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**

КОЛЛЕКТИВНАЯ РАЗРАБОТКА ПРОЕКТА ТРАНСЛЯТОРА, ПЕРЕВОДЯЩЕГО ПОДМНОЖЕСТВО ЯЗЫКА C++ В ЭКВИВАЛЕНТНОЕ ПОДМНОЖЕСТВО ЯЗЫКА PASCAL

**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. Б9120-09.03.04

\_\_\_\_\_\_\_\_\_\_\_ Гончарук Д. М.

\_\_\_\_\_\_\_\_\_\_\_ Решетнев Н. Я.

\_\_\_\_\_\_\_\_\_\_\_ Назаров М. С.

\_\_\_\_\_\_\_\_\_\_\_ Курбанов М. М.

Руководитель: Старший преподаватель Департамента ПИиИИ

Иваненко Ю. С.

\_\_\_\_\_\_\_\_\_\_\_\_ «\_\_\_» \_\_\_\_\_\_\_\_ 2024 г.

(подпись)

Защищён с оценкой \_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

(подпись) И.О. Фамилия

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2024 г.

г. Владивосток

2024 г.

**Оглавление**

[Введение 3](#_Toc161925070)

[1 Регламент проведения инспекции 4](#_Toc161925071)

[1.1 Процедура организации и проведения формальной инспекции 4](#_Toc161925072)

[1.2 Процедура организации и проведения неформальной инспекции 4](#_Toc161925073)

[1.3 Роли участников инспекции 5](#_Toc161925074)

[1.4 Порядок организации инспекции 5](#_Toc161925075)

[1.5 Порядок подготовки к инспекции 6](#_Toc161925076)

[1.6 Перечень статусов и степени важности замечаний 6](#_Toc161925077)

[1.7 Метрики, характеризующие эффективность инспекций 7](#_Toc161925078)

[2 Модель состояний задач 9](#_Toc161925079)

[2.1 Состояния задач 9](#_Toc161925080)

[2.2 Создание новой задачи 9](#_Toc161925081)

[2.3 Перехода задачи из состояния в состояние 9](#_Toc161925082)

[3 План проекта 10](#_Toc161925083)

[3.1 Проектная команда 10](#_Toc161925084)

[3.2 График разработки 11](#_Toc161925085)

[4 Требования к проекту 12](#_Toc161925086)

[4.1 История изменений документа 12](#_Toc161925087)

[4.2 Подсистемы проекта 12](#_Toc161925088)

[4.3 Требования к подсистеме «Пользовательский интерфейс» 12](#_Toc161925089)

[4.4 Требования к подсистеме «Лексический анализатор» 13](#_Toc161925090)

[4.5 Требования к подсистеме «Синтаксический анализатор» 14](#_Toc161925091)

[4.6 Требования к подсистеме «Семантический анализатор» 15](#_Toc161925092)

[5 Архитектура проекта 16](#_Toc161925093)

[5.1 История изменений документа 16](#_Toc161925094)

[5.2 Архитектурно-контекстная диаграмма системы 17](#_Toc161925095)

[5.2.1 АКД подсистемы «Лексический анализатор» 18](#_Toc161925096)

[5.2.2 АКД «Синтаксический анализатор» 19](#_Toc161925097)

[5.2.3 АКД подсистемы «Семантический анализатор» 20](#_Toc161925098)

[6. Разработка плана тестирования проекта 21](#_Toc161925099)

[Заключение 24](#_Toc161925100)

[Список литературы 25](#_Toc161925101)

Введение

Процесс промышленной разработки информационных систем охватывает различные этапы, начиная с создания плана проекта и завершаясь тестированием для чего требуется сотрудничество многочисленных специалистов различных направлений. Также необходимы унифицированные методы взаимодействия между ними, которые позволяют распределить обязанности среди членов команды в соответствии с их экспертизой.

Исходя из описанного, необходимо применять конкретные технологии коллективной разработки для увеличения эффективности работы в группе и соответствия конечного продукта установленным требованиям.

В данной работе рассматривается проблема коллективной разработки программного средства "Транслятор из C++ в Pascal", а также составление технической документации для данного инструмента.

Таким образом, основной целью данной курсовой работы является создание программного средства "Транслятор из C++ в Pascal" с использованием методов коллективной промышленной разработки.

**Задачи курсовой работы:**

* Разработать план проекта;
* Разработать регламент проведения инспекции;
* Разработать модель состояний задач;
* Разработать требования к проекту;
* Разработать архитектуру проекта;
* Разработать программу измерений проекта;
* Разработать перечень задач проекта;
* Разработать рекомендации по кодированию;
* Разработать план тестирования проекта.

1 Регламент проведения инспекции

1.1 Процедура организации и проведения формальной инспекции

Формальные инспекции направлены на оценку завершенных частей рабочих продуктов, где требуется проверка документации по дизайну или функциональности кода с соответствующими тестами для верификации. Процедура формальной инспекции предполагает участие трех членов команды.

Этапы формальной инспекции включают следующие шаги:

Планирование инспекции;

Назначение инспекции;

Собрания по инспекции;

Завершение инспекции (распространение результатов, проверка исправления недостатков в рабочем продукте).

