  
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 ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИЙ КОЛЛЕКТИВНОЙ ПРОМЫШЛЕННОЙ РАЗРАБОТКИ**

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  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Бураков А.А.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Зенков М.А.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ильченко А.С.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Крючкова Н.Ю.  
Руководитель:  
профессор департамента ПИиИИ  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Гриняк В.М., д.т.н., профессор

г. Владивосток  
2023

Оглавление

[Введение 3](#_Toc124874819)

[1 Разработка плана проекта 4](#_Toc124874820)

[2 Разработка регламента проведения инспекции 5](#_Toc124874821)

[3 Разработка модели состояний задач 8](#_Toc124874822)

[4 Разработка презентации проекта 10](#_Toc124874823)

[5 Разработка требований к проекту 13](#_Toc124874824)

[6 Разработка архитектуры проекта 17](#_Toc124874825)

[7 Разработка измерений проекта 19](#_Toc124874826)

[8 Разработка перечня задач проекта 21](#_Toc124874827)

[9 Разработка рекомендаций по кодированию 23](#_Toc124874828)

[10 Разработка плана тестирования проекта 25](#_Toc124874829)

[11 Тестирование проекта 27](#_Toc124874830)

[Заключение 31](#_Toc124874831)

[Список литературы 32](#_Toc124874832)

# Введение

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

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

В данной курсовой работе рассматривается задача коллективной разработки программного средства «Транслятор из Java в C++» и составление технической документации к данному средству.

Таким образом, целью курсовой работы является разработка программного средства «Транслятор из Java в C++» 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

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

В нашем случае исполнителями являются следующие лица:

* Team Leader — Ильченко А.С.
* Coder 1 — Бураков А.А.
* Coder 2 — Зенков М.А.
* Coder 3 — Ильченко А.С.
* Coder 4 — Крючкова Н.Ю.
* Build Engineer — Крючкова Н.Ю.
* Technical Writer — Бураков А.А.

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:

* 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 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 невозможности проведения неформальной инспекции.

## Участники инспекции

Участники могут иметь следующие роли:

1. Автор — участник, внёсший изменения в рабочий продукт. Загружает изменения в систему контроля версий и инициирует инспекцию.
2. Председатель — тимлид (Team Leader), контролирующий процесс инспекции. Назначает инспектора и одобряет внесение ключевых изменений в основную версию продукта.
3. Инспектор — участник, проводящий проверку внесённых изменений. Оставляет замечания и выносит вердикт о внесении изменений в продукт.

В инспекции в обязательном порядке присутствуют два участника, имеющие роли автора и инспектора соответственно. При внесении изменений в дизайн рабочего продукта, необходимо присутствие председателя, который может также выполнять роль инспектора.

## Этапы инспекции

1. Инициация — создание автором запроса на внесение изменений в продукт и назначение инспектора.
2. Подготовка и проведение — анализ изменений и внесение замечаний инспектором.
3. Завершение — вынесение вердикта о внесении изменений в продукт.

## Порядок организации инспекции

Работа над продуктом ведётся в системе контроля версий GIT. Автор изменений оформляет Pull Request (Merge Request) и назначает инспектора, отправляет в рабочий чат в ВК сообщение с упоминанием инспектора.

Столкнувшись с изменением дизайна проекта, инспектор обращается к председателю инспекции в рабочем чате в ВК. По окончании своей работы инспектор одобряет Pull Request или отправляет на доработку автору, уведомляя его в рабочем чате в ВК.

## Порядок подготовки и проведения инспекции

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

После анализа изменений инспектор оставляет в системе контроля версий замечания, обозначая степень их важности. При наличии замечаний, требующих исправлений, работа передаётся автору на доработку. При отсутствии подобных замечаний инспекция считается завершённой, и изменения вступают в силу.

## Перечень статусов и степени важности замечаний

1. Комментарий — рекомендация по улучшению продукта, не требующая обязательных изменений.
2. Ошибка — замечание, сообщающее о необходимости исправления.

## Порядок верификации учёта замечаний

После внесения повторных изменений инспектор просматривает замечания и проверяет соответствующие изменения. По окончании верификации выносится вердикт о внесении изменений в продукт или выносятся новые замечания.

## Метрики, характеризующие эффективность инспекций

Inspection Rate(IR): IR = Размер продукта / Общее время инспектирования

Стратегическая цель метрики — повысить качество разрабатываемого ПО.

Изучаемый объект метрики — проведение инспекции, измеряемый атрибут — производительность инспектирования.

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

* Backlog — новые и отложенные задачи.
* In progress — задачи в процессе выполнения.
* Testing — задачи в состоянии проверки работоспособности изменений.
* To review — задачи в состоянии проверки изменений на соответствие требованиям проекта.
* Done — выполненные задачи.

## Правила создания новой задачи

Любой участник команды разработки может в любое время создавать задачи в рамках назначенной ему части проекта.

## Правила перехода задачи из состояния в состояние

* В качестве системы отслеживания задач используется Git Hub Projects.
* Созданная задача имеет состояние «Backlog». Team Leader может назначать на задачу исполнителя и инспектора, также участник команды может взять роль исполнителя задачи на себя.
* Как только участник команды разработки готов приступить к задаче, на которую он назначен, он переводит её в состояние «In progress». На этом этапе исполнитель выполняет задачу.
* Когда задача выполнена, она переходит в состояние «Testing». На этом этапе исполнитель проверяет работоспособность изменений на наборе тестов.
* По завершении тестирования, задача переходит в состояние «To review». На этом этапе проходит инспекция изменений. В случае необходимости внесения изменений, состояния задачи меняется на «In progress», иначе на «Done».

