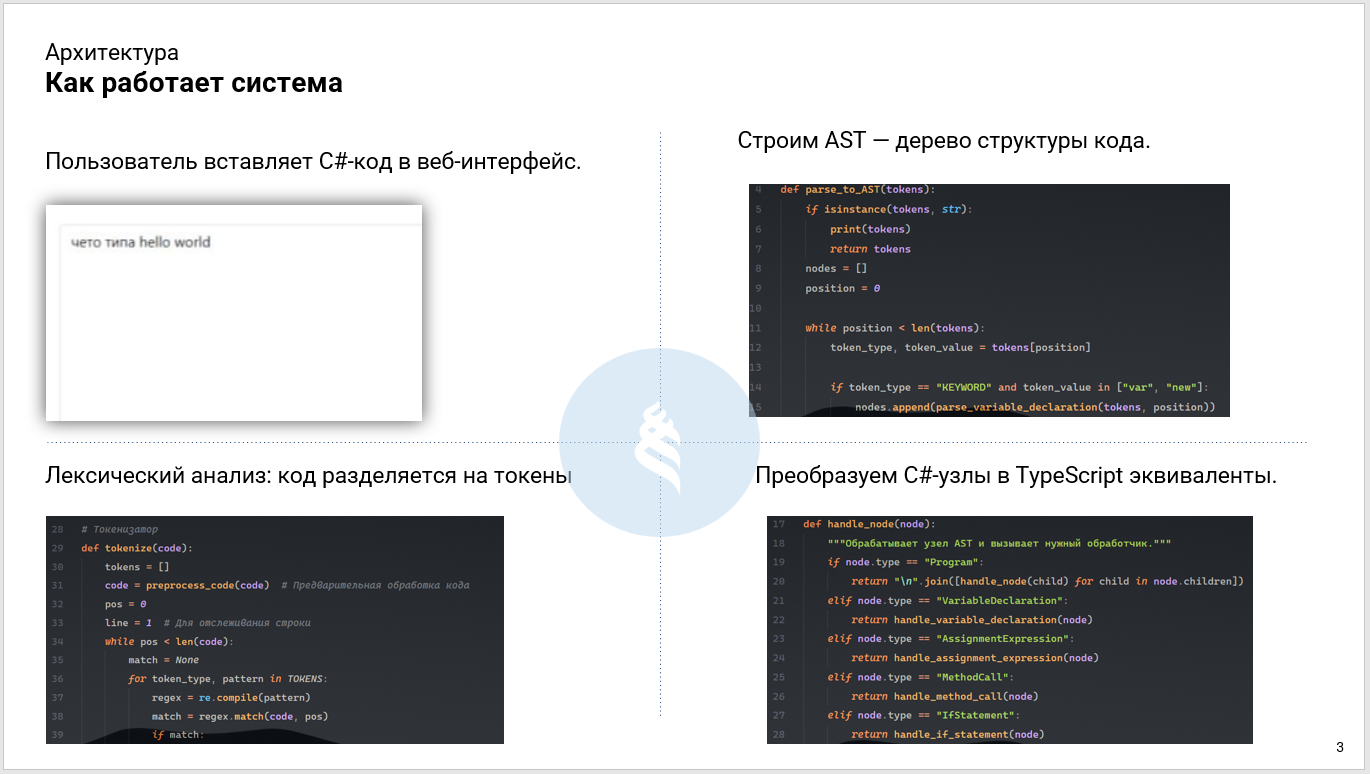


Рисунок 8 — Как работает система?