На этапе "Назначение инспекции" обязанности автора включают инициирование формальной инспекции рабочего продукта, для чего необходимо уведомить руководителя проекта.

Председатель обязан удостовериться, что рабочий продукт соответствует установленным критериям готовности к формальной инспекции.

Инспекторы, в свою очередь, должны независимо от других участников инспекции изучить предоставленный для инспекции рабочий продукт и заполнить соответствующие разделы протокола подготовки к формальной инспекции.

1.2 Процедура организации и проведения неформальной инспекции

Неформальные инспекции подвергаются маленькие части рабочих продуктов, где необходимо проверить работоспособность кода.

В процедуре формальной инспекции могут принимать участие 2 члена команды.

**Этапы неформальной инспекции:**

1. Назначение инспекции;
2. Собрания по инспекции;
3. Завершение инспекции (распространение результатов, проверка исправления недостатков в рабочем продукте).

1.3 Роли участников инспекции

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

**В роли могут быть: Гончарук Данил;**

1. **Председатель (Moderator)** - ответственный сотрудник, выполняющий роль председателя инспекции.

**В роли могут быть: Решетнев Никита;**

1. **Ведущий (Presenter)** - сотрудник, представляющий рабочий продукт инспекторам.

**В роли могут быть: Курбанов Марат;**

1. **Инспектор (Inspector)** - сотрудник, ответственный за эффективную

проверку инспектируемого рабочего продукта.

**В роли могут быть: Назаров Максим, Курбанов Марат;**

1. **Инженер по качеству** – сотрудник, ответственный за анализ метрик после инспекции.

**В роли могут быть: Решетнев Никита, Назаров Максим.**

1.4 Порядок организации инспекции

Этапы проведения инспекции включают следующие шаги:

**Планирование инспекции:** На этом этапе коллективно определяются дата, время, формат (очный или заочный) и платформа (при заочной инспекции) для проведения инспекции.

**Подготовка к инспекции:** Инспектор самостоятельно изучает предоставленный для инспекции рабочий продукт, используя свой опыт и соответствующие стандарты.

**Собрание по инспекции:** На собрании обсуждаются замечания и рекомендации, вынесенные инспектором по рабочему продукту. Обязательно присутствие инспектора и автора рабочего продукта, который подлежит инспекции. Участие остальных членов команды разработки опционально.

**Завершение инспекции:** Если рабочий продукт требует корректировок, автор фиксирует все замечания и рекомендации инспектора, разрабатывает план последующих работ и согласовывает его с инспектором. В случае, если доработки не требуются, инспектор подтверждает слияние рабочей ветки в основную ветку ("master").

1.5 Порядок подготовки к инспекции

На этапе назначения инспекции устанавливаются время и дата проведения, исходя из количества участников. Выбираются оптимальные сроки, удовлетворяющие всем членам команды, в пределах 2 дней после планирования инспекции. Важно отметить, что даты инспекций не назначаются на субботу, воскресенье или праздничные дни.

1.6 Перечень статусов и степени важности замечаний

**Дефект (Defect):** Проблема, выявленная на этапе, отличном от этапа внесения.

**Ошибка (Error):** Проблема, обнаруженная на том же этапе, на котором была выявлена.

**Комментарий (Comment):** Наблюдение, предложение, рекомендация или улучшение, предложенное для будущего выпуска рабочего продукта, или вопрос, требующий разъяснения.

**Замечание для исследования (Investigate):** Проблема, природа которой не может быть определена на собрании и требует дополнительного исследования.

Допустимые степени серьезности замечаний включают:

**Особо важная (Major):** Проблема, требующая срочного внимания и решения.

**Средняя (Moderate):** Проблема, которая влияет на работоспособность продукта, но может быть решена без немедленных мер.

**Мелкая, незначительная (Minor):** Небольшая проблема, оказывающая ограниченное воздействие на функциональность.

**Другие (Other):** Дополнительные категории для классификации замечаний, не подпадающих под основные степени серьезности.

**Порядок верификации учёта замечаний**

* Особо важная (Major) – неисправность или ошибки в программном продукте более чем 50%;
* Средняя (Moderate) - неисправность или ошибки в программном продукте более чем 20 %;
* Мелкая, незначительная (Minor) - ошибки в программном продукте более чем 10%;
* Другие (Other) – в иных случаях.

1.7 Метрики, характеризующие эффективность инспекций

Метрики по инспекциям собираются инженером по качеству **Решетневым Никитой** после каждой инспекции и анализируются с помощью контрольных карт.

**Метрики по инспекциям:**

* **Inspection Fault Density (IFD)**

IFD = (Количество найденных ошибок / Размер рабочего продукта)

Стратегическая цель метрики – повысить качество разрабатываемого ПО.

Изучаемый объект метрики – инспекция, измеряемый атрибут – плотность найденных в ходе инспекции ошибок.

Единица измерения – ошибка / <страница, требование, LOC, тест>.