# Разработка презентации проекта

Была разработана презентация проекта, состоящая из шести слайдов:

1. Титульный слайд [Рисунок 2]
2. Проблема пользователей [Рисунок 3]
3. Решение проблемы пользователей [Рисунок 4]
4. Команда разработки [Рисунок 5]
5. Принцип работы программного продукта [Рисунок 6]
6. Инструкция по использованию [Рисунок 7]

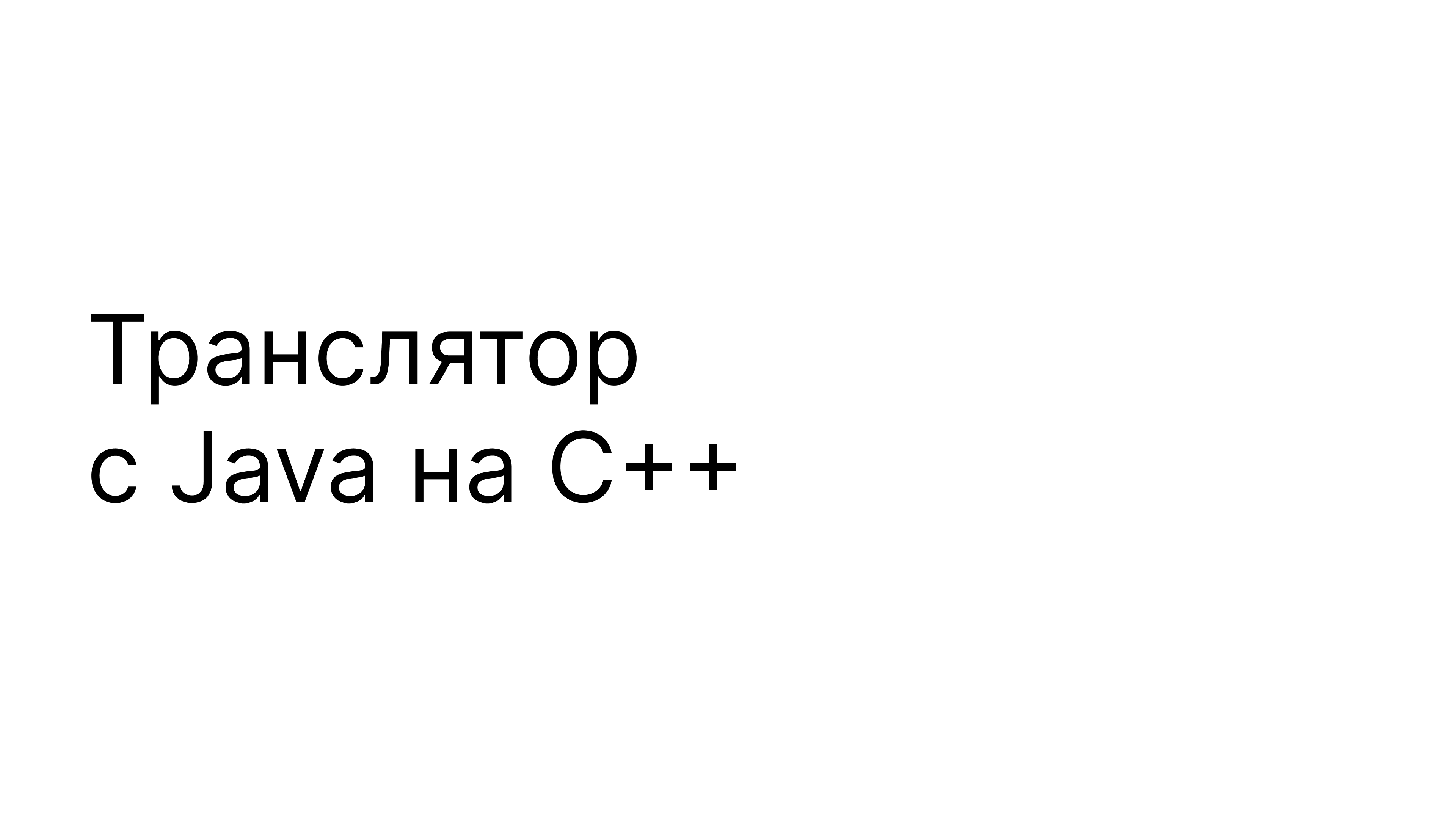


Рисунок 2 — Титульный слайд

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3 — Проблема пользователей

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 — Решение проблемы пользователей