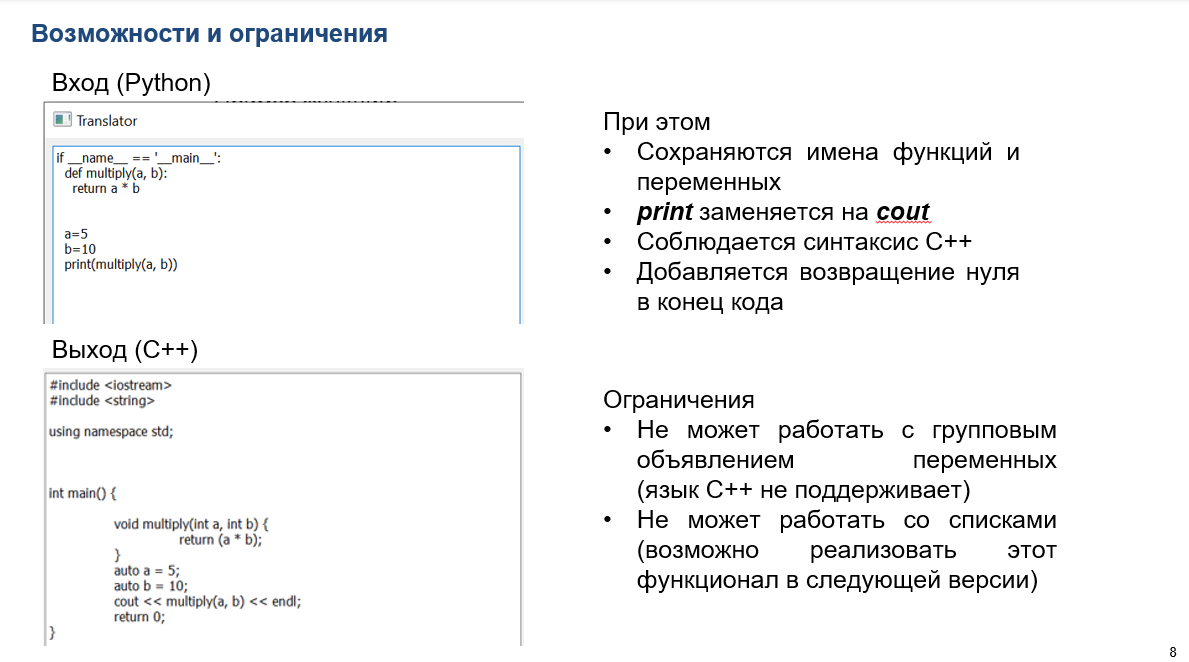


Рисунок 9 — Возможности и ограничения

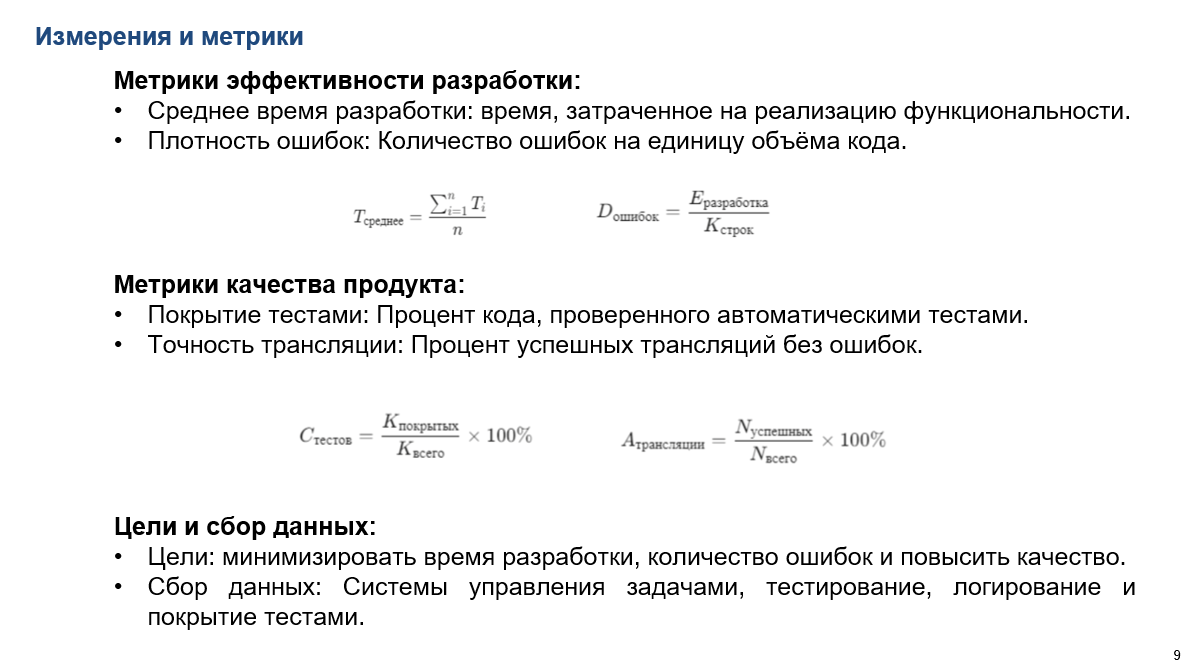


Рисунок 10 — Измерения и метрики

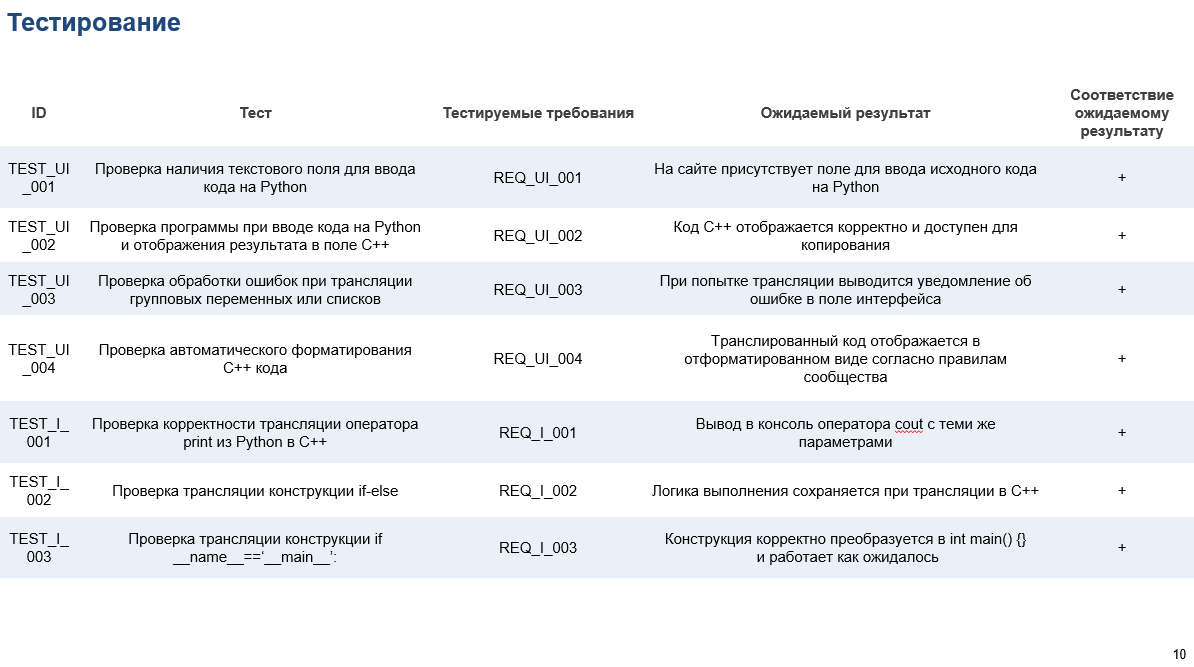


Рисунок 11 — Основные тест-кейсы



Рисунок 12 — Чем лучше аналогов?