* **Inspection Preparation Rate (IPR)**

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

1. Модель состояний задач

2.1 Состояния задач

**Перечень возможных состояний задачи и их интерпретация:**

* New (Новая задача). Аналитиками создана новая задача;
* Task assignment (Задача назначена). Аналитик назначает задачу на конкретного автора;
* Coding (Задача в разработке). Задача в разработке у автора;
* Сheck or testing (Проверка или тестирование). Задача находится в состоянии проверки или тестируется;
* Closed (Задача выполнена). Задача закрыта.

2.2 Создание новой задачи

**Правила создания новой задачи:**

Список сотрудников, которые могут создавать новые задачи: **Гончарук Данил, Решетнев Никита.**

Новые задачи могут создаваться в случае выдвижения новых требований клиентом (Артемьевой Ириной Леонидовной).

2.3 Перехода задачи из состояния в состояние

**Правила перехода задачи из состояния в состояние:**

* **New** →**Task assignment**. Переход в случае назначения аналитиком автора на задачу;
* **Task assignment** → **Coding.** Переход, если назначенный автор начал работать над задачей
* **Coding** → **Сheck or testing.** Переход в случае, если автор закончил работу над задачей и можно провести тестирование.
* **Сheck or testing** → **Closed.** Переход в случае, если задача протестирована и 80% тестов прошли удачно.

1. План проекта

Документ, описывающий план проекта, предоставляет детальную информацию о проекте, включая данные об участниках, поставленных задачах и установленных сроках. Этот документ представляет собой конечный результат этапа планирования, утверждается перед началом любых работ и становится основным и надежным источником сведений о предстоящем проекте.

3.1 Проектная команда

Таблица 1. Проектная команда

|  |  |
| --- | --- |
| **Проектная команда** | **Члены команды** |
| Тимлид | Гончарук Данил |
| Кодер 1 | Гончарук Данил |
| Кодер 2 | Решетнев Никита |
| Инженер по качеству | Курбанов Марат |
| Технический писатель | Назаров Максим |
| Тестировщик 1 | Назаров Максим |
| Тестировщик 2 | Курбанов Марат |

3.2 График разработки

Таблица 2. График разработки

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Дата**  **Разработка** | **14.9.2023-21.9.2023** | **22.9.2023-30.9.2023** | **1.10.2023-12.10.2023** | **13.10.2023-21.10.2023** | **22.10.2023-31.10.2023** | **1.11.2023-15.11.2023** | **16.11.2023-31.11.2023** | **1.12.2023-10.12.2023** | **11.12.2023-20.12.2023** | **21.12.2023-31.12.2023** | **1.1.2024-10.1.2024** | **11.1.2024-20.1.2024** | **21.1.2024-10.1.2024** |
| **План проекта программного Обеспечения** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Спецификация маркетинговых требований** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Спецификация требований к программному обеспечению** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Спецификация проектирования Программного Обеспечения** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Разработка грамматики для лексического анализатора** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Разработка лексического анализатора** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Тестирование лексического анализатора** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Разработка грамматики для синтаксического анализатора** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Разработка синтаксического анализатора** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Тестирование синтаксического анализатора** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Разработка семантического анализатора** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Тестирование семантического анализатора** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Составление документации** |  |  |  |  |  |  |  |  |  |  |  |  |  |

1. Требования к проекту

4.1 История изменений документа

Таблица 3. История изменений

|  |  |  |
| --- | --- | --- |
| **Дата** | **Автор** | **Внесённые изменения** |
| 22.09.2023 | Назаров Максим | Исходная версия требований |
| 25.10.2023 | Курбанов Марат | Написаны главы 1, 2 |
| 12.12.2023 | Курбанов Марат | Правлена глава 2, написаны главы 3, 4. |
| 29.12.2023 | Назаров Максим | Определены основные метрики и требования качества. Написаны главы 5, 6. |
| 19.03.2024 | Гончарук Данил | Исправлены UML для АКД всех подсистем. |

Программный продукт «Транслятор C++ на Pascal» предназначен для компиляции программ, написанных на языке С++, в программу на языке Pascal.

4.2 Подсистемы проекта

Программный продукт «Транслятор C++ на Pascal» состоит из следующих подсистем:

1. Пользовательский интерфейс;
2. Подсистема лексического анализатора;
3. Подсистема синтаксического анализатора;
4. Подсистема семантического анализатора.

4.3 Требования к подсистеме «Пользовательский интерфейс»

**Требование REQ\_UI\_001.1**

Текст компилированного текста должен отображаться для исходного кода на С++ (Рис. X).

**Требование REQ\_UI\_001.2**

Текст скомпилированного текста должен отображаться для целевого кода на Pascal (Рис. 1).

