

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 ТРАНСЛЯТОРА С С++ НА PYTHON**

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. Б9121-09.03.04прогин  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Колпакова А. И.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Олейникова A. E.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Егорова К. С.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Мартын В.А

Руководитель:

Старший преподаватель ДПИиИИ  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Иваненко Ю. С.

г. Владивосток  
2025

Оглавление

[Введение 4](#_Toc192151848)

[1 Разработка плана проекта 5](#_Toc192151849)

[2 Разработка регламента проведения инспекции 6](#_Toc192151850)

[3 Разработка модели состояний задач 9](#_Toc192151851)

[4 Разработка презентации проекта 11](#_Toc192151852)

[5 Разработка требований к проекту 14](#_Toc192151853)

[6 Разработка архитектуры проекта 17](#_Toc192151854)

[7 Правила по кодированию 20](#_Toc192151855)

[8 Разработка плана тестирования проекта 22](#_Toc192151856)

[Тесты для тестирования подсистемы «Пользовательский интерфейс» 22](#_Toc192151857)

[Тесты для тестирования подсистемы «Лексический анализатор» 23](#_Toc192151858)

[Тесты для тестирования подсистемы «Синтаксический анализатор» 23](#_Toc192151859)

[Тесты для тестирования подсистемы «Семантический анализатор» 26](#_Toc192151860)

[Тесты для тестирования подсистемы «Генератор кода» 27](#_Toc192151861)

[Тесты для тестирования системы в целом 27](#_Toc192151862)

[Матрица покрытия тестами требований 28](#_Toc192151863)

[9 Тестирование проекта 30](#_Toc192151864)

[Тесты для тестирования подсистемы «Пользовательский интерфейс» 30](#_Toc192151865)

[Тесты для тестирования подсистемы «Лексический анализатор» 31](#_Toc192151866)

[Тесты для тестирования подсистемы «Синтаксический анализатор» 32](#_Toc192151867)

[Тесты для тестирования подсистемы «Семантический анализатор» 34](#_Toc192151868)

[Тесты для тестирования подсистемы «Генератор кода» 36](#_Toc192151869)

[Тесты для тестирования системы в целом 36](#_Toc192151870)

[Список литературы 38](#_Toc192151871)

**Введение**

В данной курсовой работе рассматривается задача коллективной разработки проекта «Разработка транслятора с С++ на Python» и составление технической документации к нему.

Таким образом, целью курсовой работы является разработка разработки проекта «Разработка транслятора с С++ на Python» с использованием подходов коллективной промышленной разработки.

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**

Исполнители проекта:

* Team Leader – Колпакова Александра;
* Coder 1 – Колпакова Александра;
* Coder 2 – Олейникова Анна;
* Coder 3 – Егорова Ксения;
* Coder 4 – Мартын Владислав;
* Build Engineer – Колпакова Александра;
* Technical Writer – Олейникова Анна.

Перечень задач и сроки их выполнения представлены на рисунке 1.

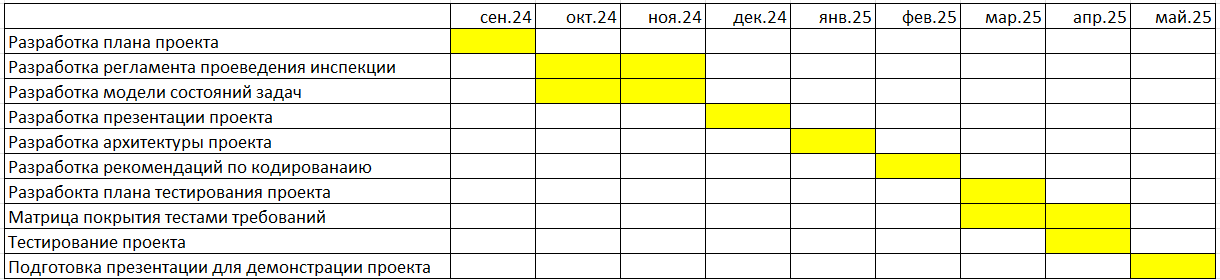


Рисунок 1 – Календарный план работы над проектом

# **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.

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:

* обнаружить ошибки в функциях, логике, содержании или реализации рабочих продуктов на ранних этапах их разработки и предотвратить их наследование;
* рационально донести замысел или реализацию продукта до всех заинтересованных лиц (через их участие);
* оптимизировать, оценить или улучшить рабочий продукт.

В таблице 1 представлены критерии отнесения к формальной/неформальной инспекции для различных типов рабочих продуктов.

|  |  |  |
| --- | --- | --- |
| Таблица 1 – Критерии отнесения к формальной/неформальной инспекции для различных типов рабочих продуктов | | |
|  | **Формальная инспекция в случае CRUD-операций над** | **Неформальная инспекция в случае CRUD-операций над** |
| **Требования** | >1 требованием | 1 требованием |
| **Документы дизайна** | >1 объектом дизайна внешнего вида готового продукта | 1 объектом дизайна внешнего вида готового продукта (изображение) |
| **Код** | >7 строчками | <8 строчками |
| **Тесты** | >3 тестами | < 4тестами |

**Перечень ролей участников инспекции и их обязанности**

Автор (Author) – сотрудник, разработавший инспектируемый рабочий продукт, либо сделавший инспектируемые изменения в существующем рабочем продукте.

Инспектор (Inspector) – сотрудник, ответственный за эффективную проверку инспектируемого рабочего продукта.

**Этапы инспекции**

1. Планирование инспекции.

При планировании инспекции коллективно выбирается дата, время, формат (очный или заочный) и платформа (при заочной инспекции) проведения инспекции.

1. Подготовка к инспекции.

Инспектор самостоятельно изучает предоставленный для инспекции рабочий продукт, используя накопленный опыт и стандарты.

1. Собрание по инспекции.

На собрании происходит обсуждение замечаний и рекомендаций инспектора по рабочему продукту. На собрании по инспекции обязательно присутствует инспектор и автор рабочего продукта, требующего инспекции. Присутствие остальных участников команды разработки по желанию.

1. Завершение инспекции.

Если рабочий продукт требует доработки, то автор фиксирует все замечания и рекомендации инспектора, разрабатывает план предстоящих работ и далее согласовывает его с инспектором. Если рабочий продукт не требует доработки, то инспектор подтверждает слияние рабочей ветки в «master» ветку.

**Перечень статусов и степени важности замечаний**

1. Ошибка – проблема, которая найдена на той же фазе, на которой внесена. Допустимые значения степени серьезности ошибки:

* критическая (critical);
* средняя (moderate);
* мелкая, незначительная (minor);
* другие (other).

1. Комментарий – это наблюдение, предложение, рекомендация или улучшение, предложенное для будущего выпуска рабочего продукта или вопрос, требующий разъяснения.