Изображение выглядит как текст, визитка

Автоматически созданное описание

Рисунок 5 — Команда разработки

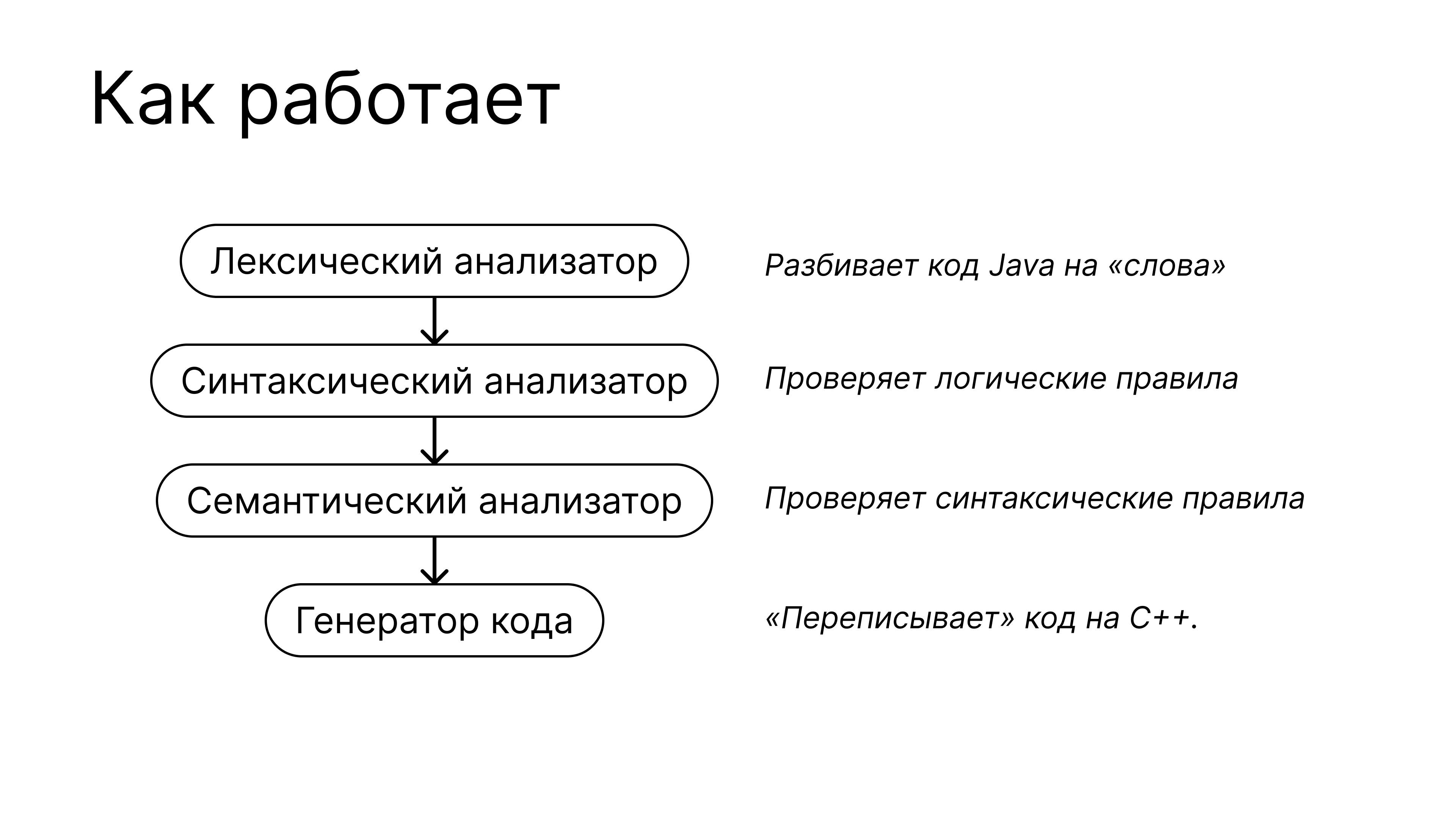


Рисунок 6 — принцип работы программного продукта



Рисунок 7 — инструкция по использованию

# Разработка требований к проекту

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

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

Программный продукт состоит из следующих подсистем:

1. Пользовательский интерфейс.
2. Модуль лексического анализа.
3. Модуль синтаксического анализа.
4. Модуль семантического анализа.
5. Модуль генерации кода.

## Общие требования

### Функциональные требования

Требование FU\_001 Продукт должен транслировать код с языка Java на C++.

Требование FU\_002 Продукт должен формировать код на языке C++.

### Нефункциональные требования

Требование UF\_001 Продукт должен быть написан на языке Python версии 3.x.x.

Требование UF\_002 Продукт должен быть развёртываем с помощью docker.

Требование UF\_003 Продукт должен иметь веб-интерфейс.

Требование UF\_004 Контроль версий продукта должен вестись в GitHub.

Требование UF\_005 Для каждой подсистемы должны быть подготовлены unit-тесты.

### Системные требования

Требование SM\_001 Продукт должен запускаться на устройствах, поддерживающих работу docker.

## Требования к подсистеме «Пользовательский интерфейс»

Требование UI\_001 На сайте должны быть следующие элементы интерфейса:

1. кнопка добавления файла с исходным кодом на Java,
2. поле для отображения исходного кода,
3. поле для отображения транслированного кода на C++,
4. кнопка отправки запроса на перевод.

Требование UI\_002 Если пользователь ввёл некорректные данные, то по нажатии кнопки отправки запроса на трансляцию (см. UI\_001) пользователь должен получать сообщение с описанием ошибки.

Требование UI\_003 Сообщение об ошибке (см. UI\_002) должно содержать в себе информацию о типе ошибки (лексическая, синтаксическая, неверный ввод) и номер строки, где она возникла.

Требование UI\_004 Система должна принимать файлы с расширением .java при добавлении файла по нажатии кнопки (см. UI\_001).

## Требования к подсистеме «Модуль лексического анализа»

Требование LX\_001 Модуль должен получать на вход набор символов и разбить его на токены (см. LX\_002) и сохранить результат в формате .json.

Требование LX\_002 Каждый токен должен содержать код лексемы из выбранного подмножества языка Java, значение лексемы, номер строки и столбца, в которых начинается лексема. JSON-структура токена [Рисунок 8].

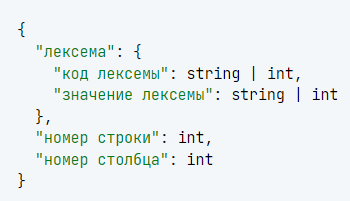


Рисунок 8 — JSON-структура токена

Требование LX\_003 В случае нахождения лексемы, не попадающей в список выделенных (см. LX\_002), токен помечается ошибкой. При этом модуль не прекращает работу.

Требование LX\_004 В случае нахождения ошибки, необходимо сформировать сообщение с описанием ошибки и доставить его до пользовательского интерфейса (см. UI\_002).