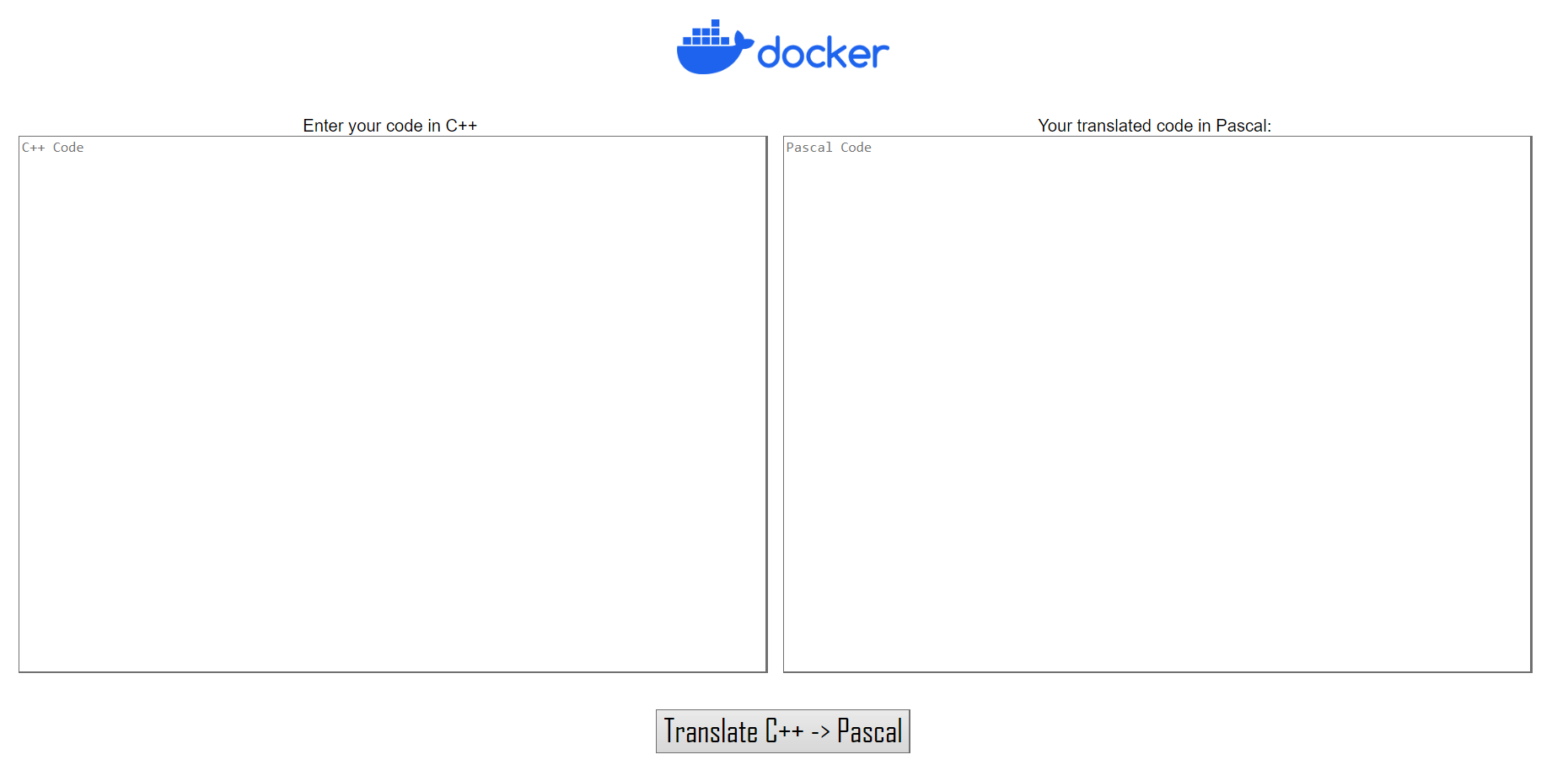


Рисунок 1 – Пользовательский интерфейс.

**Требование REQ\_UI\_002**

На рисунке X отображены общие требования к интерфейсу.