**Метрики, характеризующие эффективность инспекций**

Стратегическая цель метрики – повысить качество разрабатываемого ПО.

В качестве метрики, характеризующей эффективность инспекции, была выбрана Inspection Preparation Rate (IPR), рассчитываемая по формуле (1).

Изучаемый объект метрики – подготовка к инспекции, измеряемый атрибут – производительность подготовки к инспекции.

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 состояний задач:**

1. New – новая подзадача.
2. Analysis – в процессе анализа. В это состояние подзадачу переводит сотрудник после того, как начнет ее анализ.
3. Forward – в данном случае имеет значение «переданный на дальнейшую разработку». В это состояние задача переводится CCB после анализа при назначении задачи на разработку конкретному сотруднику.
4. Coding – кодирование. В это состояние задача переводится сотрудником разработчиком, при начале работы по кодированию, связанному с задачей.
5. Inspected – проинспектировано. В это состояние задача переводится сотрудником-разработчиком после завершения кодирования и инспектирования изменений рабочего продукта.
6. Integrated – заинтегрировано. Переводится сотрудником, осуществляющим интеграцию изменений в основную ветку рабочего продукта после успешной интеграции этих изменений.
7. Tested – протестировано. Переводится сотрудником, осуществляющим тестирование изменений в рабочий продукт (tester).
8. Closed – закрыто. В это состояние задача переводится CCB по результатам отчета о тестировании сделанных изменений.

**Правила создания новой задачи**

Созданием новых задач могут заниматься все участники команды. Происходит это на начальном этапе, когда необходимо организовать структурированную работу над проектом и в процессе работы над проектом в случае присутствия в плане слишком объемных задач, требующих упрощения. Также, когда необходимо организовать баг-фиксинг или разработать новый функционал.

**Правила перехода задачи из состояния в состояние**

Состояния задач всегда идут последовательно друг за другом, в некоторых случаях пункты могут опускаться или повторяться. Схема перехода из состояния в состояние показана на рисунке 2.

Изображение выглядит как диаграмма, текст, снимок экрана, линия

Автоматически созданное описание

Рисунок 2 – Состояния задачи

# **Разработка презентации проекта**

Первый слайд презентации на рисунке 3 содержит состав команды.



Рисунок 3 – Первый слайд

Второй слайд на рисунке 4 содержит краткое описание проекта.

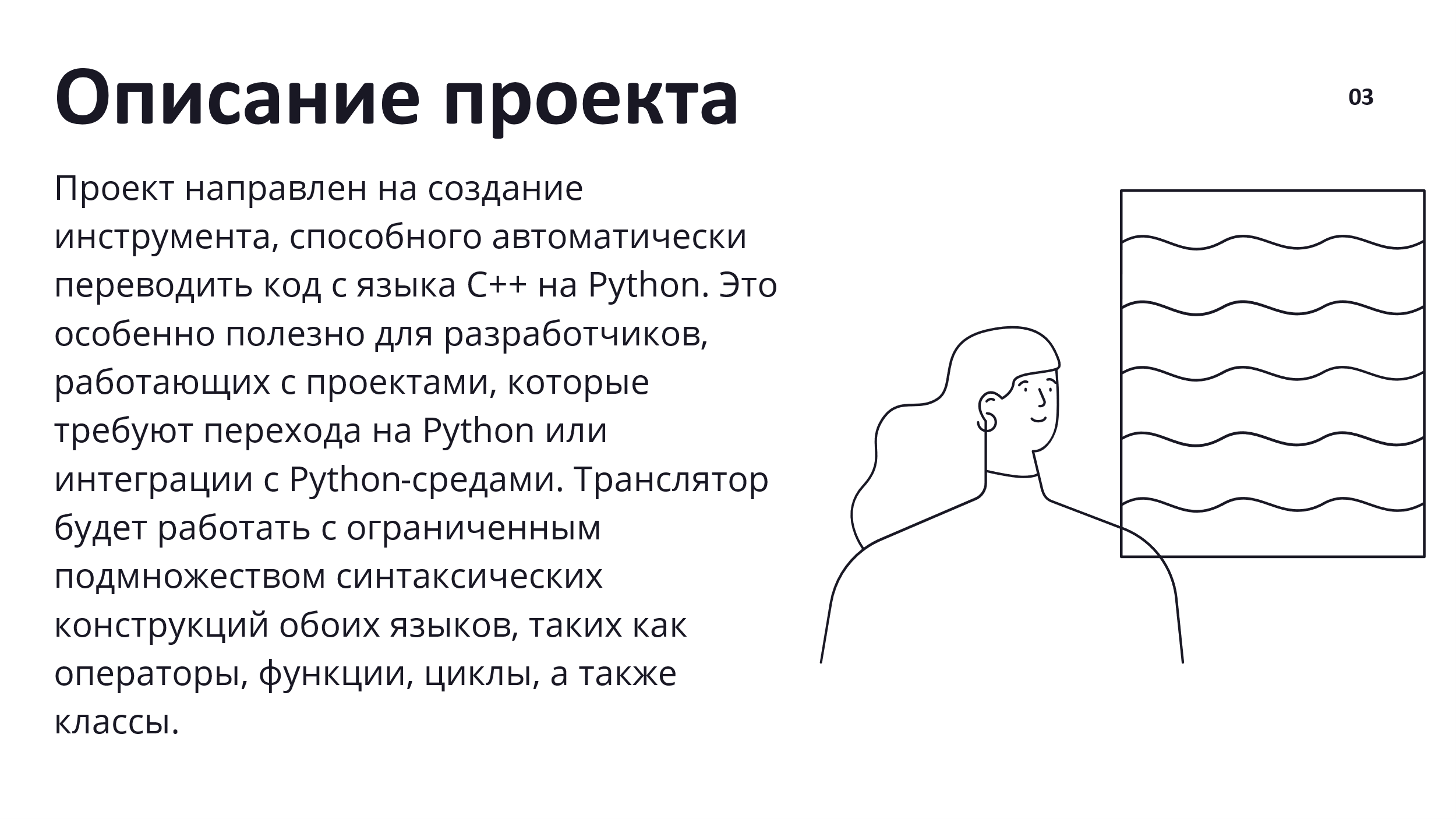


Рисунок 4 – Второй слайд

На третьем слайде представлены преимущества проекта (рисунок 5).

