  
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. Б9121-09.03.04  
  
Руководитель:

г. Владивосток  
2024

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 С# в Java» и составление технической документации к данному средству.

Таким образом, целью курсовой работы является разработка программного средства «Транслятор из С# в Java» 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 — Герус П.А.
* Coder 1 — Проценко Д.В.
* Build Engineer — Ваяй М.С.
* Technical Writer — Моисеев Д.А.

Был разработан перечень задач для выполнения и примерные сроки их реализации [Рисунок 1].

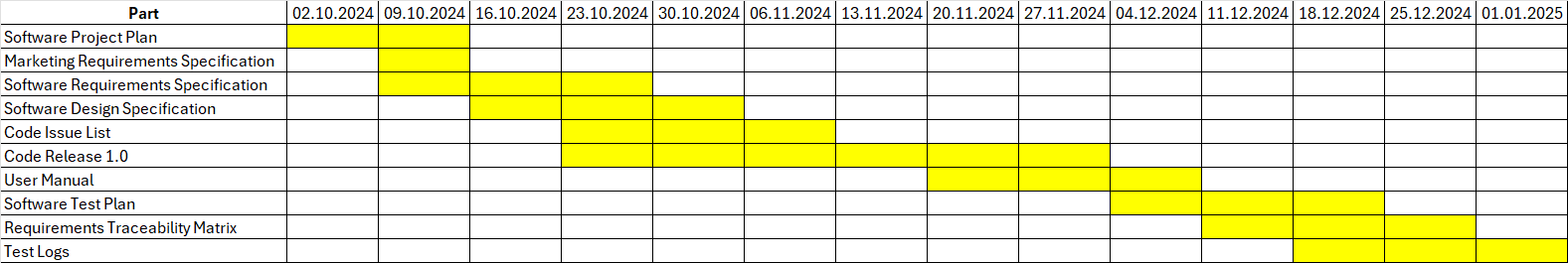


Рисунок — План проекта

# 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]

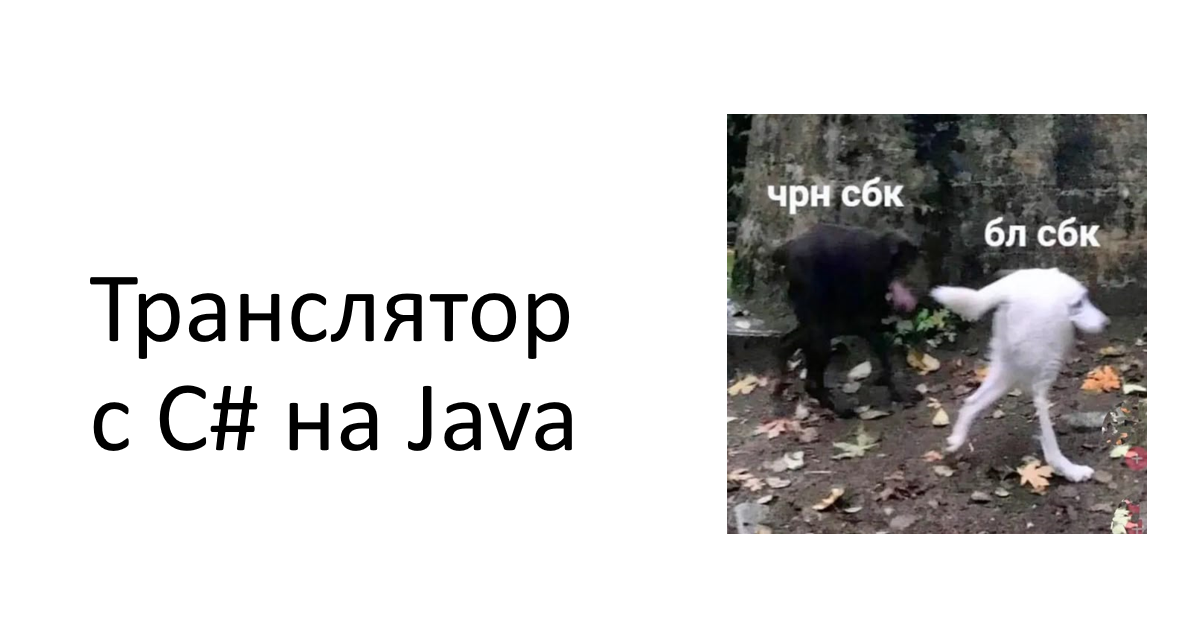


Рисунок — Титульный слайд



Рисунок — Проблема пользователей

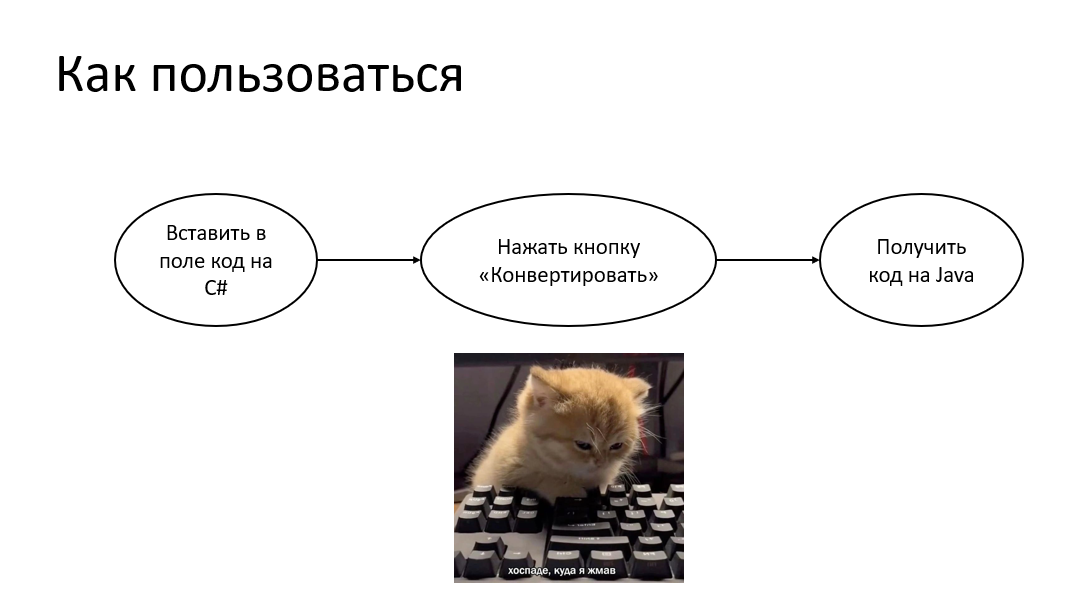


Рисунок — инструкция по использованию



Рисунок — принцип работы программного продукта

# 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 - транслятор должен преобразовывать код, написанный на языке C#, в эквивалентный код на языке Java, сохраняя основную логику и функциональность.

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.

Основные требования:

1. Поддержка синтаксиса:  
    Транслятор должен поддерживать ключевые слова, операторы, типы данных и синтаксис C#, включая:  
    - Базовые типы данных (int, double, string, bool)  
    - Классы, структуры, интерфейсы, перечисления  
    - Методы, свойства, поля  
    - Операторы (арифметические, сравнения, логические)  
    - Условные операторы (if, else, switch)  
    - Циклы (for, while, do-while)  
    - Исключения (try-catch)  
    - Объектно-ориентированное программирование (наследование, полиморфизм)  
   2. Преобразование типов данных:  