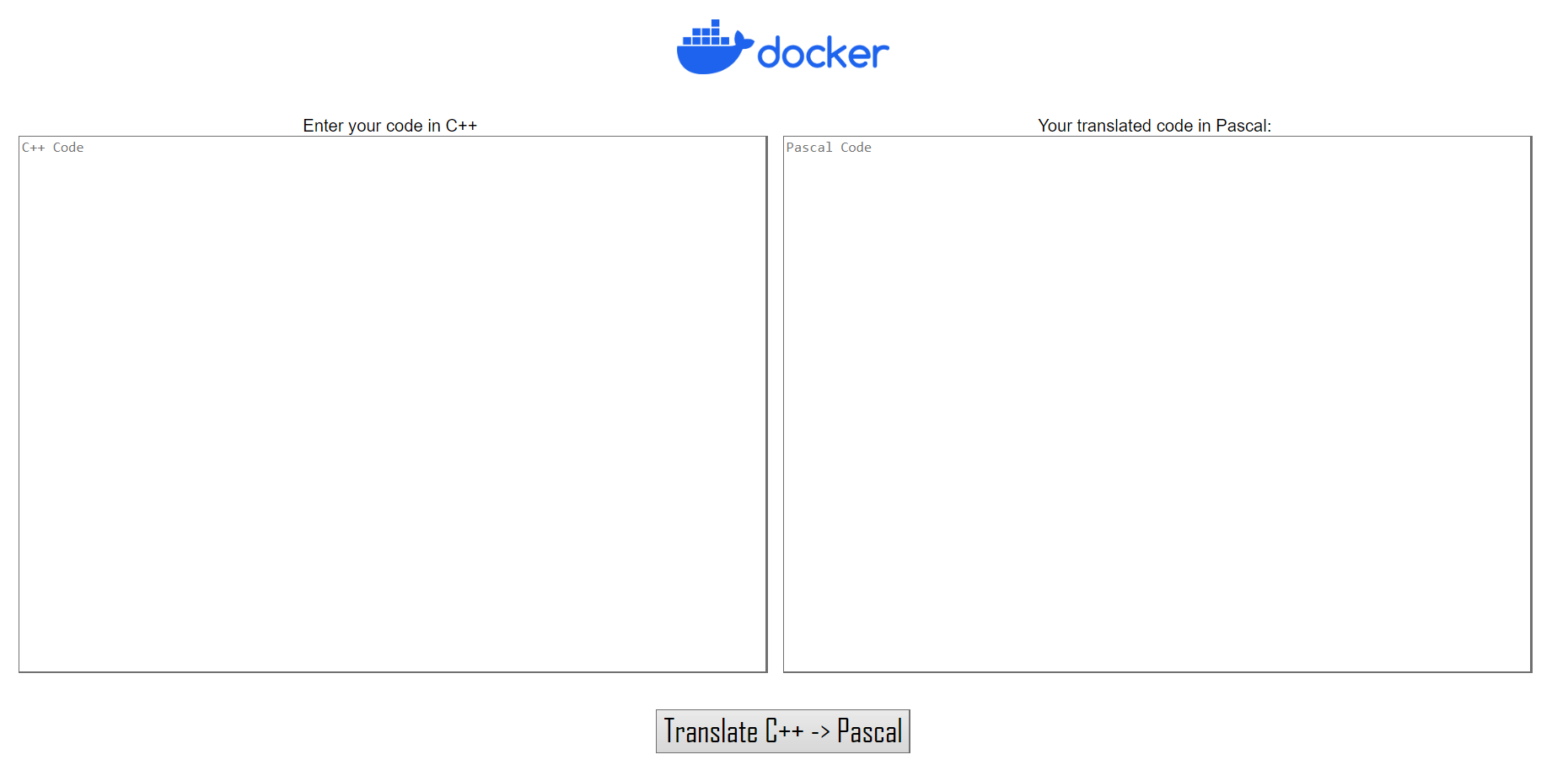


Рисунок 2 – Пользовательский интерфейс.

**Требование REQ\_UI\_003**

Если для исходного кода был введен код с ошибкой или код на другом язык, то для сообщений компилятора будет выводиться сообщения ошибке.

4.4 Требования к подсистеме «Лексический анализатор»

**Требование REQ\_LA\_001**

Подсистема лексический анализатор должна подготовить входную последовательность из для исходного кода для подсистемы синтаксический анализатор.

**Требование REQ\_LA\_002**

Подсистема должна считывать входной поток символов для исходного кода.

**Требование REQ\_LA\_003**

Подсистема должна распознавать лексемы в контексте грамматики путем классификации по типам:

**Требование REQ\_LA\_004**

В случае, когда подсистема не может идентифицировать лексему, она должна классифицировать ее как специальный токен-ошибку.

**Требование REQ\_LA\_005**

Каждая лексема должна представлять собой структуру, которая содержит идентификатор лексемы, последовательность символов лексем, выделанной из входного потока и порядковый номер лексемы.

**Требование REQ\_LA\_006**

Подсистема лексического анализатора на вход в подсистему синтаксического анализатора подает таблицу из лексем и их идентификаторов.

4.5 Требования к подсистеме «Синтаксический анализатор»

**Требование REQ\_STA \_001**

Подсистема синтаксический анализатор должна проверить входную последовательность из лексического анализатора и подать проверенную последовательность на вход семантическому анализатору.

**Требование REQ STA \_002**

Подсистема по средствам входных данных из лексического анализатор должна построить дерево разбора.

**Требование REQ\_STA \_003**

С помощью дерева разбора подсистема должна проверить синтаксическую корректность кода.

**Требование REQ\_STA \_004**

В случае нахождения ошибки при построении дерева разбора система должна вывести сообщения об ошибках для сообщения компилятора.

**Требование REQ\_STA \_005**

Подсистема синтаксического анализатора на вход в подсистему семантического анализатора подает дерево разбора в случае успешного его построения.

4.6 Требования к подсистеме «Семантический анализатор»

**Требование REQ\_SMA \_001**

Подсистема семантический анализатор должна проверить входную последовательность из синтаксического анализатора и подать проверенную последовательность на вход генератору кода.

**Требование REQ\_SMA \_002**

Подсистема по средствам входных данных из синтаксического анализатора должна провести проверку контекстных условий.

**Требование REQ\_SMA \_003**

В случае нахождения ошибки при проверки контекстных условий система должна вывести сообщения об ошибках для сообщения компилятора.

**Требование REQ\_SMA \_004**

Подсистема семантического анализатора после успешной проверки условий должна подать проверенную последовательность на генератор кода.