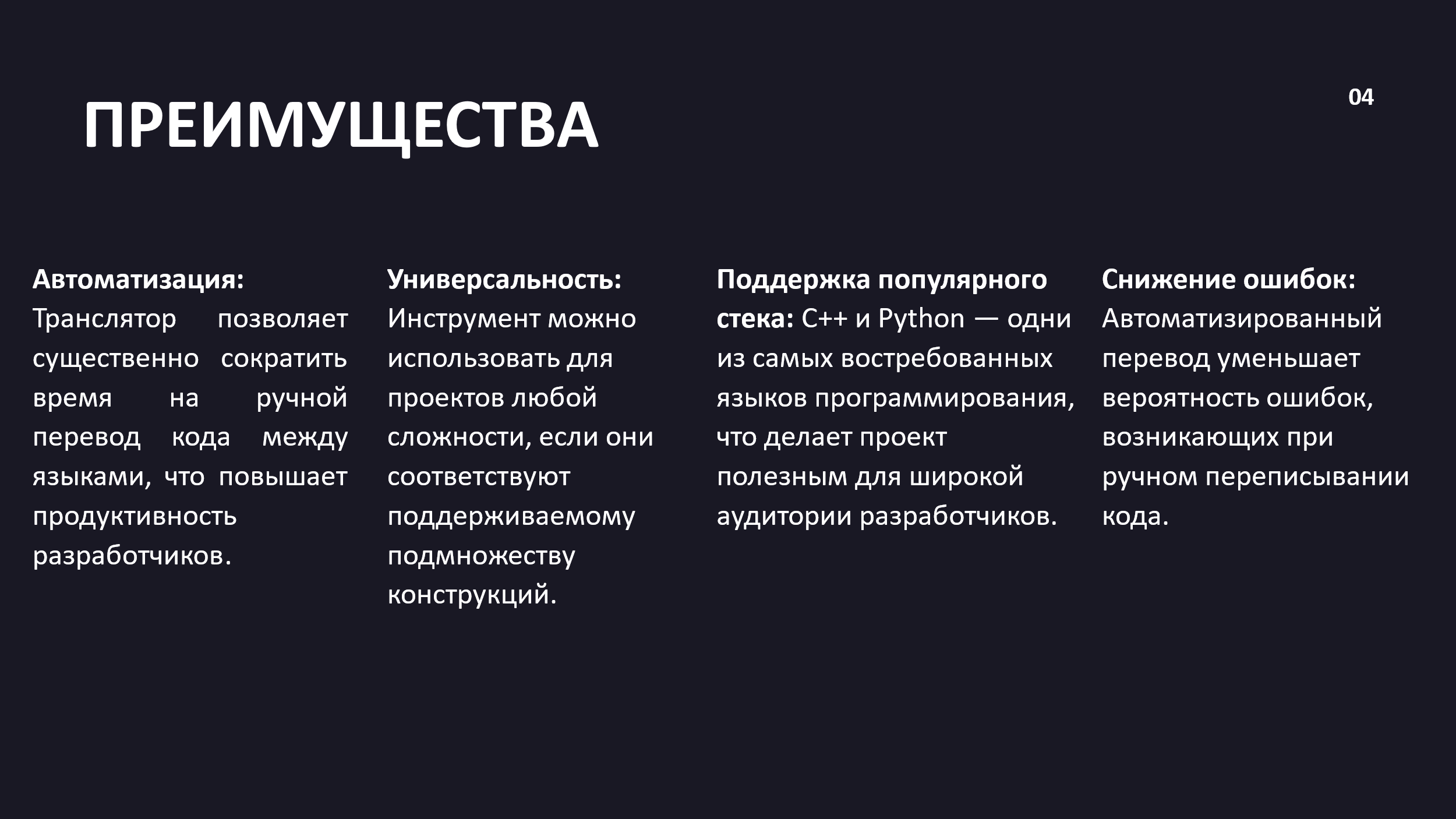


Рисунок 5 – Третий слайд

На четвертом слайде представлен прототип проекта пользовательского интерфейса (рисунок 6).

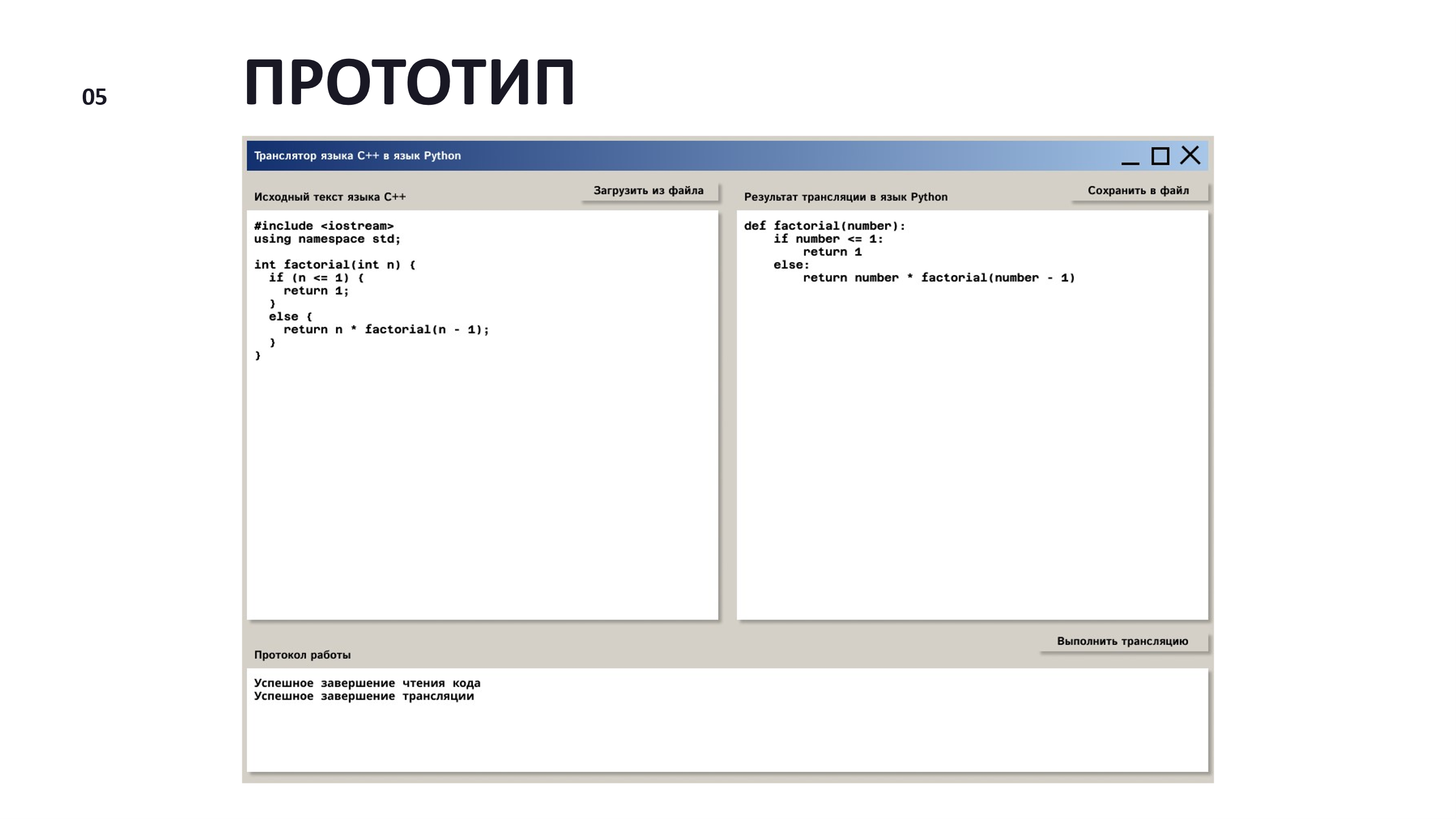


Рисунок 6 – Четвертый слайд

На пятом слайде представлен основной функционал проекта (рисунок 7).

