

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. Б9119-09.03.04прогин  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Колпакова А. И.

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

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

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

Руководитель:  
асисстент департамента ПИиИИ,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Иваненко Ю. С.

г. Владивосток  
2025

Оглавление

[Введение 3](#_Toc184167066)

[1. Разработка плана проекта 4](#_Toc184167067)

[2. Разработка регламента проведения инспекции 5](#_Toc184167068)

[3. Разработка модели состояний задач 8](#_Toc184167069)

[4. Разработка презентации проекта 10](#_Toc184167070)

[5. Разработка требований к проекту 12](#_Toc184167071)

[6. Разработка архитектуры проекта 15](#_Toc184167072)

**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.

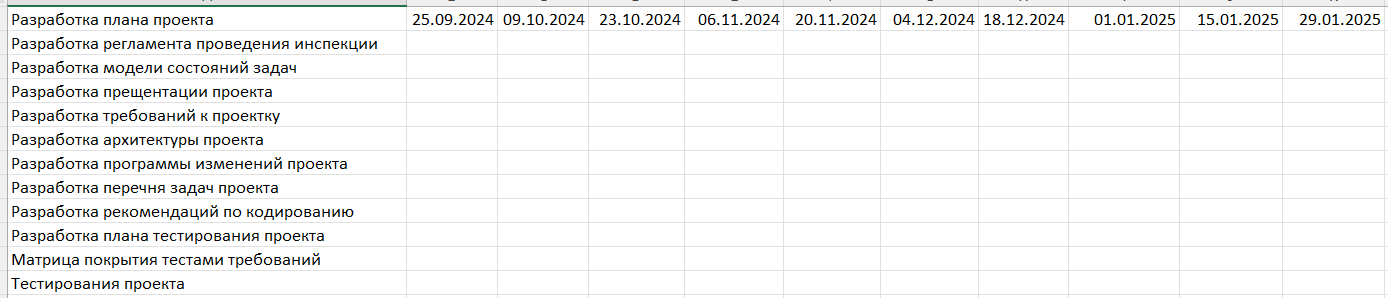


Рисунок 1 – Календарный план работы над проектом

# **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.

**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 серьёзности 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):

IPR = (Количество инспекторов \* Размер продукта) / Общее время подготовки

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 начнёт её 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 сотрудникомразработчиком, 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 отчёта 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 содержит состав команды (рисунок 3)



Рисунок 3 - Первый слайд

Второй слайд содержит краткое описание проекта (рисунок 4)



Рисунок 4 - второй слайд

На третьем слайде представлены преимущества проекта (рисунок 5)