1. Архитектура проекта
   1. История изменений документа

Таблица 4. История изменений

|  |  |  |
| --- | --- | --- |
| **Дата** | **Автор** | **Внесённые изменения** |
| 27.12.2023 | Гончарук Данил | Исходная версия архитектуры проекта |
| 07.01.2024 | Гончарук Данил | Изменение архитектуры лексического анализатора. |
| 10.02.2024 | Гончарук Данил | Изменение архитектуры семантического анализатора. |
| 11.02.2024 | Гончарук Данил | Изменение правил кодогенерации. |
| 21.03.2024 | Решетнев Никита | Адаптация архитектуры проекта под web-приложение. |

Программный продукт «Транслятор C++ на Pascal» предназначен для компиляции программ, написанных на языке С++, в программу на языке Pascal.

Настоящий документ содержит спецификации дизайна, включающие в себя:

1. АКД системы;
2. АКД подсистема лексического анализатора;
3. АКД подсистема синтаксического анализатора;
4. АКД подсистема семантического анализатора

5.2 Архитектурно-контекстная диаграмма системы

На рисунке 3 представлена архитектурно контекстная диаграмма системы «Транслятор C++ на Pascal».

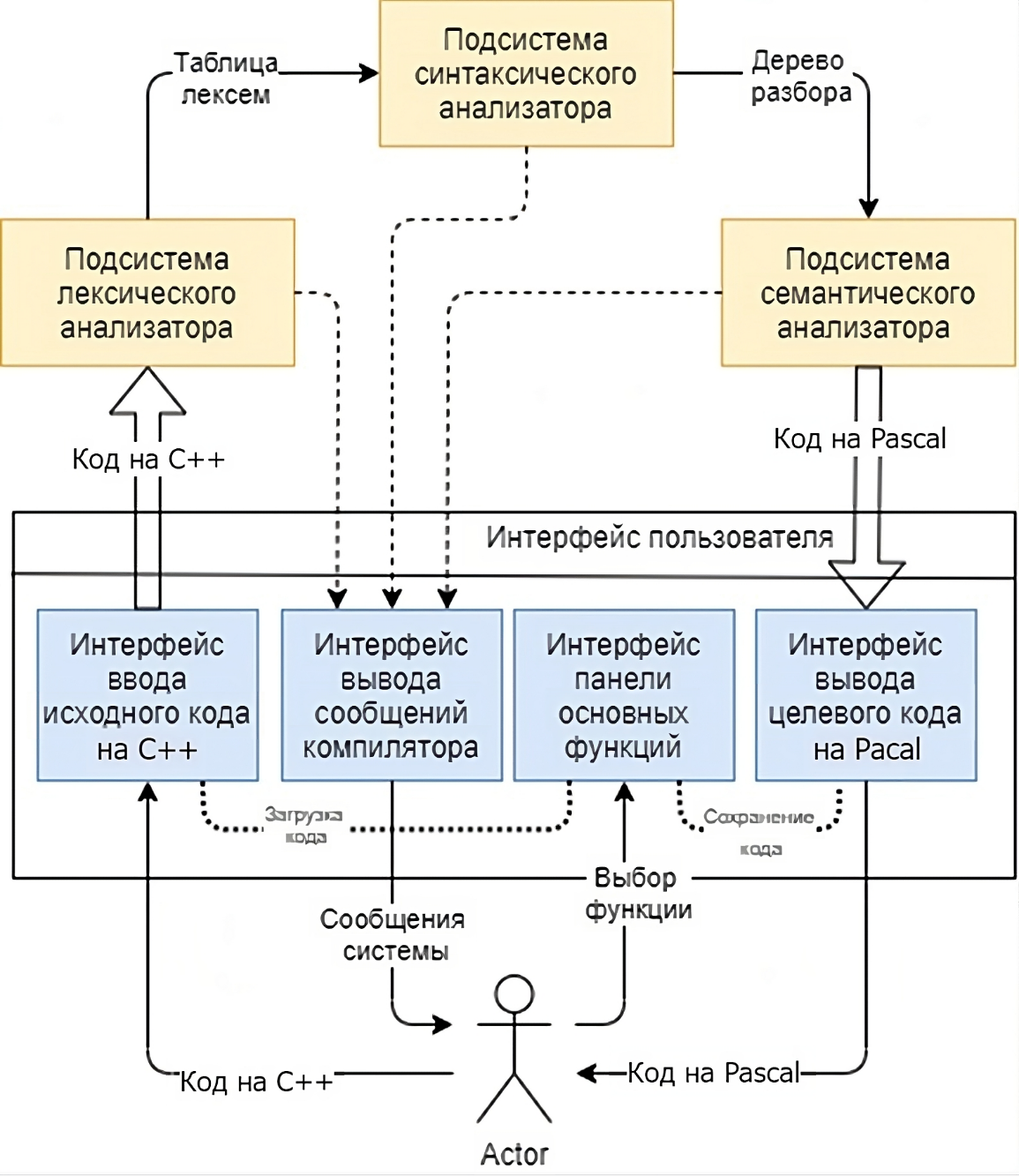


Рисунок 3 – АКД системы

5.2.1 АКД подсистемы «Лексический анализатор»

На рисунке 4 представлена архитектурно контекстная диаграмма подсистемы лексического анализатора.