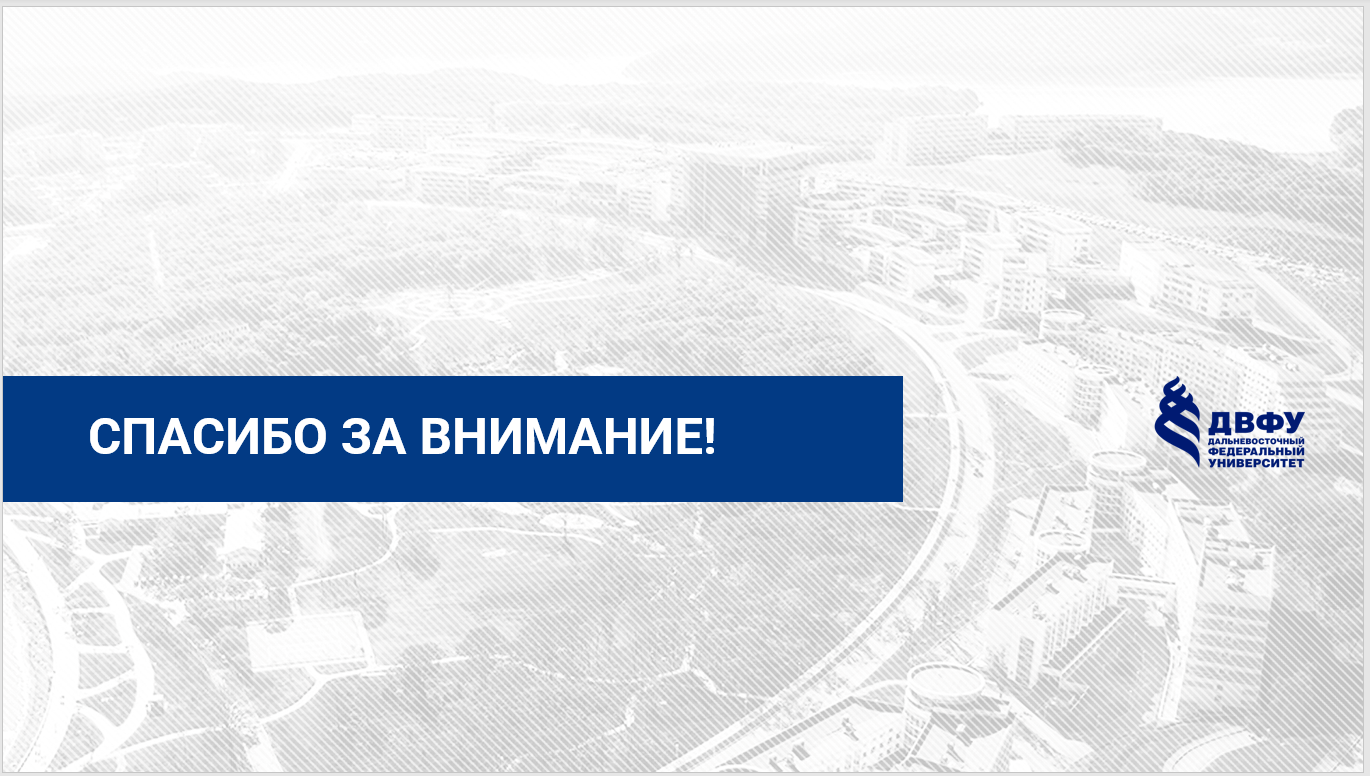


Рисунок 13 — Спасибо за внимание

# Разработка требований к проекту

Программный продукт: транслятор, переводящий подмножество языка Python в эквивалентное подмножество языка C++.

Программный продукт предназначен для автоматического перевода, поданного на вход пользователем кода на языке программирования Python в код на языке C++.

Программный продукт состоит из следующих подсистем:

1. Пользовательский интерфейс
2. Трансляция кода

Система представляет из себя десктопное приложение.

## Функциональные требования

Функциональные требования определяют функцию, которую должна выполнять система или системный компонент, чтобы удовлетворить потребности или ожидания пользователей.

Вот список функций, из которых состоит система и требований к ним:

1. FE-1 – Ручной ввод исходного кода на ЯП Python:

Система должна предоставлять возможность ввода исходного кода на языке Python через редактируемое текстовое поле.

1. FE-2 – Трансляция исходного кода в ЯП C++:

Система должна корректно преобразовывать исходный код на Python в эквивалентный код на C++. Трансляция должна поддерживать основные конструкции Python, включая переменные, операторы, циклы, условия, функции и классы.

1. FE-3 – Копирование результата на ЯП C++ в буфер обмена:

Система должна позволять пользователю копировать сгенерированный код на C++ в буфер обмена.

1. FE-4 – Форматирование исходного кода по принятым правилам в сообществе:

Система должна автоматически форматировать исходный код Python в соответствии с общепринятыми стандартами форматирования.

1. FE-5 – Вывод ошибки в исходном коде с информацией о типе ошибки и позиции в коде:

Система должна отслеживать ошибки в исходном коде (например, синтаксические ошибки или неподдерживаемые конструкции). При обнаружении ошибки система должна выводить сообщение, которое включает указание типа ошибки.

## Требования к подсистеме «Пользовательский интерфейс»

Описание пользовательского интерфейса:

1. Наличие редактируемого текстового поля для ввода исходного кода на Python.
2. Наличие нередактируемого, копируемого текстового поля для результата.
3. Наличие нередактируемого, текстового поля для вывода информации об ошибках.

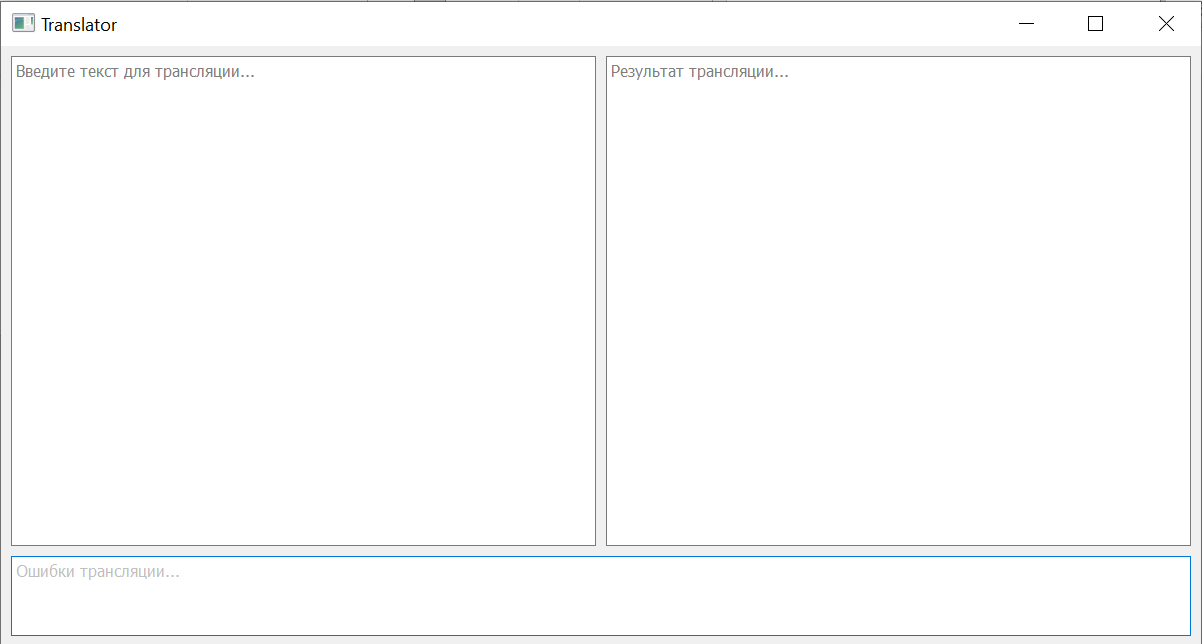


Рисунок 14 — Интерфейс программы

*Требование REQ\_UI\_001*

В пользовательском интерфейсе должна быть доступна функция ввода исходного кода на языке Python, через редактируемое текстовое поле.