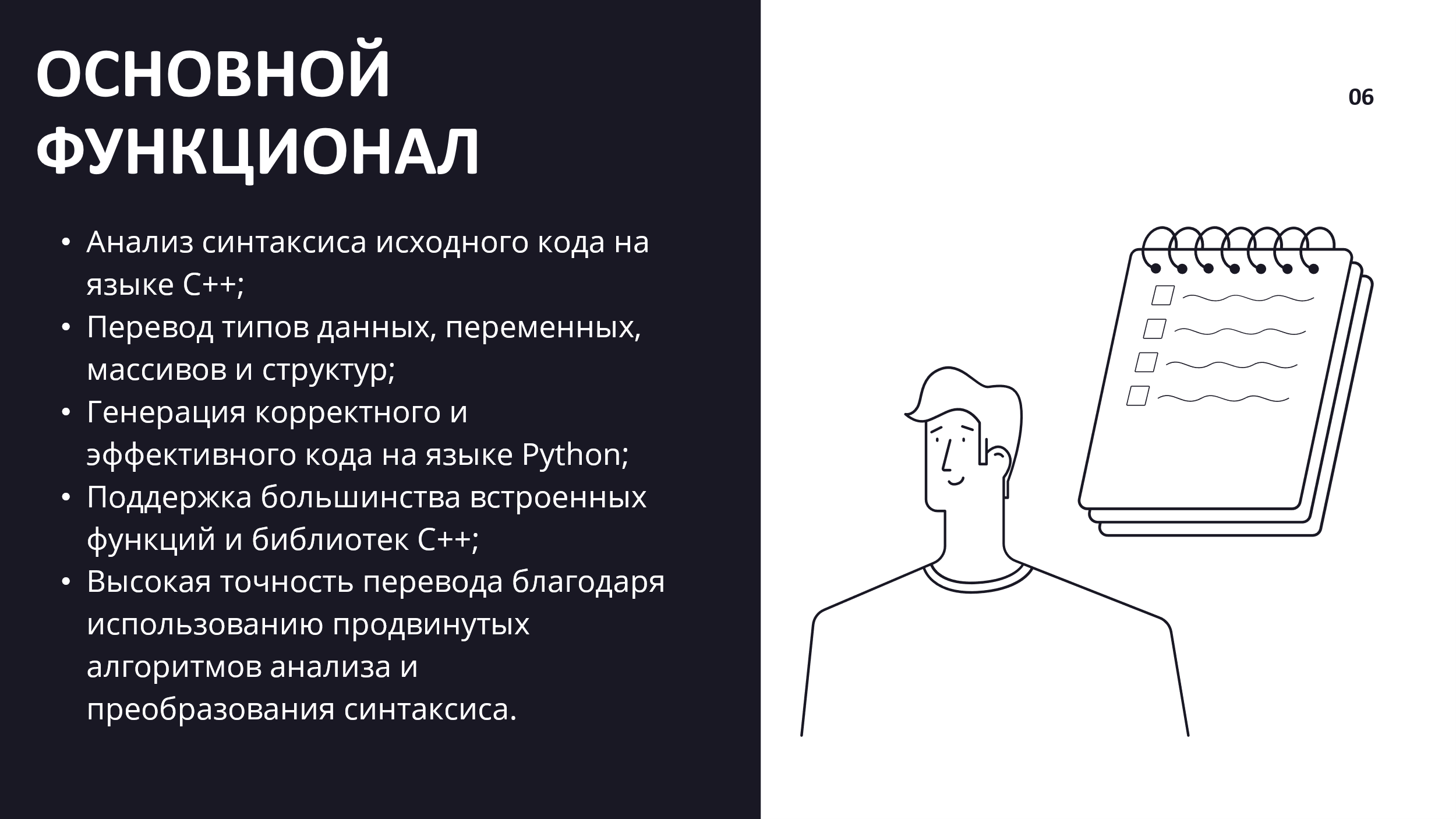


Рисунок 7 – Пятый слайд

# **Разработка требований к проекту**

Программный продукт *Транслятор С++ -> Python* предназначен для трансляции (перевода) с языка программирования С++ на язык Python.

Программный продукт *Транслятор С++ -> Python* состоит из следующих подсистем:

1. пользовательский интерфейс (UI);
2. лексический анализатор (LA);
3. синтаксический анализатор (SA);
4. семантический анализатор (SemA);
5. генератор кода (CG).

**Требования к подсистеме «Пользовательский интерфейс»**

Требование REQ\_UI\_001

Пользовательский интерфейс продукта должен содержать:

* поле ввода исходного кода на языке C++;
* поле вывода транслированного кода на языке Python;
* кнопку трансляции языка;
* кнопку загрузки файла;
* кнопку выгрузки файла;
* окно протокола работы.

Требование REQ\_UI\_002

При допущении пользователем ошибки, при написании в поле ввода исходного кода и нажатии на кнопку трансляции, в отдельном окне должно выводиться сообщение об ошибке.

Требование REQ\_UI\_003

Кнопка трансляции языка на рисунке 8 должна иметь вид прямоугольника с подписью «Выполнить трансляцию».



Рисунок 8 – Кнопка трансляции

Требование REQ\_UI\_004

Поля ввода и вывода исходного и транслированного языка соответственно, должны располагаться на одном уровне, причем поле ввода должно находиться левее.

Требование REQ\_UI\_005

Кнопка выгрузки файла на рисунке 9 должна иметь вид прямоугольника с подписью «Загрузить из файла».



Рисунок 9 – Кнопка выгрузки

Требование REQ\_UI\_006

Кнопка загрузки в файл на рисунке 10 должна иметь вид прямоугольника с подписью «Сохранить в файл».



Рисунок 10 – Кнопка загрузки

Требование REQ\_UI\_007

Поле вывода протокола работы должно располагаться под полями ввода и вывода кода.