## Требования к подсистеме «Модуль синтаксического анализа»

Требование SY\_001 На вход модулю подаётся JSON-файл (см. LX\_001), на основе которого строится дерево разбора.

Требование SY\_002 В случае нахождения синтаксической или лексической (см. LX\_003) ошибки, необходимо сформировать сообщение с описанием ошибки и доставить его до пользовательского интерфейса (см. UI\_002).

## Требования к подсистеме «Модуль семантического анализа»

Требование SE\_001 На вход поступает дерево разбора, (см. SY\_001), оно проверяется на корректность модулем.

Требование SE\_002 В случае нахождения семантической ошибки, необходимо сформировать сообщение с описанием ошибки и доставить его до пользовательского интерфейса (см. UI\_002).

## Требования к подсистеме «Модуль генерации кода»

Требование CG\_001 На вход поступает дополненное дерево разбора (см. SE\_001), на его основе происходит генерация кода на языке C++ (см. CG\_002) и выводится в пользовательский интерфейс (см. UI\_001).

Требование CG\_002 Каждому токену языка Java и каждой конструкции языка должна быть выделена единственным образом конструкция языка C++.

# 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.

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 между подсистемами разрабатываемого транслятора [Рисунок 9]. На вход интерфейс принимает файл, содержащий код на языке Java. 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++.

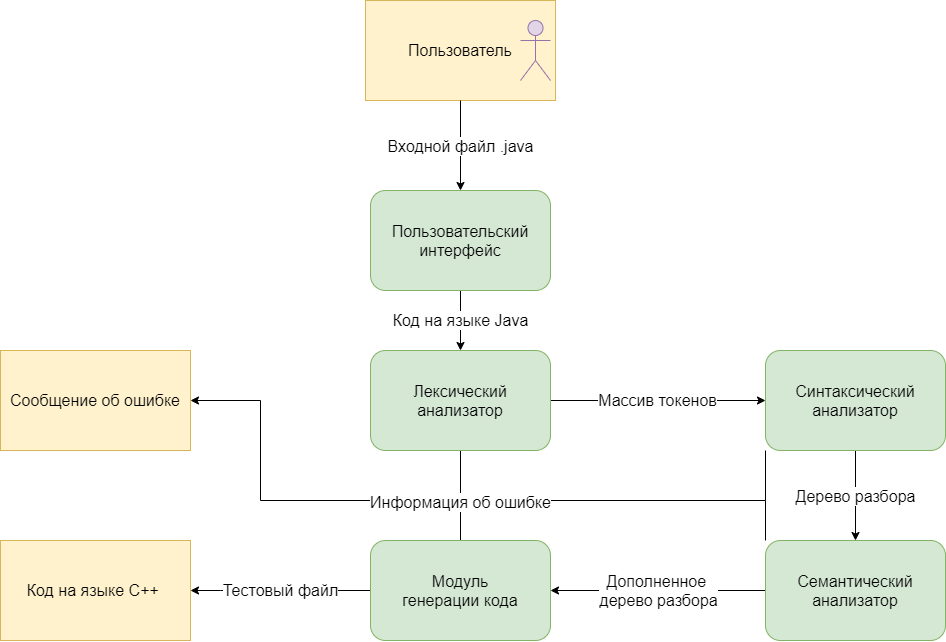


Рисунок 9 — Диаграмма потоков данных

# 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. На крупных предприятиях программа измерений осуществляется специальным отделом по обеспечению качества (SQA — Software Quality Assurance team).

## Метрики эффективности процесса разработки

1. Problem Resolution Rate (PRR)

PRR = Количество дней на обработку задачи

Стратегическая цель метрики — сократить сроки разработки модулей ПО.

Изучаемый объект метрики — задача.

Измеряемый атрибут — время обработки.

Единица измерения — день.

2. Faults Screening (FS)

FS = (Общее количество ошибок − Число ошибок найденных до бета-тестирования) × 100% / Общее количество ошибок

Стратегическая цель метрики — повысить качество разрабатываемого ПО.

Изучаемый объект метрики — проект.

Измеряемый атрибут — эффективность обнаружения дефектов.

Единица измерения — %.

## Метрики качества программного продукта

1. Beta Testing Faults (BTF)

BTF = Число обнаруженных на этапе бета-тестирования ошибок / LOC

Стратегическая цель метрики — повысить качество разрабатываемого ПО.

Изучаемый объект метрики — продукт.

Измеряемый атрибут — плотность неполадок.

Единица измерения — неполадка / LOC.

# Разработка перечня задач проекта

## Общие задачи

[UF-1] Придумать как разнести по файликам модули

Создать модули для подсистем.

[UF-2] Подготовить пайплайн для тестов

Подготовить файлы с кодом на языке Java, скрипты для автоматического запуска тестов программной системы на них.

## Задачи для подсистемы «Пользовательский интерфейс»

[UI-1] Добавить основные элементы интерфейса

Необходимо добавить на сайт поля блоки с кодом, инструменты загрузки файла и кнопку начала трансляции (см. UI\_001).

[UI-2] Вывод однородных ошибок

Необходимо собрать ошибки всех модулей и выводить их на экран пользователя в одном формате в случае их возникновения (см. UI\_002). Сообщение об ошибке должно содержать в себе её примерное местоположение (см. UI\_003).

[UI-3] Загрузка файла для трансляции

Код на Java должен быть закружен в систему в виде файла, в интерфейсе нужен соответствующий функционал (см. UI\_004).

## Задачи для подсистемы «Модуль лексического анализа»

[LX-1] Разработать начальную версию

Вход: любой код на языке Java.