*Требование REQ\_UI\_002*

В пользовательском интерфейсе должен появляться транслированный код на языке C++, если пользователь ввел данные в поле ввода исходного кода.

*Требование REQ\_UI\_003*

В пользовательском интерфейсе, при наличии ошибок в исходном коде, должна отображаться информация об ошибке.

*Требование REQ\_UI\_004*

Система должна форматировать результирующий код на языке C++ по принятым правилам.

*Требование REQ\_UI\_005*

В пользовательском интерфейса, должна быть возможность скопировать результирующий код на языке C++.

## Требования к подсистеме «Трансляция кода»

### Общие требования подсистемы

*Требование REQ\_T\_001*

Подсистема должна корректно транслировать вывод на консоль для логирования результатов работы.

### Требования к модулю токенизатора

*Требование REQ\_T\_002*

Токенизатор должен обнаруживать и корректно обрабатывать символы, которые не могут быть интерпретированы как часть допустимых лексем.

*Требование REQ\_T\_003*

Токенизатор должен обнаруживать строковые литералы, которые не закрыты кавычками или скобками.

### Требования к модулю лексического анализатора

*Требование REQ\_T\_004*

Анализатор должен выявлять незавершенные команды (например, неоконченные выражения, пропущенные операторы).

*Требование REQ\_T\_005*

Анализатор должен проверять, что блоки кода (циклы, условия, функции) имеют корректное завершение.

*Требование REQ\_T\_006*

Анализатор должен выявлять неподдерживаемые структуры данных, трансляция которых не была реализована (декораторы, списки).

### Требования к модулю формирования АСД

*Требование REQ\_T\_007*

Подсистема должна корректно формировать узлы дерева для условных конструкций, циклов и функций.

### Требования к модулю генерации кода

*Требование REQ\_T\_008*

Подсистема должна корректно транслировать условные конструкции.

*Требование REQ\_T\_009*

Подсистема должна корректно транслировать циклы.

*Требование REQ\_T\_010*

Подсистема должна корректно транслировать структуры данных (класс, тернарный оператор, лямбда функции).

*Требование REQ\_T\_011*

Подсистема должна обеспечить автоматическое определение типов переменных в C++ через auto (например: int, double, bool, std::string, и т. д.) при трансляции Python-кода.

## Пользовательские требования

Пользователь должен иметь возможности:

* Писать исходный код вручную в окно ввода в интерфейсе.
* Вставлять в окно ввода в интерфейсе исходный код из буфера обмена.
* Инициировать запуск трансляции кода.
* Получать и просматривать преобразованный код на C++ в окне вывода в интерфейсе.
* Получать и просматривать информацию об ошибках в исходном коде (если таковые имеются) в окне вывода ошибок в интерфейсе.

# Разработка архитектуры проекта

Архитектура программного обеспечения относится к фундаментальным структурам программной системы и дисциплине создания таких структур и систем. Каждая структура включает элементы программного обеспечения, отношения между ними, а также свойства как элементов, так и отношений. Архитектура программной системы — это метафора, аналогичная архитектуре здания. Он функционирует как план для системы и проекта разработки, в котором излагаются задачи, которые должны быть выполнены командами разработчиков. Была разработана архитектурно-контекстная диаграмма (Рисунок 15)

Одним из способов представления архитектуры проекта является диаграмма потоков данных.

Диаграммы потоков данных (Data Flow Diagrams — DFD) представляют собой иерархию функциональных процессов, связанных потоками данных. Цель такого представления — продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.

Была разработана диаграмма потоков данных внутри подсистемы разрабатываемого транслятора (Рисунок 8). На вход интерфейс принимает код на языке Python. Далее полученный код передается лексическому анализатору, который преобразует валидный код в массив токенов. Полученный массив токенов предоставляется синтаксическому анализатору для проверки синтаксиса входного языка. После проверки строится дерево разбора, которое в свою очередь уже будет обрабатывать сематический анализатор. Обработанное семантическим анализатором дерево передается генератору кода, который, в свою очередь, отдает интерфейсу код на языке C++.