**Требования к подсистеме «Лексический анализатор»**

Требование REQ\_LA\_001

На вход подсистеме поступает исходный код на языке С++. На выход подсистемы поступает множество токенов, либо ошибка.

Требование REQ\_LA\_002

Каждый токен представляет собой четверку <тип, значение, номер строки, номер столбца>.

**Требования к подсистеме «Синтаксический анализатор»**

Требование REQ\_SA\_001

Синтаксический анализатор получает на вход массив токенов от лексического анализатора. На его основе строит абстрактное синтаксическое дерево разбора, если синтаксических ошибок нет, иначе – выдает ошибку.

**Требования к подсистеме «Семантический анализатор»**

Требование REQ\_SemA\_001

Подсистема получает на вход дерево разбора от синтаксического анализатора, обеспечивает анализ входных данных на соответствие заданным семантическим правилам и при отсутствии семантических ошибок выводит надпись «Семантический анализ прошел успешно!», иначе выдает ошибку.

**Требования к подсистеме «Генератор кода»**

Требование REQ\_CG\_001

Генератор кода получает на вход дерево разбора от синтаксического анализатора. На выходе выводит код программы на языке Python.

1. **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.

Программный продукт *Транслятор C++ -> Python* состоит из следующих подсистем:

* пользовательский интерфейс;
* лексический анализатор;
* синтаксический анализатор;
* семантический анализатор;
* генератор кода.

**Контекстная диаграмма потоков данных**

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.

На рисунке 11 представлена диаграмма потоков данных

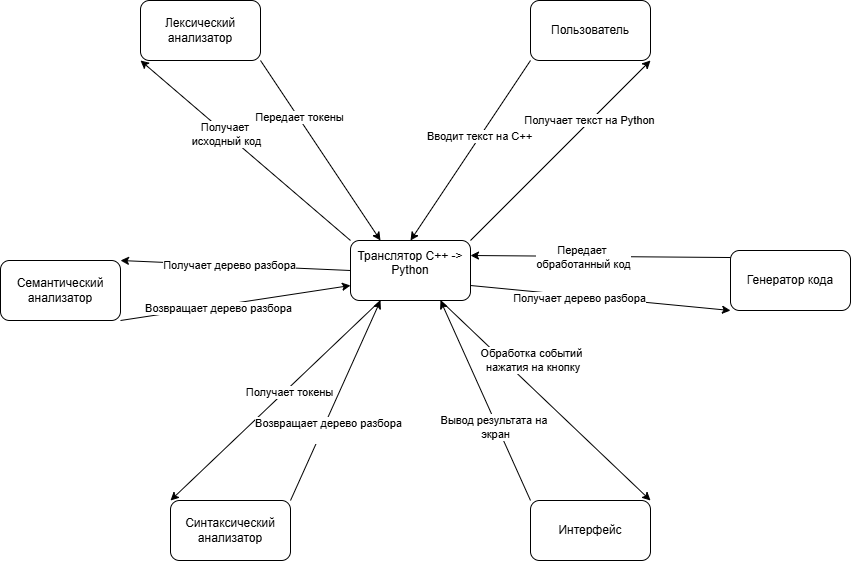


Рисунок 11 – Контекстная диаграмма

**Диаграмма User Flow**

User Flow (пользовательский поток) – это визуальное представление последовательности действий, которые пользователь выполняет для достижения своей цели в приложении. Он представляет собой блок-схемы, где визуально отображается каждый шаг, который проходит пользователь от точки входа до достижения конечной цели.

Цель такого представления – продемонстрировать, путь пользователя взаимодействия с интерфейсом от ввода исходного кода на языке C++ до получения транслируемого кода на Python.

На рисунке 12 представлена диаграмма User Flow.

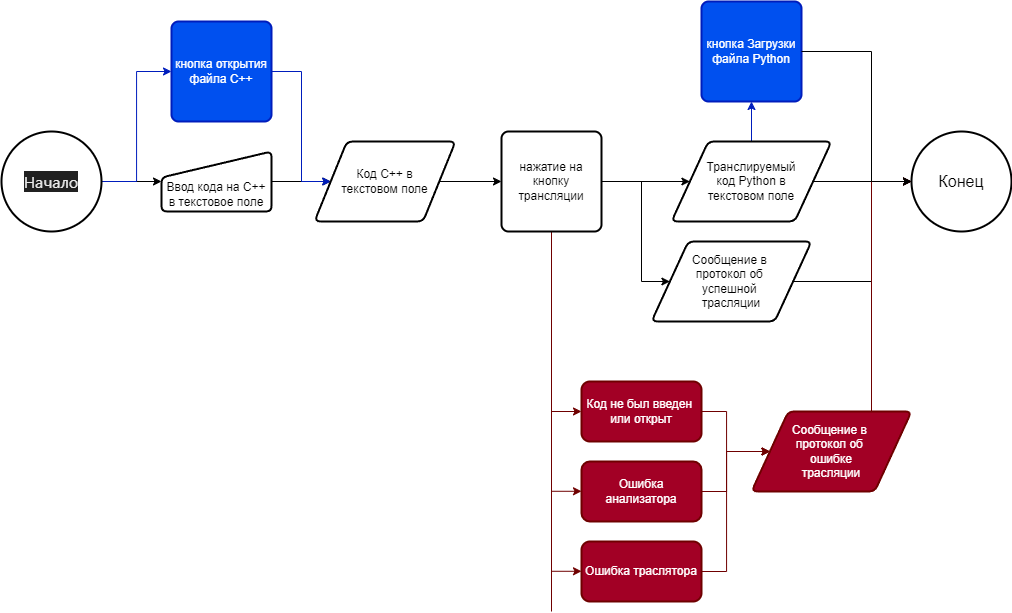


Рисунок 12 – Диаграмма User Flow

1. **Правила по кодированию**

При написании кода на языке программирования python необходимо опираться на стандарт написания кода PEP8. Этот документ описывает соглашение о том, как писать код для языка python, включая стандартную библиотеку, входящую в состав python.

Ниже перечислены некоторые пункты, для полного ознакомления вам необходимо прочитать полное руководство по написанию кода на python.

**Отступы**

Используйте 4 пробела на каждый уровень отступа.

Продолжительные строки должны выравнивать обернутые элементы либо вертикально, используя неявную линию в скобках (круглых, квадратных или фигурных), либо с использованием висячего отступа. При использовании висячего отступа следует применять следующие соображения: на первой линии не должно быть аргументов, а остальные строки должны четко восприниматься как продолжение линии.

**Табуляция**

Табуляция должна использоваться только для поддержки кода, написанного с отступами с помощью табуляции.

Максимальная длина строки – ограничьте длину строки максимум 79 символами. Для более длинных блоков текста с меньшими структурными ограничениями (строки документации или комментарии), длину строки следует ограничить 72 символами.