Выход: JSON-файл с токенами (LX\_001, LX\_002, LX\_003).

[LX-2] Обеспечить вывод человекопонятных ошибок

Реализация требования LX\_004.

[LX-3] Написать и проверить тесты для лексера

Подготовить начальный набор тестов для модуля.

## Задачи для подсистемы «Модуль синтаксического анализа»

[SY-1] Разработать начальную версию

Вход: JSON-файл с токенами (SY\_001).

Выход: дерево разбора.

[SY-2] Обеспечить вывод человекопонятных ошибок

Реализация требования SY\_002.

[SY-3] Написать и проверить тесты для синтаксера

Подготовить начальный набор тестов для модуля.

## Задачи для подсистемы «Модуль семантического анализа»

[SE-1] Начальная версия семантического анализатора

Вход: дерево разбора.

Выход: дерево разбора и сообщение о семантической ошибке, если таковая имеется (SE\_001).

[SE-2] Обеспечить вывод человекопонятных ошибок

Реализация требования SE\_002.

[SE-3] Написать и проверить тесты для семантического анализатора

Подготовить начальный набор тестов для модуля.

## Задачи для подсистемы «Модуль генерации кода»

[CG-1] Начальная версия генератора кода

Вход: дерево разбора.

Выход: текст на языке C++ (см. CG\_002).

[CG-2] Написать и проверить тесты для генератора кода

Подготовить начальный набор тестов для модуля.

# 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.

В основе рекомендаций — руководство по оформлению кода на Python PEP-8. Все правила PEP-8 должны соблюдаться в проекте. Ниже перечислены самые важные рекомендации и требования по оформлению кода.

## Рекомендации и требования к оформлению кода

1. snake\_case для всех имён, кроме констант и классов.
2. SCREAMING\_SNAKE\_CASE для констант.
3. UpperCamelCase для имён классов.
4. Во всех именах ключевое слово должно быть существительным, за исключением имён функций и методов.
5. Имена функций и методов должны начинаться с глагола.
6. В файле не должно быть больше 250 строк.
7. Функции отделяются друг от друга двумя пустыми строками.
8. Методы отделяются друг от друга одной пустой строкой.
9. Каждый модуль должен находиться в отдельной папке.
10. В выражениях не должны участвовать неименованные константы (magic numbers).

## Чек-лист для проверки корректности кода

* snake\_case для всех имён, кроме констант и классов.
* SCREAMING\_SNAKE\_CASE для констант.
* UpperCamelCase для имён классов.
* Во всех именах ключевое слово — существительное, кроме функций и методов.
* Имена функций и методов начинаются с глагола.
* Нет файлов, где больше 250 строк.
* Функции отделены друг от друга двумя пустыми строками.
* Методы отделены друг от друга одной пустой строкой.

# Разработка плана тестирования проекта

## Тесты для подсистемы «Пользовательский интерфейс»

**Тест TEST\_UI\_001**

Тестируемые требования: UI\_001

На сайте должны быть следующие элементы интерфейса:

* кнопка добавления файла с исходным кодом на Java,
* поле для отображения исходного кода,
* поле для отображения транслированного кода на C++,
* кнопка отправки запроса на перевод.

**Тест TEST\_UI\_002**

Тестируемые требования: UI\_002, UI\_003

Сообщение об ошибке при вводе некорректного файла.

**Тест TEST\_UI\_003**

Тестируемые требования: UI\_004

Сайт должен принимать на вход файл только с расширением .java

## Тесты для подсистемы «Модуль лексического анализа»

**Тест TEST\_LX\_001**

Тестируемые требования: LX\_001, LX\_002

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

**Тест TEST\_LX\_002**

Тестируемые требования: LX\_003, LX\_004

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\_SY\_001**

Тестируемые требования: SY\_001

Проверка работоспособности подсистемы на работу с основными арифметическими операциями. На вход подаётся файл с кодом, в котором создаются две переменные типа int: а = 5 и b = 2. Далее они используются в формуле (3 + (a - 1)) \* b / 2.

**Тест TEST\_SY\_002**

Тестируемые требования: SY\_001

Проверка работоспособности подсистемы на работу с несколькими пользовательскими методами. На вход подаётся файл с кодом, в котором объявлены два метода: до метода main и после него.

**Тест TEST\_SY\_003**

Тестируемые требования: SY\_002

Проверка корректности сообщения при ошибке с недостающей точкой с запятой. На вход подаётся файл, в котором не хватает точки с запятой на 6 строке.

**Тест TEST\_SY\_004**

Тестируемые требования: SY\_002

Проверка корректности сообщения при ошибке с отсутствующей в грамматике конструкцией языка. На вход подаётся файл, в котором не хватает скобок в блоке условного оператора в 4 строке.

## Тесты для подсистемы «Модуль семантического анализа»

**Тест TEST\_SE\_001**

Тестируемые требования: SE\_001

Проверка работоспособности подсистемы на работу с основными арифметическими операциями. На вход подаётся файл с кодом, в котором создаются две переменные типа int: а = 5 и b = 2. Далее они используются в формуле (3 + (a - 1)) \* b / 2, значение которой присваивается переменной c типа double.

**Тест TEST\_SE\_002**

Тестируемые требования: SE\_001

Проверка работоспособности подсистемы на работу с вложенными всегда достижимыми return у функции. На вход подаётся файл с кодом, в котором описана функция типа int, операторы return записаны в обеих ветках условного оператора внутри кода функции.

**Тест TEST\_SE\_003**

Тестируемые требования: SE\_002

Проверка корректности сообщения при ошибке с множественным объявлением функции с одной и той же сигнатурой plus(int a, int b).