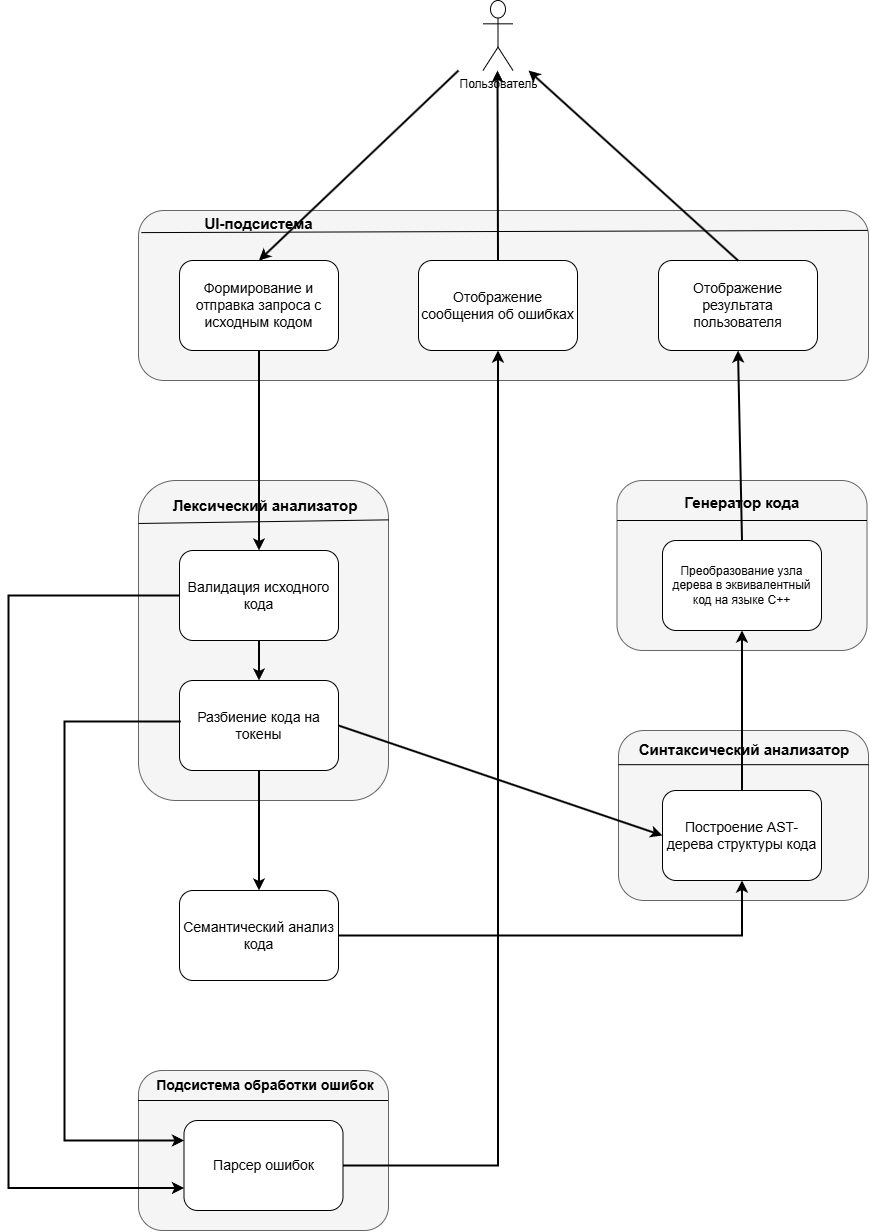


Рисунок 15 - Архитектурно-контекстная диаграмма.

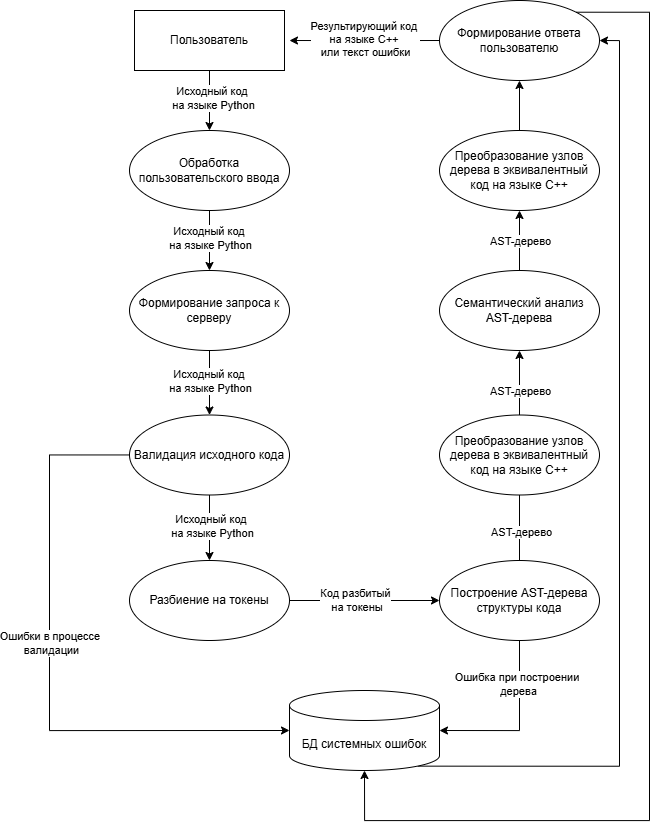


Рисунок 16 – Диаграмма потока данных в подсистеме транслятора

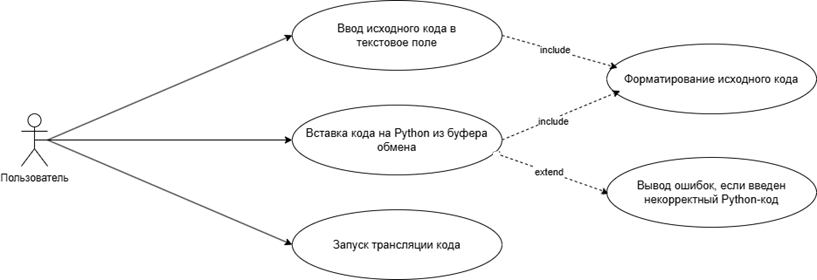


Рисунок 17 – Диаграмма вариантов использования системы

# Разработка измерений проекта

Контроль за производственным процессом и его результатами является ключевым видом деятельности на современном предприятии, производящем программное обеспечение на заказ. В силу специфики такого продукта, как программное обеспечение, для оценки эффективности процесса и качества конечного продукта применяются особые методы. Комплекс мероприятий, направленных на количественную оценку эффективности работы компании, называется программой измерений компании. Программа измерений выполняется как в рамках отдельных проектов, осуществляемых компанией, так и в рамках определённых видов деятельности компании.

## Метрики эффективности процесса разработки

1. **Среднее время разработки одной функциональности**

**Описание:** измеряет среднее время, затраченное на разработку, тестирование и интеграцию одной новой функции (например, поддержка новой конструкции языка).

**Формула:**

, (2)

где Ti — время разработки i-й функции, n— общее количество функциий.

**Цель:** минимизировать время разработки при сохранении качества кода.

**Сбор данных:** временные метки начала и конца работыre над задачей (например, из системы управления задачами).

1. **Плотность ошибок в процессе разработки**

**Описание:** измеряет количество ошибок (багов), выявленных на единицу объема кода, во время разработки.

**Формула:**

, (3)

где Eразработка— количество ошибок, найденных в ходе тестирования, Kстрок — общее число строк кода.

**Цель:** снизить количество ошибок при разработке.

**Сбор данных:** анализ репортов из системы тестирования и анализа кода.

## Метрики качества программного продукта

1. **Покрытие тестами**

**Описание:** показывает процент кода, который был проверен с помощью автоматических тестов.

**Формула:**

, (4)

где Kпокрытых— количество строк кода, проверенных тестами, Kвсего— общее количество строк кода.  
**Цель:** увеличивать покрытие тестами, чтобы минимизировать вероятность регрессий и багов.