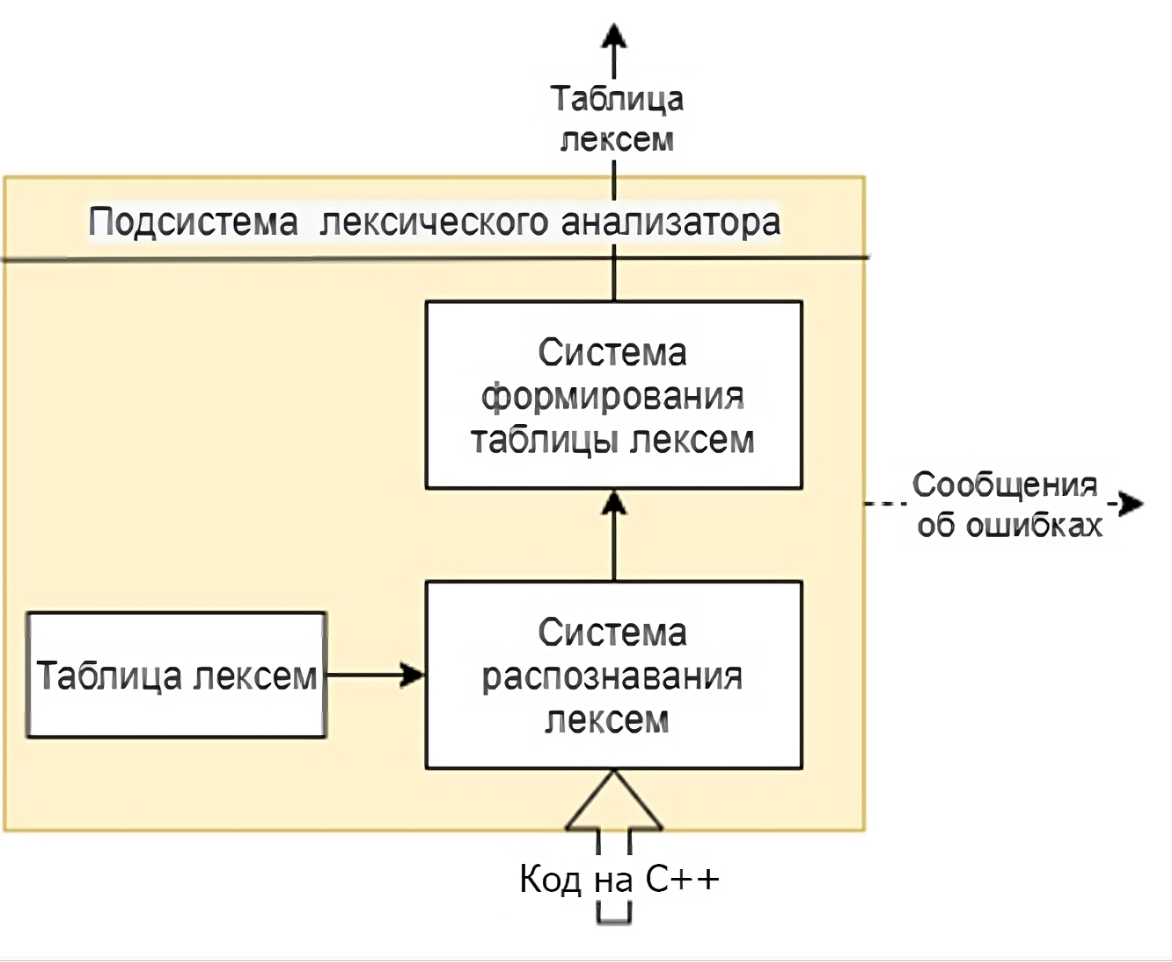


Рисунок 4 – АКД подсистемы лексического анализатора

5.2.2 АКД «Синтаксический анализатор»

На рисунке 5 представлена архитектурно контекстная диаграмма подсистемы синтаксического анализатора.

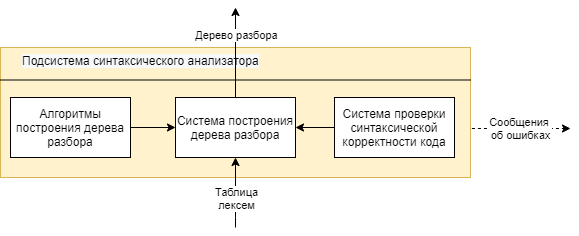


Рисунок 5 – АКД подсистемы синтаксического анализатора

5.2.3 АКД подсистемы «Семантический анализатор»

На рисунке 6 представлена архитектурно контекстная диаграмма подсистемы семантического анализатора.

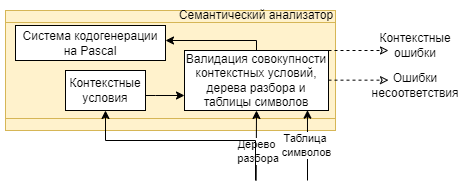


Рисунок 6 – АКД подсистемы семантического анализатора

6. Разработка плана тестирования проекта

**Тесты модуля «Пользовательский интерфейс»:**

**Тест TEST\_UI\_01**

Тестируемые требования: REQ\_UI\_001.1, REQ\_UI\_001.2

Описание теста: Текст компилированного кода должен отображаться для исходного кода на С++ и для кода на Pascal

Ожидаемый результат: отображение текста.