**Тест TEST\_SE\_004**

Тестируемые требования: SE\_002

Проверка корректности сообщения при ошибке с присвоением переменной типа int значения типа double.

## Тесты для подсистемы «Модуль генерации кода»

**Тест TEST\_CG\_001**

Тестируемые требования: CG\_001

Проверка работоспособности подсистемы на работу с основными арифметическими операциями, объявлением функции и стандартными функциями println, min и max. На вход подаётся файл с кодом, в котором создаются две переменные типа int: а = 5 и b = 2; а также функция formula, получающая на вход два параметра типа int a и b и возвращающая результат формулы (3 + (a - 1)) \* b / 2. В функции main переменные a и b передаются в каестве параметров в функцию formula, полученный результат присваивается переменной типа int c. После этого перменная c передаётся в качестве параметра в функцию println, после этого переменные a и b передаются в качестве параметров в функции min и max, результат которых в свою очередь опять передаётся в функцию println для проверки.

**Тест TEST\_CG\_002**

Тестируемые требования: CG\_001

Проверка работоспособности подсистемы на работу с циклами for, while и do while. На вход подаётся файл с кодом, в котором содержится цикл for для переменной типа int i от 0 до 5, внутри которого с помощью функции println выводится текущее значение i. Далее после цикла for создаётся переменная типа int i = 0, после этого проверяется цикл while с условием i < 5, внутри которого сначала выводится текущее значение переменной i с помощью функции println, а затем значение i инкрементируется. После цикла while переменной i присваивается значение 0 и проверяется цикл do while с условием i < 5, внутреннее содержимое цикла аналогично циклу while.

**Тест TEST\_CG\_003**

Тестируемые требования: CG\_001

Проверка работоспособности подсистемы на работу с условным оператором. На вход подаётся файл с кодом, в котором создаются две переменные типа float: а = 1.5 и b = 2.0. Затем в условном операторе if проверяется условие a < b && b == 2.0, в блоке if с помощью функции println выводится переменная a, в блоке else выводится переменная b.

**Тест TEST\_CG\_004**

Тестируемые требования: CG\_002

Проверка подсистемы на единообразие выводимого результата для всех токенов и конструкций языка, присутствующих в грамматике. На вход подаётся файл с кодом, содержимое которого аналогично тестам TEST\_CG\_001, TEST\_CG\_002 и TEST\_CG\_003, объединённым вместе. Помимо этого в функции main создаётся по одной переменной следующих типов: double и char; а также две переменные типа boolean, одной из которых присваивается значение true, другой - значение false. Помимо конструкции if из теста TEST\_CG\_003 проверяются также конструкии if с операторами сравнения >, >=, <= и !=, а также с логическими операторами ! и ||. Помимо этого проверяются все операторы присваивания: сначала создаётся переменная типа int k = 1, после этого этой переменной присваиваются значения с помощью операторов +=, -=, \*=, /=, %=, а также проверяются операторы ++ и --. Для проверки всех математических операторов создаётся переменная типа float f, которой присваивается значение формулы (1.0 + 0.5 - 0.25) \* 2.0 / (5 % 2);

**Тест TEST\_CG\_005**

Тестируемые требования: CG\_002

Проверка подсистемы на единообразие выводимого результата для всех токенов и конструкций языка, присутствующих в грамматике. На вход подаётся файл с кодом, содержимое которого аналогично тесту TEST\_CG\_004, но с изменёнными именами и значениями переменных, а также с некоторыми переставленными местами циклами и вызываемыми функциями.

## Матрица покрытия тестами требований

Была построена матрица покрытия тестами требований [Рисунок 10].