**Сбор данных:** использование инструментов для измерения покрытия кода тестами.

1. **Точность трансляции**

**Описание:** измеряет процент корректных трансляций, где исходный код успешно преобразован в целевой язык без ошибок.

**Формула:**

, (5)

где Nуспешных— количество корректных трансляций, Nвсего— общее количество выполненных трансляций.

**Цель:** максимизировать точность трансляции.

**Сбор данных:** логи выполнения трансляции и автоматическая проверка результата.

# Разработка перечня задач проекта

Разработка перечня задач проекта включает формирование структурированного списка работ с детализацией по общим задачам и подсистемам.

## Общие задачи

[UF-1] Файловая структура

Необходимо создать модули для подсистем.

## Задачи для подсистемы «Пользовательский интерфейс»

[UI-1] Проектирование интерфейса

На основе требований разработать интерфейс. Необходимо добавить поля блоки с кодом и поле для вывода ошибок.

[UI-2] Вывод однородных ошибок

Необходимо собрать ошибки всех модулей и выводить их на экран пользователя в одном формате в случае их возникновения.

# Разработка рекомендаций по кодированию

Для создания качественного кода на любом языке программирования, обладающего таким свойствами, как удобочитаемость (readability) и понятность (understandability), необходимо следовать хорошо определённым стандартам и руководствам. Особенно это актуально при коллективной разработке программ. Любой стандарт кодирования призван определить набор правил, которые способствуют разработке более единообразного кода и минимизации числа общераспространенных ошибок в нем, не ущемляя при этом права разработчика на творчество.

Программная система будет реализована на языке программирования Python. PEP 8 — документ, описывающий соглашение о том, как писать код на языке Python. PEP 8 создан на основе рекомендаций создателя языка Гвидо ван Россума.  Ключевая идея Гвидо такова: код читается намного больше раз, чем пишется.

**Основные требования к оформлению кода:**

* Длина строки не должна превышать 79 символов.
* Отступы вложенных блоков кода — 4 пробела.
* Стиль именования переменных и функций соответствует разделу [PEP 8: Naming Conventions](https://peps.python.org/pep-0008/#prescriptive-naming-conventions).
* При переносе строк необходимо следовать [правилам отступов](https://peps.python.org/pep-0008/#indentation).
* Функции, объявленные на верхнем уровне программы (не вложенные в другие блоки кода), должны разделяться двумя пустыми строками.
* В коде проекта должны применяться кавычки одного типа: либо одинарные, внутри которых при необходимости используются двойные, либо наоборот.

**Основные рекомендации к оформлению кода:**

* Применять бэкслеши \ для переноса строк не рекомендуется.
* В коде не должно быть неиспользуемых импортов.
* Импорты в коде следует описывать согласно иерархии, в таком порядке: 1. Импорты из стандартной библиотеки. 2. Импорты из сторонних библиотек.   
  Внутри каждой группы импорты следует сортировать в алфавитном порядке.
* Комментарии должны начинаться со знака #. Символ # от текста комментария отделяется одним пробелом. Комментарий должен быть выровнен с кодом, к которому он относится, и следовать тем же правилам отступов.
* Если комментарий размещён в той же строке, что и код, между кодом и комментарием должно быть два пробела.
* Комментарий должен начинаться с заглавной буквы, а после комментария ставится точка: комментарий оформляется так же, как предложение в обычном тексте.
* Строка комментария не должна превышать 72 символа.

**Запреты при написании кода:**

* Запрещается использовать конструкции препроцессора, такие как #define, не предусмотренные в Python. Это не только бесполезно в Python, но и нарушает читаемость кода.
* Запрещается переопределять ключевые слова и операторы языка. Например, нельзя использовать конструкции вида closed = protected.
* Не допускается использование нестандартизированных средств, характерных для других языков программирования (например, макросов из C/C++).

# Разработка плана тестирования проекта

План тестирования определяет стратегию, методы, инструменты и этапы проверки функциональности и качества разрабатываемой системы.

## Тесты для подсистемы «Пользовательский интерфейс»

**Тест TEST\_UI\_001**

Тестируемые требования: REQ\_UI\_001

Проверка наличия поля ввода для исходного кода на Python.

Ожидаемое поведение: при загрузке страницы пользователь видит активное текстовое поле, в которое можно ввести или вставить Python-код.

**Тест TEST\_UI\_002**

Тестируемые требования: REQ\_UI\_001, REQ\_UI\_002

Проверка отображения результата трансляции. На вход подается корректный код на Python, вводимый в соответствующее поле. Ожидаемый результат – в поле для вывода отображается автоматически сгенерированный код на языке C++ с возможностью его копирования.

**Тест TEST\_UI\_003**

Тестируемые требования: REQ\_UI\_001, REQ\_UI\_003

Проверка отображения сообщений об ошибках. На вход подается некорректный код на языке Python (например, с синтаксической ошибкой). Ожидаемый результат – на странице появляется сообщение об ошибке с описанием причины и указанием строки, где она произошла.

**Тест TEST\_UI\_004**

Тестируемые требования: REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004

Проверка автоматического форматирования результата. На вход подается валидный, но неструктурированный код Python. Ожидаемый результат – результирующий C++ код отображается в читаемом виде с соблюдением правил отступов, переносов строк и стандартов форматирования.

**Тест TEST\_UI\_005**

Тестируемые требования: REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004, REQ\_UI\_005

Проверка возможности копирования результирующего кода на языке C++. На вход подается корректный Python код, после чего отображается транслированный C++ код. Ожидаемый результат — возможность скопировать сгенерированный C++ код с помощью кнопки «Копировать».

## Тесты для подсистемы «Транслятор кода»

**Тест TEST\_T\_001**

Тестируемые требования: REQ\_T\_001, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004

Проверка корректности трансляции вывода на консоль. На вход подается программа на языке Python, содержащая несколько операторов print() с различными типами данных. Ожидаемый результат – корректное преобразование в конструкции std::cout на языке C++ с соответствующим выводом значений.

**Тест TEST\_T\_002**

Тестируемые требования: REQ\_T\_002, REQ\_UI\_001, REQ\_UI\_003

Проверка обработки недопустимых символов. На вход подается программа, содержащая символы, не являющиеся частью допустимых лексем (например, print(©)). Ожидаемый результат – сообщение об ошибке лексического анализа, трансляция не выполняется, узлы дерева не формируются.