**Кодировка исходного файла**

Кодировка Python должна быть UTF-8 (ASCII в Python 2).

**Импорты**

Каждый импорт, как правило, должен быть на отдельной строке, пример на рисунке 13.

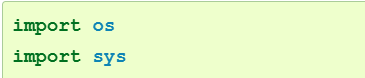
****

Рисунок 13 – Пример работы с import

Импорты всегда помещаются в начале файла, сразу после комментариев к модулю и строк документации, и перед объявлением констант.

Импорты должны быть сгруппированы в следующем порядке:

* 1. импорты из стандартной библиотеки,
  2. импорты сторонних библиотек,
  3. импорты модулей текущего проекта.

Вставляйте пустую строку между каждой группой импортов.

Указывайте спецификации \_\_all\_\_ после импортов.

**Именование переменных и функций**

При именовании переменных и функций следует руководствоваться исключительно назначением соответствующей переменной или функцией. При именовании использовать прописные латинские буквы в формате snake\_case. Если функция используется для получения какого-то значения использовать префикс get, если устанавливает какое-то значение – set. Для объявления констант использовать UPPER\_SNAKE\_CASE.

**Именование классов**

Для именования классов использовать CapWords, классы именовать в единственном числе.

**8 Разработка плана тестирования проекта**

**Тесты для тестирования подсистемы «Пользовательский интерфейс»**

***Тест TEST\_UI\_001***

Тестируемые требования: *REQ\_UI\_001*, REQ\_UI\_003, REQ\_UI\_005, REQ\_UI\_006.

Описание теста: проверка наличия всех элементов интерфейса. Открыть приложение и убедиться, что присутствуют: поле ввода кода C++; поле вывода кода Python; кнопка трансляции; кнопка загрузки файла; кнопка выгрузки файла; окно протокола. Проверить внешний вид кнопок трансляции, открытия файла и загрузки файла. Убедиться, что кнопки имеют прямоугольную форму и содержит подпись «Транслировать», «Открыть файл» и «Загрузить файл».

Ожидаемый результат: все перечисленные элементы присутствуют в пользовательском интерфейсе. Кнопки имеют прямоугольную форму и надписи «Транслировать», «Открыть файл» и «Загрузить файл».

**Тест TEST\_UI\_002**

Тестируемые требования: REQ\_UI\_002

Описание теста: ввести в поле ввода кода C++ строку с ошибкой и нажать кнопку трансляции. Проверить, отображается ли окно с сообщением об ошибке: «Введите код на C++!», «Ошибка парсера:», «Ошибка АСД», «Ошибка генератора».

Ожидаемый результат: в протоколе трансляции появляется сообщение об ошибке.

**Тест TEST\_UI\_003**

Тестируемые требования: REQ\_UI\_004, REQ\_UI\_007

Описание теста: открыть приложение и проверить расположение полей. Убедиться, что поле ввода кода C++ находится слева, а поле вывода кода Python — справа, и оба поля находятся на одном уровне. Поле трансляции расположено ниже, и ширина поля соответствует ширине экрана.

Ожидаемый результат: поля ввода и вывода расположены горизонтально, поле ввода находится левее поля вывода, поле трансляции ниже и по ширине экрана.

**Тесты для тестирования подсистемы «Лексический анализатор»**

***Тест TEST\_LA\_001***

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

Описание теста: Лексический анализатор принимает на вход строку с текстом кода на C++. Необходимо проверить, что анализатор корректно распознает ключевые слова, идентификаторы, разделители и числовые значения и преобразовывает их в соответствующие токены

Ожидаемый результат: Лексический анализатор возвращает список токенов, соответствующий входному коду. Каждый токен представлен в формате <тип, значение, номер строки, номер столбца>. Ошибки не возникают.

***Тест TEST\_LA\_002***

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

Описание теста: Лексический анализатор принимает на вход строку с текстом кода на C++ в котором содержится недопустимый символ «#». Анализатор должен выявить ошибку и сообщить о ней пользователю в окне протокола трансляции.

Ожидаемый результат: Лексический анализатор выявляет неизвестный символ. Выдается сообщение об ошибке. Ошибка отображается в отдельном окне пользовательского интерфейса. Анализатор прекращает дальнейшую обработку ввода после обнаружения ошибки

**Тесты для тестирования подсистемы «Синтаксический анализатор»**

**Тест TEST\_SA\_001**

Тестируемые требования: REQ\_UI\_001, REQ\_UI\_002, REQ\_LA\_001, REQ\_SA\_001.

Описание теста:

1. нажать на поле ввода исходного кода на языке С++;
2. ввести код с ошибкой введения неожидаемого типа токена

int int main() {

while (true) {

break;

}

}

1. нажать на кнопку «Выполнить трансляцию».

Ожидаемый результат: окно протокола работы содержит сообщение о синтаксической ошибке вида SyntaxError: Expected identifier after type keyword.

**Тест TEST\_SA\_002**

Тестируемые требования: REQ\_UI\_001, REQ\_LA\_001, REQ\_SA\_001.

Описание теста:

1. нажать на поле ввода исходного кода на языке С++;
2. ввести код без синтаксических ошибок

int main() {

int y = 10;

return y;

}

1. нажать на кнопку «Выполнить трансляцию».

Ожидаемый результат: окно протокола работы содержит список токенов и дерево разбора.

Окно протокола работы:

[Token(type='NEWLINE', value='\n', line=2, column=1), Token(type='KEYWORD', value='int', line=2, column=5), Token(type='IDENTIFIER', value='main', line=2, column=9), Token(type='SEPARATOR', value='(', line=2, column=13), Token(type='SEPARATOR', value=')', line=2, column=14), Token(type='SEPARATOR', value='{', line=2, column=16), Token(type='NEWLINE', value='\n', line=3, column=17), Token(type='KEYWORD', value='int', line=3, column=9), Token(type='IDENTIFIER', value='y', line=3, column=13), Token(type='OPERATOR', value='=', line=3, column=15), Token(type='NUMBER', value='10', line=3, column=17), Token(type='SEPARATOR', value=';', line=3, column=19), Token(type='NEWLINE', value='\n', line=4, column=20), Token(type='KEYWORD', value='return', line=4, column=9), Token(type='IDENTIFIER', value='y', line=4, column=16), Token(type='SEPARATOR', value=';', line=4, column=17), Token(type='NEWLINE', value='\n', line=5, column=18), Token(type='SEPARATOR', value='}', line=5, column=5), Token(type='NEWLINE', value='\n', line=6, column=6)]

AST Output:

Statements:

Function: int main

Parameters:

Body:

Statements:

Variable Declaration: int y

Assigned value:

Value: 10

Return:

Unknown node type: <class 'nodes.VariableUsageNode'>

**Тесты для тестирования подсистемы «Семантический анализатор»**

**Тест TEST\_SemA\_001**

Тестируемые требования: REQ\_UI\_001, REQ\_UI\_002, REQ\_LA\_001, REQ\_SA\_001, REQ\_SemA\_001.