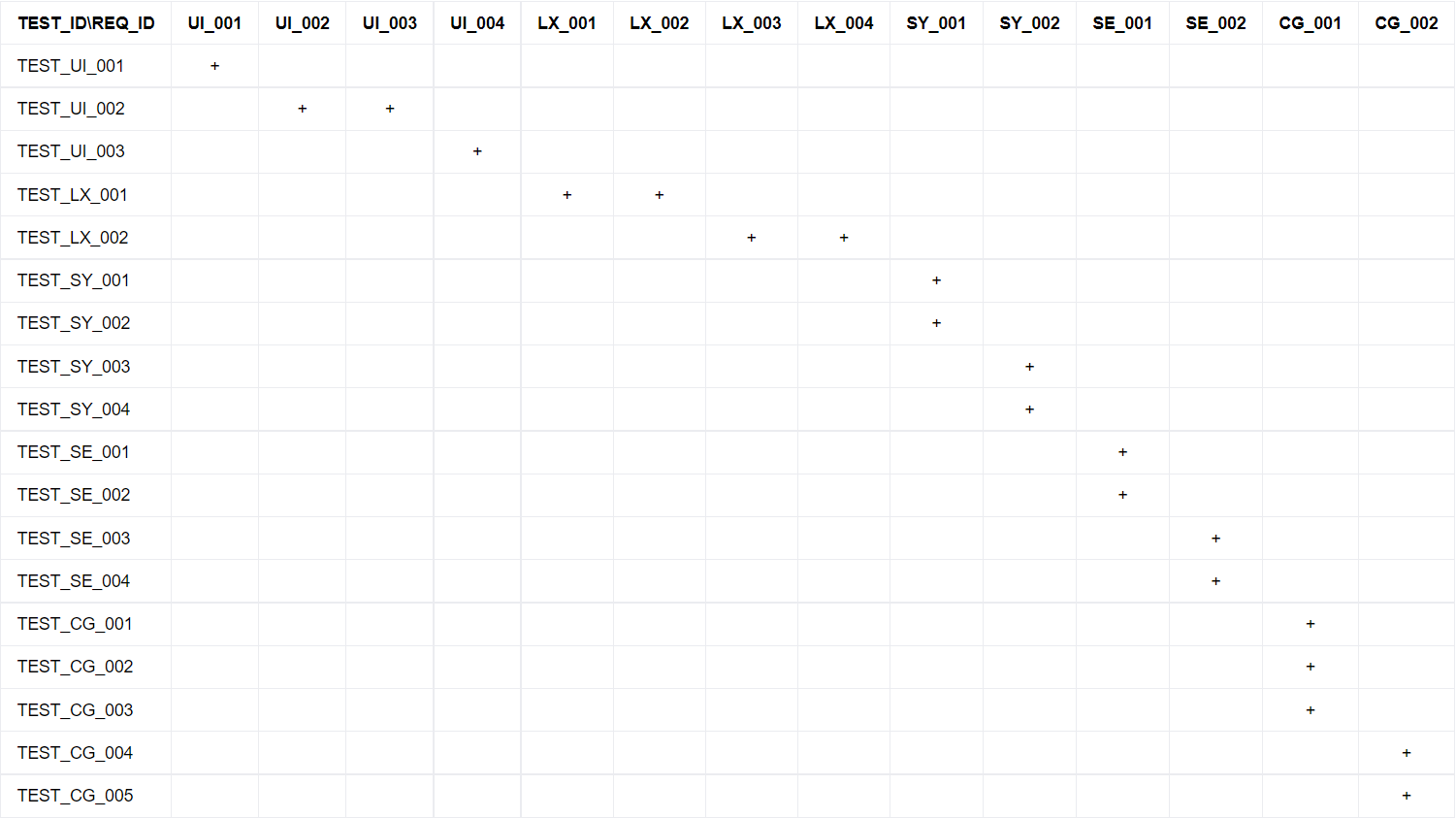


Рисунок 10 — Матрица покрытия тестами требований

# Тестирование проекта

## Тесты для подсистемы «Пользовательский интерфейс»

**Тест TEST\_UI\_001**

Тестируемые требования: UI\_001

На сайте должны быть следующие элементы интерфейса:

* кнопка добавления файла с исходным кодом на Java,
* поле для отображения исходного кода,
* поле для отображения транслированного кода на C++,
* кнопка отправки запроса на перевод.

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: Все элементы интерфейса есть на сайте

Видимый результат: Все элементы интерфейса есть на сайте

Резюме: Тест пройден

**Тест TEST\_UI\_002**

Тестируемые требования: UI\_002, UI\_003

Сообщение об ошибке при вводе некорректного файла.

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: Вывод исходного кода и сообщения об ошибке, которое содержит описание ошибки

Видимый результат: Вывод исходного кода и сообщения об ошибке, которое содержит описание ошибки

Резюме: Тест пройден

**Тест TEST\_UI\_003**

Тестируемые требования: UI\_004

Сайт должен принимать на вход файл только с расширением .java

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: возможность прикрепить только файлы с расширением .java

Видимый результат: возможность прикрепить только файлы с расширением .java

Резюме: Тест пройден

## Тесты для подсистемы «Модуль лексического анализа»

**Тест TEST\_LX\_001**

Тестируемые требования: LX\_001, LX\_002

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

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: Файл с массивом токенов формата JSON

Видимый результат: Файл с массивом токенов формата JSON

Резюме: Тест пройден

**Тест TEST\_LX\_002**

Тестируемые требования: LX\_003, LX\_004

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.

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: Модуль ловит лексему, не прекращает работу и выводит сообщение, которое показывается пользователю.

Видимый результат: Модуль ловит лексему, не прекращает работу и выводит сообщение, которое показывается пользователю.

Резюме: Тест пройден

## Тесты для подсистемы «Модуль синтаксического анализа»

**Тест TEST\_SY\_001**

Тестируемые требования: SY\_001

Проверка работоспособности подсистемы на работу с основными арифметическими операциями. На вход подаётся файл с кодом, в котором создаются две переменные типа int: а = 5 и b = 2. Далее они используются в формуле (3 + (a - 1)) \* b / 2.

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: Дерево разбора

Видимый результат: Дерево разбора

Резюме: Тест пройден

**Тест TEST\_SY\_002**

Тестируемые требования: SY\_001 Проверка работоспособности подсистемы на работу с несколькими пользовательскими методами. На вход подаётся файл с кодом, в котором объявлены два метода: до метода main и после него.

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: Дерево разбора

Видимый результат: Сообщение о синтаксической ошибке в строке 15

Резюме: Тест не пройден

**Тест TEST\_SY\_003**

Тестируемые требования: SY\_002

Проверка корректности сообщения при ошибке с недостающей точкой с запятой. На вход подаётся файл, в котором не хватает точки с запятой на 6 строке.

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: Сообщение о синтаксической ошибке в строке 7

Видимый результат: Сообщение о синтаксической ошибке в строке 7

Резюме: Тест пройден

**Тест TEST\_SY\_004**

Тестируемые требования: SY\_002

Проверка корректности сообщения при ошибке с отсутствующей в грамматике конструкцией языка. На вход подаётся файл, в котором не хватает скобок в блоке условного оператора в 4 строке.

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: Сообщение о синтаксической ошибке в строке 4

Видимый результат: Сообщение о синтаксической ошибке в строке 4

Резюме: Тест пройден

## Тесты для подсистемы «Модуль семантического анализа»

**Тест TEST\_SE\_001**

Тестируемые требования: SE\_001

Проверка работоспособности подсистемы на работу с основными арифметическими операциями. На вход подаётся файл с кодом, в котором создаются две переменные типа int: а = 5 и b = 2. Далее они используются в формуле (3 + (a - 1)) \* b / 2, значение которой присваивается переменной c типа double.