**Тест TEST\_T\_003**

Тестируемые требования: REQ\_T\_003, REQ\_UI\_001, REQ\_UI\_003

Проверка обработки незакрытых строковых литералов. На вход подается строка без закрывающей кавычки (например, print("Hello)). Ожидаемый результат – сообщение об ошибке лексического анализа, построение дерева прерывается.

**Тест TEST\_T\_004**

Тестируемые требования: REQ\_T\_004, REQ\_UI\_001, REQ\_UI\_003

Проверка выявления незавершённых команд. На вход подается программа с неоконченным выражением (например, a =). Ожидаемый результат – сообщение о синтаксической ошибке, дальнейшая трансляция не выполняется.

**Тест TEST\_T\_005**

Тестируемые требования: REQ\_T\_005, REQ\_UI\_001, REQ\_UI\_003

Проверка корректности завершения блоков кода. На вход подается программа с неполным определением блока (например, if x > 0: без тела). Ожидаемый результат – сообщение об ошибке структуры кода, построение дерева прерывается.

**Тест TEST\_T\_006\_1**, REQ\_UI\_001, REQ\_UI\_003

Тестируемые требования: REQ\_T\_006

Проверка обработки неподдерживаемой структуры данных — декоратора. На вход подается программа с функцией, использующей декоратор. Ожидаемый результат – сообщение об ошибке, указывающее на отсутствие поддержки конструкции.

**Тест TEST\_T\_006\_2**

Тестируемые требования: REQ\_T\_006, REQ\_UI\_001, REQ\_UI\_003

Проверка обработки неподдерживаемой структуры данных — списка. На вход подается программа, содержащая список (например, a = [1, 2, 3]). Ожидаемый результат – сообщение об ошибке, указывающее на отсутствие поддержки конструкции.

**Тест TEST\_T\_008**

Тестируемые требования: REQ\_T\_007, REQ\_T\_008, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004

Проверка корректности трансляции условных конструкций. На вход подается программа с оператором if-elif-else. Ожидаемый результат – корректное преобразование в условные конструкции C++ с сохранением логики выполнения.

**Тест TEST\_T\_009**

Тестируемые требования: REQ\_T\_007, REQ\_T\_009, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004

Проверка корректности трансляции цикла for. На вход подается программа с использованием цикла for и функции range(). Ожидаемый результат – преобразование в цикл for на C++ с корректной инициализацией, условием и инкрементом.

**Тест TEST\_T\_010\_1**

Тестируемые требования: REQ\_T\_007, REQ\_T\_010, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004

Проверка корректности трансляции пользовательского класса. На вход подается программа с определением класса, содержащего метод и его вызов. Ожидаемый результат – корректное преобразование в класс на языке C++ с соответствующими методами.

**Тест TEST\_T\_010\_2**

Тестируемые требования: REQ\_T\_007, REQ\_T\_010, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004

Проверка корректности трансляции тернарного оператора. На вход подается выражение в формате x = a if cond else b. Ожидаемый результат – корректное преобразование в x = cond ? a : b; в C++.

**Тест TEST\_T\_010\_3**

Тестируемые требования: REQ\_T\_007, REQ\_T\_010, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004

Проверка обработки лямбда-функций. На вход подается программа с определением и вызовом лямбда-функции. Ожидаемый результат – корректное преобразование функции.

**Тест TEST\_T\_011**

Тестируемые требования: REQ\_T\_007, REQ\_T\_011, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004

Проверка корректности автоматического определения типов переменных. На вход подается программа с несколькими переменными типов int, float, bool, str. Ожидаемый результат – переменные транслируются с использованием auto в C++ и соответствующих типов (int, double, bool, std::string).

# Тестирование проекта

Тестирование проекта включает системную проверку функциональности, производительности и безопасности системы для обеспечения соответствия требованиям и выявления дефектов перед выпуском.

Таблица 1 - Тесты

| **№** | **Входные данные** | **Ситуация** | **Выходные данные** |
| --- | --- | --- | --- |
| 1 | print("Hello, World) | Пользователь пробует вписать код в поля ввода (TEST\_UI\_001) | -  (Не проверяются) |
| 2 | print("Hello, World) | Тестирование отображения результата трансляции (TEST\_UI\_002) | std::cout <<  "Hello, World";  (Корректный C++ код) |
| 3 | print("Hello, World | Тестирование отображения сообщений об ошибках (TEST\_UI\_003) | Syntax Error.  (Сообщение об ошибке.) |
| 4 | def greet(name):\nprint("Hello,", name) | Тестирование автоматического форматирования результата (TEST\_UI\_004) | void greet(std::string name) {\n std::cout << "Hello, " << name;\n} (C++ код отформатирован с правильными отступами и структурой) |

*Продолжение таблицы 1 - Тесты*

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Входные данные** | **Ситуация** | **Выходные данные** |
| 5 | print("Hello") | Тестирование возможности копирования C++ кода (TEST\_UI\_005) | std::cout << "Hello"; (Возможность копировать C++ код с помощью кнопки «Копировать») |
| 6 | print("Hello", 123, True) | Проверка корректной трансляции вывода на консоль (TEST\_T\_001) | std::cout << "Hello" << 123 << true; (Трансляция в std::cout) |
| 7 | print(©) | Ввод содержит символы, не являющиеся допустимыми лексемами (TEST\_T\_002) | Syntax Error  (Ошибка лексического анализа, трансляция не выполняется.) |
| 8 | print("Hello) | Обработка незакрытых строковых литералов (TEST\_T\_003) | Syntax Error  (Ошибка синтаксического анализа, незакрытая строка.) |
| 9 | a = | Обработка незавершённой команды (TEST\_T\_004) | Syntax Error  (Ошибка синтаксического анализа, незавершенная команда.) |