Описание теста:

1. нажать на поле ввода исходного кода на языке С++;
2. ввести код с семантической ошибкой

int main() {

    int x;

    y = 10;

    return 0;

}

1. нажать на кнопку «Выполнить трансляцию».

Ожидаемый результат: окно протокола работы содержит список токенов, дерево разбора, результат работы семантического анализатора - Семантическая ошибка: Переменная ‘у’ не определена.

**Тест TEST\_SemA\_002**

Тестируемые требования: REQ\_UI\_001, REQ\_LA\_001, REQ\_SA\_001, REQ\_SemA\_001.

Описание теста:

1. нажать на поле ввода исходного кода на языке С++;
2. ввести код без семантических, лексических и синтаксических ошибок

int main() {

    int x;

    х = 10;

    return 0;

}

1. нажать на кнопку «Выполнить трансляцию».

Ожидаемый результат: окно протокола работы содержит список токенов, дерево разбора, результат работы семантического анализатора - Семантический анализ прошел успешно!

**Тесты для тестирования подсистемы «Генератор кода»**

**Тест TEST\_CG\_001**

Тестируемые требования: REQ\_UI\_001, REQ\_LA\_001, REQ\_SA\_001, REQ\_SemA\_001, REQ\_CG\_001.

Описание теста:

1. нажать на поле ввода исходного кода на языке С++;
2. ввести код без лексических, синтаксических и семантических ошибок

int add(int a, int b) {

return a + b;

}

void greet() {

std::cout << "Hi!" ;

}

1. нажать на кнопку «Выполнить трансляцию».

Ожидаемый результат: окно протокола работы содержит список токенов, дерево разбора, результат работы семантического анализатора, поле вывода транслированного кода на языке Python содержит код на языке Python.

Поле вывода транслированного кода на языке Python:

def add(a, b):

return a + b

def greet():

print("Hi!")

**Тесты для тестирования системы в целом**

***Тест TEST\_SYS\_001***

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, REQ\_UI\_009, REQ\_LA\_001, REQ\_LA\_002, REQ\_SA\_001, REQ\_SemA\_001, REQ\_CG\_001*.*

Описание теста: запустить программу, ввести исходный код на языке C++, нажать кнопку «Транслировать».

Ожидаемый результат: запуститься окно интерфейса, транслируемый код на Python будет выведен в правом окне, ошибок не возникнет.

**Матрица покрытия тестами требований**

Матрица соответствия требований используется QA-инженерами для валидации покрытия требований по продукту тестами. Цель «Traceability Matrix» состоит в том, чтобы выяснить:

* какие требования «покрыты» тестами, а какие нет;
* избыточность тестов (одно функциональное требование покрыто большим количеством тестов).

Данный тестовый артефакт является неотъемлемой частью тестирования.

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

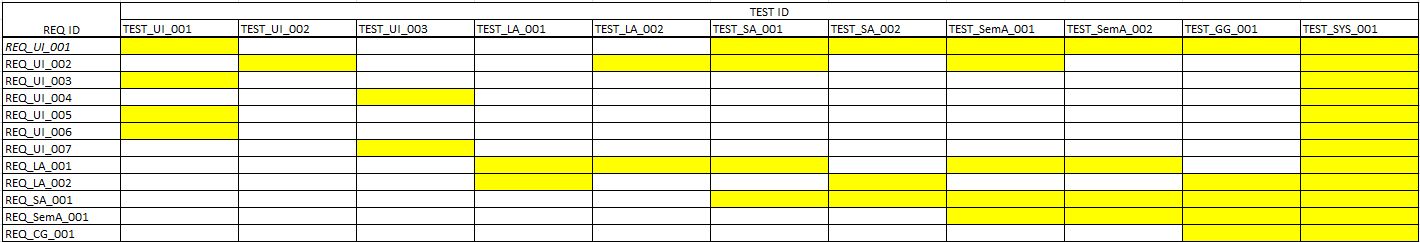


Рисунок 11 - Матрица покрытия тестами требований

**9 Тестирование проекта**

**Тесты для тестирования подсистемы «Пользовательский интерфейс»**

***Тест TEST\_UI\_001***

Тестируемые требования: *REQ\_UI\_001*, REQ\_UI\_003, REQ\_UI\_005, REQ\_UI\_006.

Описание теста: проверка наличия всех элементов интерфейса. Открыть приложение и убедиться, что присутствуют: поле ввода кода C++; поле вывода кода Python; кнопка трансляции; кнопка загрузки файла; кнопка выгрузки файла; окно протокола. Проверить внешний вид кнопок трансляции, открытия файла и загрузки файла. Убедиться, что кнопки имеют прямоугольную форму и содержит подпись «Транслировать», «Открыть файл» и «Загрузить файл».

Ожидаемый результат: все перечисленные элементы присутствуют в пользовательском интерфейсе. Кнопки имеют прямоугольную форму и надписи «Транслировать», «Открыть файл» и «Загрузить файл».

Видимый результат: полностью совпадает с ожидаемым

Тестируемая версия продукта:1.0.0

Резюме: Тест пройден

**Тест TEST\_UI\_002**

Тестируемые требования: REQ\_UI\_002

Описание теста: ввести в поле ввода кода C++ строку с ошибкой и нажать кнопку трансляции. Проверить, отображается ли окно с сообщением об ошибке: «Введите код на C++!», «Ошибка парсера:», «Ошибка АСД», «Ошибка генератора».

Ожидаемый результат: в протоколе трансляции появляется сообщение об ошибке.

Видимый результат: полностью совпадает с ожидаемым

Тестируемая версия продукта:1.0.0

Резюме: Тест пройден

**Тест TEST\_UI\_003**

Тестируемые требования: REQ\_UI\_004, REQ\_UI\_007

Описание теста: открыть приложение и проверить расположение полей. Убедиться, что поле ввода кода C++ находится слева, а поле вывода кода Python — справа, и оба поля находятся на одном уровне. Поле трансляции расположено ниже и ширина поля соответствует ширине экрана.

Ожидаемый результат: поля ввода и вывода расположены горизонтально, поле ввода находится левее поля вывода, поле трансляции ниже и по ширине экрана.

Видимый результат: полностью совпадает с ожидаемым

Тестируемая версия продукта:1.0.0

Резюме: Тест пройден

**Тесты для тестирования подсистемы «Лексический анализатор»**

***Тест TEST\_LA\_001***

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

Описание теста: Лексический анализатор принимает на вход строку с текстом кода на C++. Необходимо проверить, что анализатор корректно распознает ключевые слова, идентификаторы, разделители и числовые значения и преобразовывает их в соответствующие токены

Ожидаемый результат: Лексический анализатор возвращает список токенов, соответствующий входному коду. Каждый токен представлен в формате <тип, значение, номер строки, номер столбца>. Ошибки не возникают.