Рисунок 5 - третий слайд

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

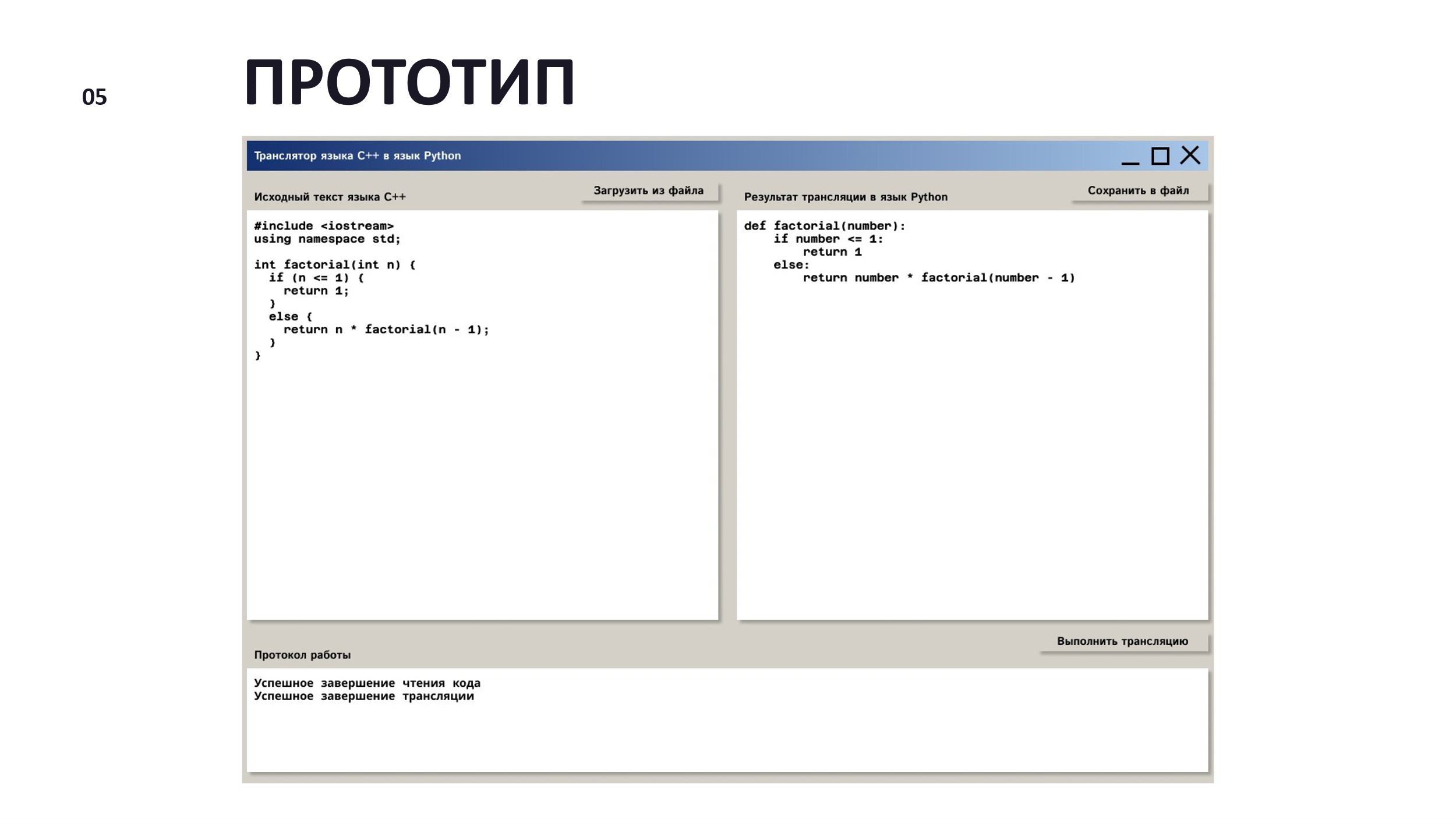


Рисунок 6 - четвертый слайд

На пятом слайде представлен основной функционал проекта (рисунок 7)

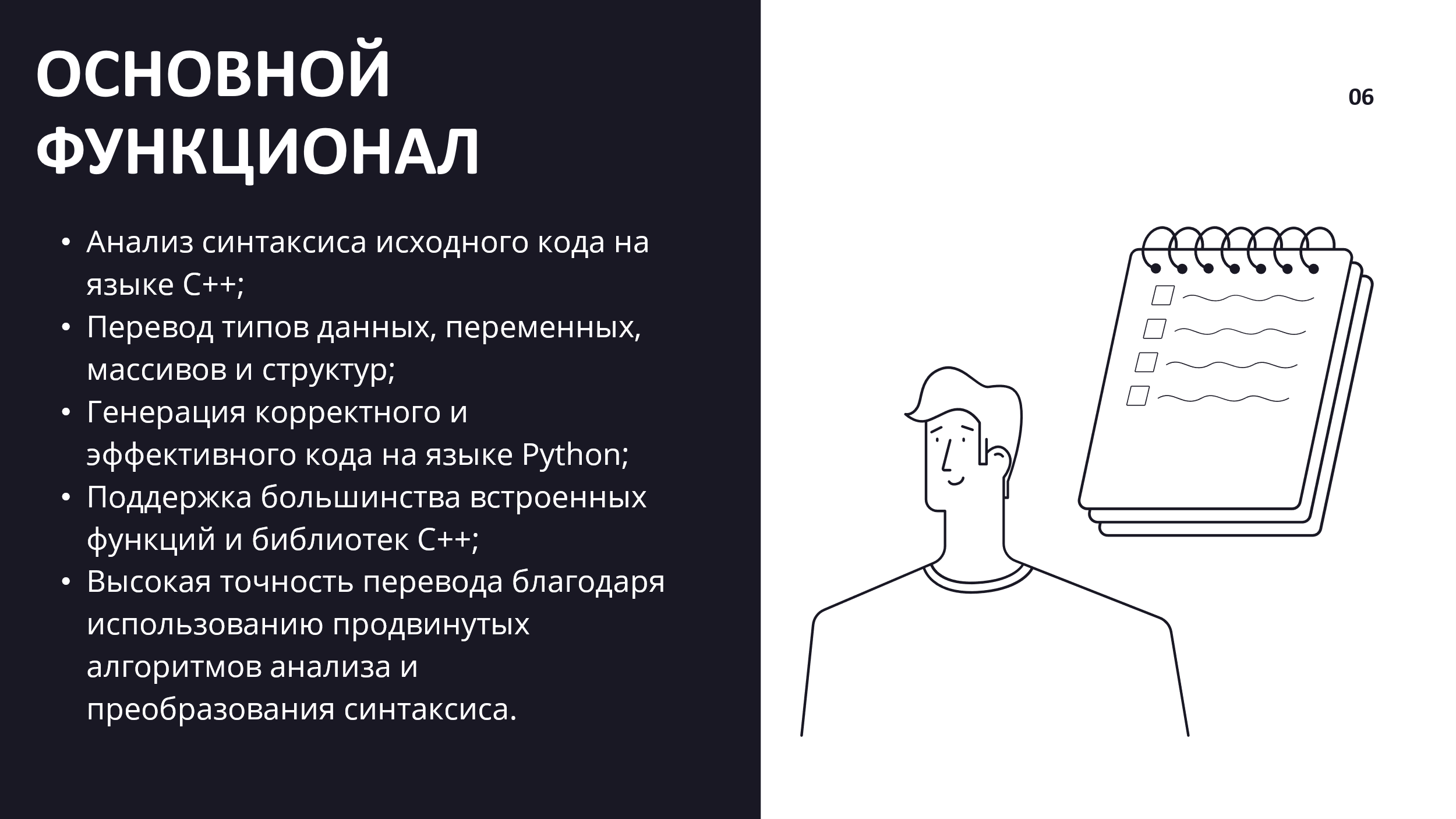


Рисунок 7 - пятый слайд

# **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. 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
* Кнопка трансляции языка
* Кнопка загрузки файла
* Кнопка выгрузки файла
* Окно протокола работы

Требование REQ\_UI\_002

При допущении пользователем синтаксической 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 должна иметь вид прямоугольника с подписью «Выполнить трансляцию» (См. рисунок 7).