   Транслятор должен уметь преобразовывать типы данных C# в соответствующие типы данных Java, например:  
    - int в int  
    - double в double  
    - string в String  
    - bool в boolean  
    - List<T> в ArrayList<T>  
    - Dictionary<K, V> в HashMap<K, V>  
    Транслятор должен учитывать нюансы преобразования, например, для DateTime использовать Date или Calendar в Java.  
   3. Преобразование логики:  
    Транслятор должен уметь преобразовывать логику кода, например:  
    - Вызовы методов, конструкторов, статических методов  
    - Условные операторы и циклы  
    - Обработку исключений  
    - Использование стандартных библиотек .NET (например, [System.IO](https://vk.com/away.php?to=http%3A%2F%2FSystem.IO&cc_key=)) в соответствующие классы Java (например, [java.io](https://vk.com/away.php?to=http%3A%2F%2Fjava.io&cc_key=))  
   4. Создание исходного кода Java:  
    Транслятор должен генерировать корректный исходный код Java, который может быть скомпилирован и запущен.  
   5. Обработка ошибок:  
    Транслятор должен уметь выявлять и сообщать о синтаксических ошибках в исходном коде C#.  
    Транслятор должен уметь выявлять и сообщать о несовместимости между типами данных C# и Java.  
    Транслятор должен уметь предоставлять исчерпывающие сообщения об ошибках с указанием местоположения ошибки в исходном коде C#.  
    Транслятор не должен быть просто строковым преобразователем. Он должен учитывать контекст, семантику кода C# и обеспечивать его корректное преобразование в Java.  
   Транслятор должен быть как можно более полным, чтобы обеспечить максимальную совместимость с исходным кодом C#.  
   Транслятор должен быть гибким и расширяемым, чтобы его можно было легко адаптировать к новым функциям и обновлениям языка C# и Java.