**Тест TEST\_UI\_02**

Тестируемые требования: REQ\_UI\_002

Описание теста: Соответствие общим требованиям интерфейса

Ожидаемый результат: все элементы интерфейса отображаются корректно

**Тест TEST\_UI\_03**

Тестируемые требования: REQ\_UI\_003

Описание теста: Если для исходного кода был введен код с ошибкой или код на другом язык, то для сообщений компилятора будет выводиться сообщения ошибке.

Ожидаемый результат: выводится сообщение об ошибке.

**Тесты для подсистемы «Модуль лексического анализа»**

**Тест TEST\_LX\_001**

Тестируемые требования: REQ\_LA\_001, REQ\_LA\_002, REQ\_LA\_003

Проверка разбиения кода С++ на токены, содержащие всю описанную в требованиях информацию, и их соответствие формату JSON. На вход модулю подаётся строку, содержащую код на языке С++. Производится попытка разбиения на токены и сохранение их в формате JSON.

**Тест TEST\_LX\_002**

Тестируемые требования: REQ\_LA\_004, REQ\_LA\_005, REQ\_LA\_006

Проверка работоспособности в случае нахождения лексемы, не попадающей в список выделенных, токен помечается ошибкой. При этом модуль не прекращает работу и формирует сообщение с описанием ошибки и доставляет его до интерфейса пользователя.

**Тесты для подсистемы «Модуль синтаксического анализа»**

**Тест TEST\_STA\_001**

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

REQ\_STA \_003, REQ\_STA \_005

Описание теста: Система синтаксический анализатор должна проверить входную последовательность из лексического анализатора и подать проверенную последовательность на вход семантическому анализатору

Ожидаемый результат: при проверке ошибок не выявлено. Семантический анализатор принял проверенную последовательность.

**Тест TEST\_STA\_002**

Тестируемые требования: REQ\_STA \_004

Описание теста: В случае нахождения ошибки при построении дерева разбора система должна вывести сообщения об ошибках для сообщения компилятора

Ожидаемый результат: выводится сообщение об ошибке

**Тесты для подсистемы «Модуль семантического анализа»**

**Тест TEST\_SMA \_001**

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

REQ\_SMA \_004

Описание теста: Подсистема семантический анализатор должна проверить входную последовательность из синтаксического анализатора и подать проверенную последовательность на вход генератору кода

Ожидаемый результат: при проверке ошибок не выявлено  
**Тест TEST\_SMA \_002**

Тестируемые требования: REQ\_SMA \_003

Описание теста: В случае нахождения ошибки при проверки контекстных условий система должна вывести сообщения об ошибках для сообщения компилятора.

Ожидаемый результат: выводится сообщение об ошибке

**Матрица покрытия тестами требований**

Была построена матрица покрытия тестами требований [Рисунок 10].

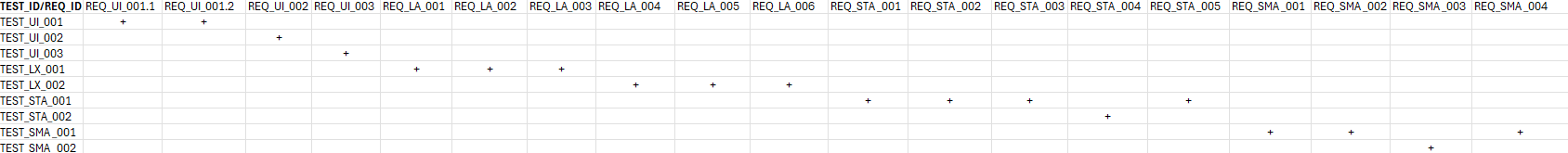


Рисунок 7 — Матрица покрытия тестами требований.

Заключение

В ходе курсовой работы были решены следующие задачи:

* Разработан регламент проведения инспекции;
* Разработана модель состояний задач;
* Разработан план проекта;
* Разработаны требования к проекту;
* Разработана архитектуру проекта;
* Разработана программа измерений проекта;
* Разработан перечень задач проекта;
* Разработаны рекомендации по кодированию;
* Разработан план тестирования проекта.

Список литературы

* 1. Ахо, А. Компиляторы: принципы, технологии и инструментарий компиляторов / И. В. Красиков. – М.: Вильямс, 1986. – 1184 с. ;
  2. Бруно, К. Л. LLVM: инфраструктура для разработки компиляторов / К. Л. Бруно, А. Рафаэль. – М.: ДМК Пресс, 2015. – 342 с.;
  3. Вирт, Н. Построение компиляторов / Н. Вирт. – М.: ДМК Пресс, 2010. – 192 с.;
  4. C++ documentation [Электронный ресурс]. URL: https://devdocs.io/cpp/ (дата обращения 20.02.2024).