Рисунок 7 – 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 должна иметь вид прямоугольника с подписью «Загрузить из файла» (См. рисунок 8).



Рисунок 8 – Кнопка выгрузки

Требование REQ\_UI\_006

Кнопка загрузки в файл должна иметь вид прямоугольника с подписью «Сохранить в файл» (См. рисунок 9).



Рисунок 9 – 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 строится абстрактное синтаксическое дерево разбора, если синтаксических ошибок нет, иначе – выдаётся ошибка.

**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 10 представлена диаграмма потоков данных

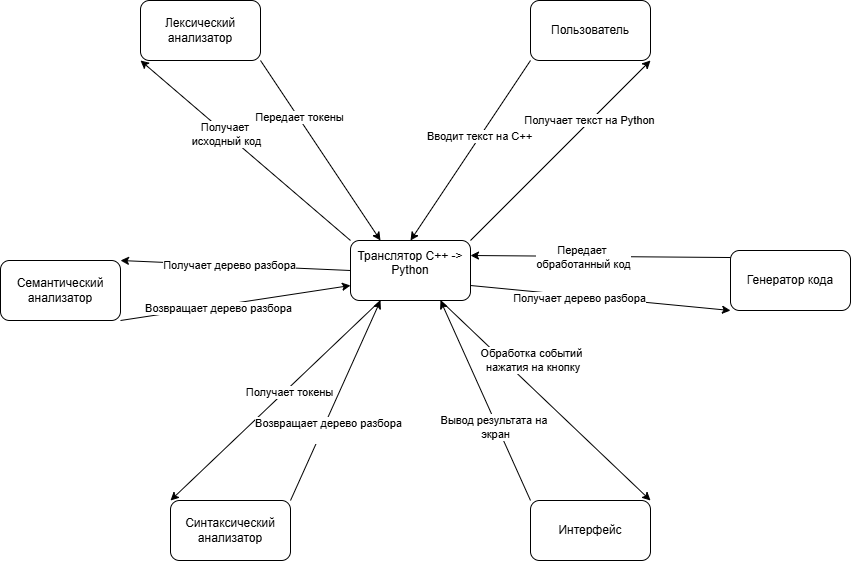


Рисунок 10 - Контекстная диаграмма

**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 11 представлена диаграмма User Flow

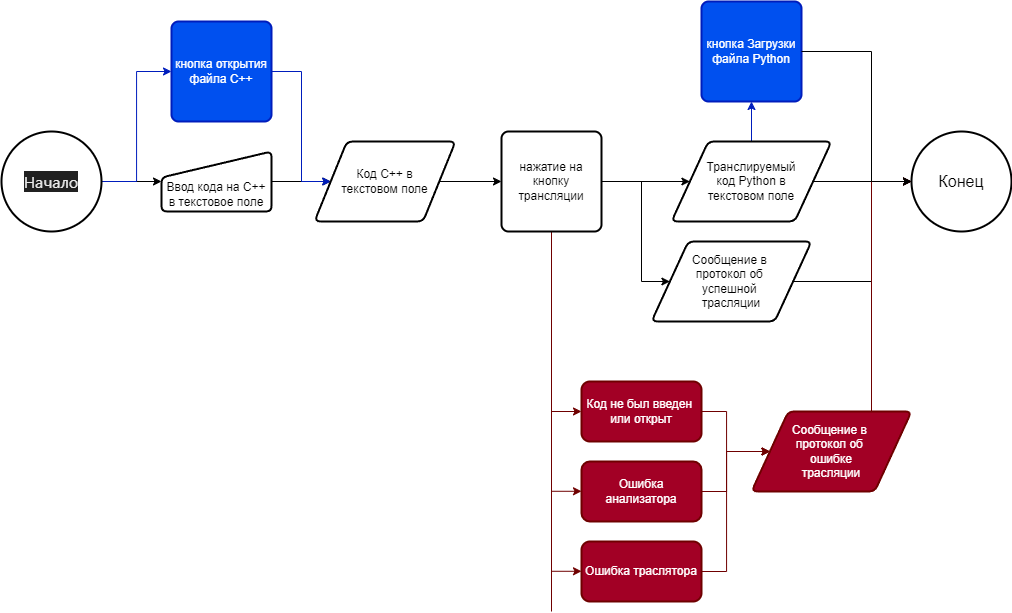


Рисунок 11 - User Flow