# 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++.

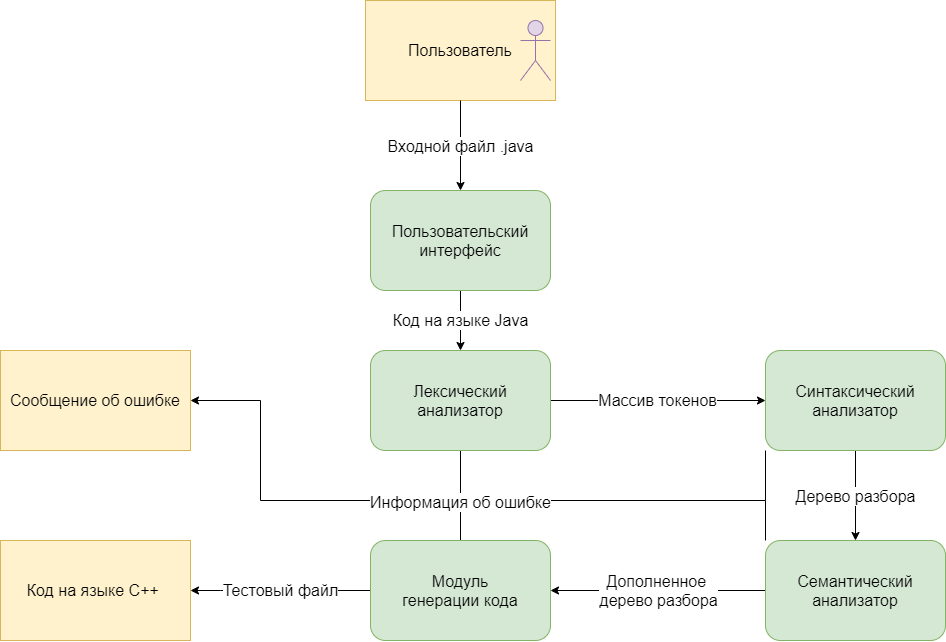


Рисунок 1 — Диаграмма потоков данных

Также составим диаграмму, описывающую сценарий взаимодействия пользователя с нашей системой, то есть «Use case» диаграмму\

Сценарии использования транслятора

1. Загрузить файл с исходным кодом.  
   Пользователь предоставляет файл с кодом на языке C# для анализа и преобразования.
2. Просмотреть ошибки в исходном коде.  
   Пользователь видит сообщения об ошибках в исходном коде (лексических, синтаксических, семантических) и их местоположение.
3. Исправить исходный код.  
   Пользователь исправляет ошибки в коде (вне приложения) и повторно загружает файл.
4. Запустить трансляцию.  
   Пользователь инициирует процесс трансляции кода из C# в Java.
5. Просмотреть сгенерированный код.  
   Пользователь может просмотреть результирующий код на Java в интерфейсе.
6. Сохранить сгенерированный код.  
   Пользователь сохраняет преобразованный файл в указанное место.