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: Дерево корректно

Видимый результат: Дерево корректно

Резюме: Тест пройден

**Тест TEST\_SE\_002**

Тестируемые требования: SE\_001

Проверка работоспособности подсистемы на работу с вложенными всегда достижимыми return у функции. На вход подаётся файл с кодом, в котором описана функция типа int, операторы return записаны в обеих ветках условного оператора внутри кода функции.

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: Дерево корректно

Видимый результат: semantic error in string 4: missing expression with return for function plus

Резюме: Тест не пройден

**Тест TEST\_SE\_003**

Тестируемые требования: SE\_002

Проверка корректности сообщения при ошибке с множественным объявлением функции с одной и той же сигнатурой plus(int a, int b).

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: semantic error in string 8: function plus already declared

Видимый результат: semantic error in string 8: function plus already declared

Резюме: Тест пройден

**Тест TEST\_SE\_004**

Тестируемые требования: SE\_002

Проверка корректности сообщения при ошибке с присвоением переменной типа int значения типа double.

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: semantic error in string 6: return value double does not match the required type int

Видимый результат: semantic error in string 6: return value double does not match the required type int

Резюме: Тест пройден

## Тесты для подсистемы «Модуль генерации кода»

**Тест TEST\_CG\_001**

Тестируемые требования: CG\_001

Проверка работоспособности подсистемы на работу с основными арифметическими операциями, объявлением функции и стандартными функциями println, min и max. На вход подаётся файл с кодом, в котором создаются две переменные типа int: а = 5 и b = 2; а также функция formula, получающая на вход два параметра типа int a и b и возвращающая результат формулы (3 + (a - 1)) \* b / 2. В функции main переменные a и b передаются в каестве параметров в функцию formula, полученный результат присваивается переменной типа int c. После этого перменная c передаётся в качестве параметра в функцию println, после этого переменные a и b передаются в качестве параметров в функции min и max, результат которых в свою очередь опять передаётся в функцию println для проверки.

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: код на C++ с корректным отображением

Видимый результат: код на C++ с корректным отображением

Резюме: Тест пройден

**Тест TEST\_CG\_002**

Тестируемые требования: CG\_001

Проверка работоспособности подсистемы на работу с циклами for, while и do while. На вход подаётся файл с кодом, в котором содержится цикл for для переменной типа int i от 0 до 5, внутри которого с помощью функции println выводится текущее значение i. Далее после цикла for создаётся переменная типа int i = 0, после этого проверяется цикл while с условием i < 5, внутри которого сначала выводится текущее значение переменной i с помощью функции println, а затем значение i инкрементируется. После цикла while переменной i присваивается значение 0 и проверяется цикл do while с условием i < 5, внутреннее содержимое цикла аналогично циклу while.

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: код на C++ с корректным отображением

Видимый результат: некорректное отображение внутри цикла for

Резюме: Тест не пройден

**Тест TEST\_CG\_003**

Тестируемые требования: CG\_001

Проверка работоспособности подсистемы на работу с условным оператором. На вход подаётся файл с кодом, в котором создаются две переменные типа float: а = 1.5 и b = 2.0. Затем в условном операторе if проверяется условие a < b && b == 2.0, в блоке if с помощью функции println выводится переменная a, в блоке else выводится переменная b.

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: код на C++ с корректным отображением

Видимый результат: код на C++ с корректным отображением

Резюме: Тест пройден

**Тест TEST\_CG\_004**

Тестируемые требования: CG\_002

Проверка подсистемы на единообразие выводимого результата для всех токенов и конструкций языка, присутствующих в грамматике. На вход подаётся файл с кодом, содержимое которого аналогично тестам TEST\_CG\_001, TEST\_CG\_002 и TEST\_CG\_003, объединённым вместе. Помимо этого в функции main создаётся по одной переменной следующих типов: double и char; а также две переменные типа boolean, одной из которых присваивается значение true, другой - значение false. Помимо конструкции if из теста TEST\_CG\_003 проверяются также конструкии if с операторами сравнения >, >=, <= и !=, а также с логическими операторами ! и ||. Помимо этого проверяются все операторы присваивания: сначала создаётся переменная типа int k = 1, после этого этой переменной присваиваются значения с помощью операторов +=, -=, \*=, /=, %=, а также проверяются операторы ++ и --. Для проверки всех математических операторов создаётся переменная типа float f, которой присваивается значение формулы (1.0 + 0.5 - 0.25) \* 2.0 / (5 % 2);

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: код на C++ с единообразным результатом для всех токенов и конструкций

Видимый результат: код на C++ с единообразным результатом для всех токенов и конструкций

Резюме: Тест пройден

**Тест TEST\_CG\_005**

Тестируемые требования: CG\_002

Проверка подсистемы на единообразие выводимого результата для всех токенов и конструкций языка, присутствующих в грамматике. На вход подаётся файл с кодом, содержимое которого аналогично тесту TEST\_CG\_004, но с изменёнными именами и значениями переменных, а также с некоторыми переставленными местами циклами и вызываемыми функциями.

Тестируемая версия продукта: 1.0.1

Ожидаемый результат: код на C++ с единообразным результатом для всех токенов и конструкций

Видимый результат: код на C++ с единообразным результатом для всех токенов и конструкций

Резюме: Тест пройден

# Заключение

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

1. Количество дней на обработку задачи (PRR) составило 14.533 дней.
2. Эффективность обнаружения дефектов (FS) составила 82.352%
3. Плотность неполадок (BTF) составила 0.0026 неполадок / LOC.

В рамках курсовой работы было разработано программное средство «Транслятор из Java 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 вариант.