*Продолжение таблицы 1 - Тесты*

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Входные данные** | **Ситуация** | **Выходные данные** |
| 10 | if x > 0: | Проверка корректности завершения блоков кода, отсутствует тело конструкции (TEST\_T\_005) | Syntax Error  (Ошибка структуры кода, отсутствует тело блока if.) |
| 11 | @decorator\ndef func(): pass | Обработка неподдерживаемой конструкции — декоратор (TEST\_T\_006\_1) | Decorator is not supported.  (Ошибка неподдерживаемой структуры, декоратор не поддерживается.) |
| 12 | a = [1, 2, 3] | Обработка неподдерживаемой конструкции — список (TEST\_T\_006\_2) | Collection “list” is not implemented yet.  (Ошибка неподдерживаемой структуры, списки не поддерживаются, возможно добавление поддержки в будущем) |
| 13 | if x > 0:\n print("yes")\nelif x < 0:\n print("no")\nelse:\n print("zero") | Проверка трансляции условной конструкции if-elif-else (TEST\_T\_008) | if (x > 0) { std::cout << "yes"; } else if (x < 0) { std::cout << "no"; } else { std::cout << "zero"; } (Корректная трансляция конструкции if-elif-else) |

*Окончание таблицы 1 - Тесты*

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Входные данные** | **Ситуация** | **Выходные данные** |
| 14 | for i in range(5): print(i) | Проверка трансляции цикла for (TEST\_T\_009) | for (int i = 0; i < 5; ++i) { std::cout << i; } (Корректная трансляция цикла for) |
| 15 | class MyClass:\n def method(self):\n print("Hello")\nobj = MyClass()\nobj.method() | Трансляция пользовательского класса (TEST\_T\_010\_1) | class MyClass { public: void method() { std::cout << "Hello"; } }; MyClass obj; obj.method(); (Корректное преобразование класса и метода) |
| 16 | x = a if cond else b | Трансляция тернарного оператора (TEST\_T\_010\_2) | x = (cond) ? a : b; (Преобразование в тернарный оператор) |
| 17 | f = lambda x: x \* 2\nprint(f(5)) | Обработка лямбда-функции (TEST\_T\_010\_3) | auto f = [](int x) { return x \* 2; }; std::cout << f(5);  (Трансляция лямбда-функции) |
| 18 | a = 5\nb = 2.0\nc = "text"\nd = True | Проверка автоматического определения типов переменных (TEST\_T\_011) | int a = 5; double b = 2.0; std::string c = "text"; bool d = true; (Использование типов int, double, std::string, bool в C++) |

На рисунке представлена матрица покрытия требований тестами (Requirements Traceability Matrix), отображающая соответствие между тестовыми сценариями и проверяемыми функциональными требованиями системы.

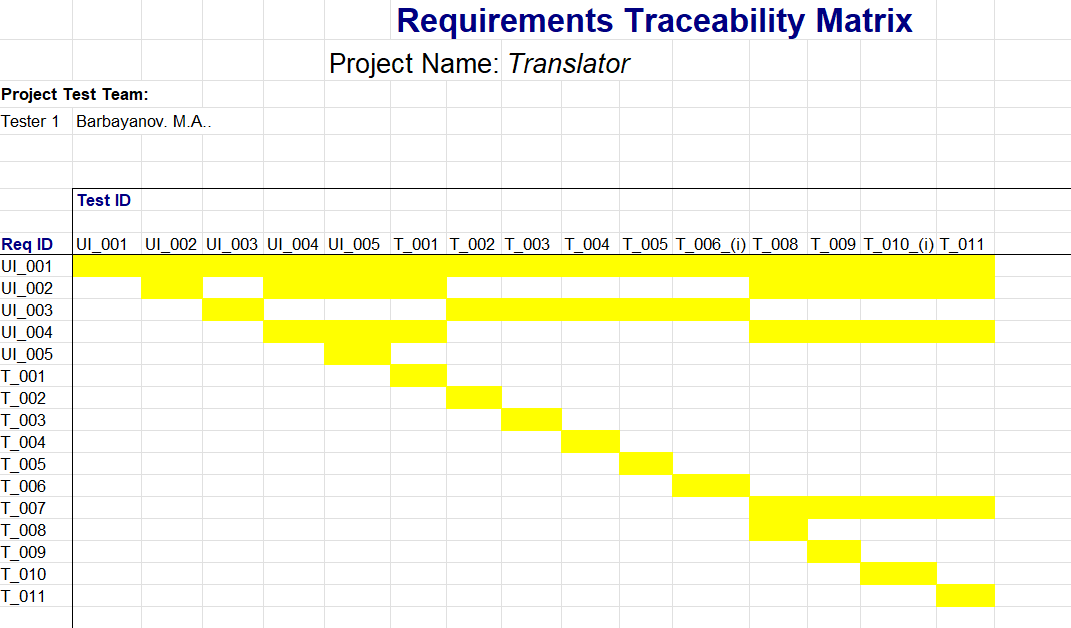


Рисунок 18 – Матрица покрытия требований.

# Заключение

По окончанию всех работ посчитаем метрики, описанные в 7 главе:

1. Количество дней на обработку задачи (PRR) составило 16.333 дней.

2. Эффективность обнаружения дефектов (FS) составила 78.352%

3. Плотность неполадок (BTF) составила 0.023 неполадок / LOC.

В рамках курсовой работы было разработано программное средство «Транслятор из Python в C++» с использованием подходов коллективной промышленной разработки, для чего были решены следующие поставленные задачи:

− Разработан план проекта;

− Разработан регламент проведения инспекции;

− Разработана модель состояний задач;

− Разработана презентация проекта;

− Разработаны требования к проекту;

− Разработана архитектура проекта;

− Разработать измерения проекта;

− Разработан перечь задач проекта;

− Разработаны рекомендации по кодированию;

− Разработан план тестирования проекта;

− Проект протестирован.

Таким образом, цель данного курсового проекта была достигнута.

# Список литературы

1. Гриняк В.М. Лекции по дисциплине «Технологии коллективной промышленной разработки информационных систем». / [Электронный ресурс] // BlackBoard: [сайт] — URL: https://bb.dvfu.ru/webapps/blackboard/content/listContent.jsp?course\_id=\_5025\_1&content\_id=\_172365\_1&mode=reset (дата обращения: 24.03.2025).
2. Компилятор за выходные: лексер и парсер/ [Электронный ресурс] // Хабр: [сайт] — URL: https://habr.com/ru/articles/787800/ (дата обращения: 14.04.2025).
3. The Python Language Reference / [Электронный ресурс] // Python Software Foundation: [сайт] — URL: https://docs.python.org/3/reference/ (дата обращения: 28.04.2025).
4. Crafting Interpreters / [Электронный ресурс] // Robert Nystrom:  
    [сайт] — URL: https://craftinginterpreters.com/ (дата обращения: 29.04.2025).