Видимый результат: полностью совпадает с ожидаемым

Тестируемая версия продукта:1.0.0

Резюме: Тест пройден

***Тест TEST\_LA\_002***

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

Описание теста: Лексический анализатор принимает на вход строку с текстом кода на C++ в котором содержится недопустимый символ «#». Анализатор должен выявить ошибку и сообщить о ней пользователю в окне протокола трансляции.

Ожидаемый результат: Лексический анализатор выявляет неизвестный символ. Выдается сообщение об ошибке. Ошибка отображается в отдельном окне пользовательского интерфейса. Анализатор прекращает дальнейшую обработку ввода после обнаружения ошибки

Видимый результат: полностью совпадает с ожидаемым

Тестируемая версия продукта: 1.0.0

Резюме: Тест пройден

**Тесты для тестирования подсистемы «Синтаксический анализатор»**

**Тест TEST\_SA\_001**

Тестируемые требования: REQ\_UI\_001, REQ\_UI\_002, REQ\_LA\_001, REQ\_SA\_001.

Описание теста:

1. нажать на поле ввода исходного кода на языке С++;
2. ввести код с ошибкой введения неожидаемого типа токена

int int main() {

while (true) {

break;

}

}

1. нажать на кнопку «Выполнить трансляцию».

Ожидаемый результат: *Окно протокола работы* содержит сообщение о синтаксической ошибке вида SyntaxError: Expected identifier after type keyword.

Видимый результат: Полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

**Тест TEST\_SA\_002**

Тестируемые требования: REQ\_UI\_001, REQ\_LA\_001, REQ\_SA\_001.

Описание теста:

1. нажать на поле ввода исходного кода на языке С++;
2. ввести код без синтаксических ошибок

int main() {

int y = 10;

return y;

}

1. нажать на кнопку «Выполнить трансляцию».

Ожидаемый результат: *Окно протокола работы* содержит список токенов и дерево разбора.

Окно протокола работы:

*[Token(type='NEWLINE', value='\n', line=2, column=1), Token(type='KEYWORD', value='int', line=2, column=5), Token(type='IDENTIFIER', value='main', line=2, column=9), Token(type='SEPARATOR', value='(', line=2, column=13), Token(type='SEPARATOR', value=')', line=2, column=14), Token(type='SEPARATOR', value='{', line=2, column=16), Token(type='NEWLINE', value='\n', line=3, column=17), Token(type='KEYWORD', value='int', line=3, column=9), Token(type='IDENTIFIER', value='y', line=3, column=13), Token(type='OPERATOR', value='=', line=3, column=15), Token(type='NUMBER', value='10', line=3, column=17), Token(type='SEPARATOR', value=';', line=3, column=19), Token(type='NEWLINE', value='\n', line=4, column=20), Token(type='KEYWORD', value='return', line=4, column=9), Token(type='IDENTIFIER', value='y', line=4, column=16), Token(type='SEPARATOR', value=';', line=4, column=17), Token(type='NEWLINE', value='\n', line=5, column=18), Token(type='SEPARATOR', value='}', line=5, column=5), Token(type='NEWLINE', value='\n', line=6, column=6)]*

*AST Output:*

*Statements:*

*Function: int main*

*Parameters:*

*Body:*

*Statements:*

*Variable Declaration: int y*

*Assigned value:*

*Value: 10*

*Return:*

*Unknown node type: <class 'nodes.VariableUsageNode'>*

Видимый результат: Полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

**Тесты для тестирования подсистемы «Семантический анализатор»**

**Тест TEST\_SemA\_001**

Тестируемые требования: REQ\_UI\_001, REQ\_UI\_002, REQ\_LA\_001, REQ\_SA\_001, REQ\_SemA\_001.

Описание теста:

1. нажать на поле ввода исходного кода на языке С++;
2. ввести код с семантической ошибкой

int main() {

int x;

      y = 10;

return 0;

}

1. нажать на кнопку «Выполнить трансляцию».

Ожидаемый результат: *Окно протокола работы* содержит список токенов, дерево разбора, результат работы семантического анализатора - «Семантическая ошибка: Переменная ‘у’ не определена».

Видимый результат: Полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

**Тест TEST\_SemA\_002**

Тестируемые требования: REQ\_UI\_001, REQ\_LA\_001, REQ\_SA\_001, REQ\_SemA\_001.

Описание теста:

1. нажать на поле ввода исходного кода на языке С++;
2. ввести код без семантических, лексических и синтаксических ошибок

int main() {

int x;

      x = 10;

return 0;

}

1. нажать на кнопку «Выполнить трансляцию».

Ожидаемый результат: *Окно протокола работы* содержит список токенов, дерево разбора, результат работы семантического анализатора - «Семантический анализ прошел успешно!».

Видимый результат: Полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

**Тесты для тестирования подсистемы «Генератор кода»**

**Тест TEST\_CG\_001**

Тестируемые требования: REQ\_UI\_001, REQ\_LA\_001, REQ\_SA\_001, REQ\_SemA\_001, REQ\_CG\_001.

Описание теста:

1. нажать на поле ввода исходного кода на языке С++;
2. ввести код без лексических, синтаксических и семантических ошибок

int add(int a, int b) {

return a + b;

}

void greet() {

std::cout << "Hi!";

}

1. нажать на кнопку «Выполнить трансляцию».

Ожидаемый результат: *Окно протокола работы* содержит список токенов, дерево разбора, результат работы семантического анализатора, *Поле вывода транслированного кода на языке Python* содержит код на языке Python.

*Поле вывода транслированного кода на языке Python*:

def add(a, b):

return a + b

def greet():

print("Hi!")

Видимый результат: Полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден**.**

**Тесты для тестирования системы в целом**

***Тест TEST\_SYS\_001***

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, REQ\_UI\_009, REQ\_LA\_001, REQ\_LA\_002, REQ\_SA\_001, REQ\_SemA\_001, REQ\_CG\_001*.*

Описание теста: запустить программу, ввести исходный код на языке C++, нажать кнопку «Транслировать».

Ожидаемый результат: запустится окно интерфейса, транслируемый код на Python будет выведен в правом окне, ошибок не возникнет.

Видимый результат: Полностью совпадает с ожидаемым

Тестируемая версия продукта: 1.0.0

Резюме: Тест пройден

**Список литературы**

1. PEP 8 – руководство по написанию кода на Python [Электронный ресурс]. – Режим доступа: [https://pythonworld.ru/](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html) (дата обращения: 15.01.2025)
2. На блэкборд