

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  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Аликаева А. А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Калиниченко П. А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кореньков В. В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кублей И. В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Слободчиков В. А.

Руководитель:  
профессор департамента ПИиИИ, д.т.н, профессор  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Гриняк В.М.

г. Владивосток  
2023

**Оглавление**

[Введение 3](#_Toc123657969)

[1 Разработка плана проекта 4](#_Toc123657970)

[2 Разработка регламента проведения инспекции 5](#_Toc123657971)

[3 Разработка модели состояний задач 8](#_Toc123657972)

[4 Разработка презентации проекта 9](#_Toc123657973)

[5 Разработка требований к проекту 11](#_Toc123657974)

[6 Разработка архитектуры проекта 14](#_Toc123657975)

[7 Разработка измерений проекта 16](#_Toc123657976)

[8 Разработка перечня задач проекта 18](#_Toc123657977)

[9 Разработка рекомендаций по кодированию 20](#_Toc123657978)

[10 Разработка плана тестирования проекта 22](#_Toc123657979)

[11 Тестирование проекта 31](#_Toc123657980)

[Заключение 41](#_Toc123657981)

[Список литературы 42](#_Toc123657982)

# 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 Pascal в C#» и составление технической документации к данному средству.

Таким образом, целью курсовой работы является разработка программного средства «Транслятор из Pascal 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.

# 1 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 – Кореньков Владислав;
* Coder 1 – Калиниченко Полина;
* Coder 2 – Кублей Иван;
* Coder 3 – Слободчиков Владислав;
* Build Engineer – Кореньков Владислав;
* Technical Writer – Аликаева Алиса.

На рисунке 1 представлен перечень задач для выполнения и примерные сроки их реализации.

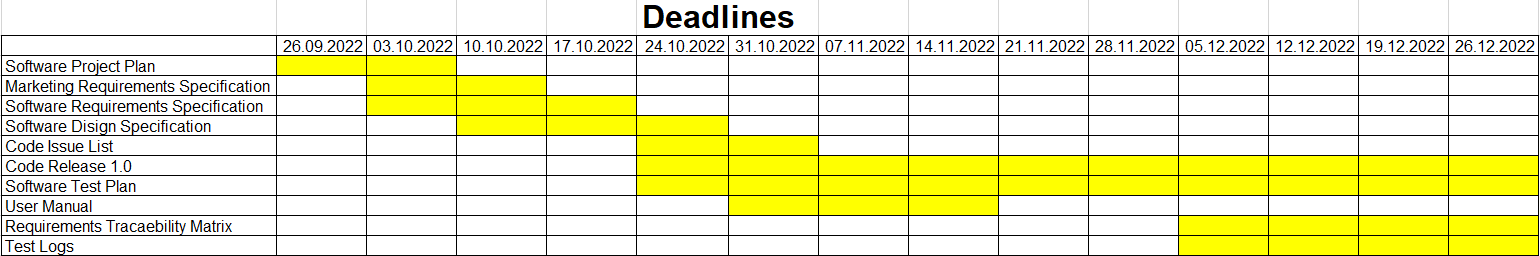


Рисунок 1 – План проекта

# 2 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 отнесения к формальной инспекции

* Разработана новая версия документации или исправлена существующая документация.
* Создание pull request в master в репозиторий на платформе GitHub.

## Участники инспекции

Роли участников формальной инспекции:

* 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.

Роли участников инспекции:

* Автор: Кублей Иван, Слободчиков Владислав, Калиниченко Полина, Аликаева Алиса.
* Инспектор: Кореньков Владислав.

## Этапы инспекции

## Планирование инспекции.

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

## Подготовка к инспекции.

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

## Собрания 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.

## 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

* Требующие исправления
* Требующие обсуждения

## Метрики, характеризующие эффективность инспекций

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>.

# 3 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:

Правило перехода: создание задачи.

1. In queue – задача находится в очереди на исполнение.

Правило перехода: разработчик берется за выполнение задачи.

1. In progress – задача находится в разработке.

Правило перехода: разработчик считает, что задача выполнена.

1. Review – задача требует одобрения от ревьюера.

Правило перехода: задача успешно прошла ревью. Если требуются доработки, то задача возвращается в In progress.

1. Closed – задача завершена.

## Создатели задач:

1. In queue – Владислав Кореньков
2. In progress – разработчик
3. Review – разработчик
4. Closed – Владислав Кореньков

# 4 Разработка презентации проекта

На рисунке 2 представлена титульная страница презентации.

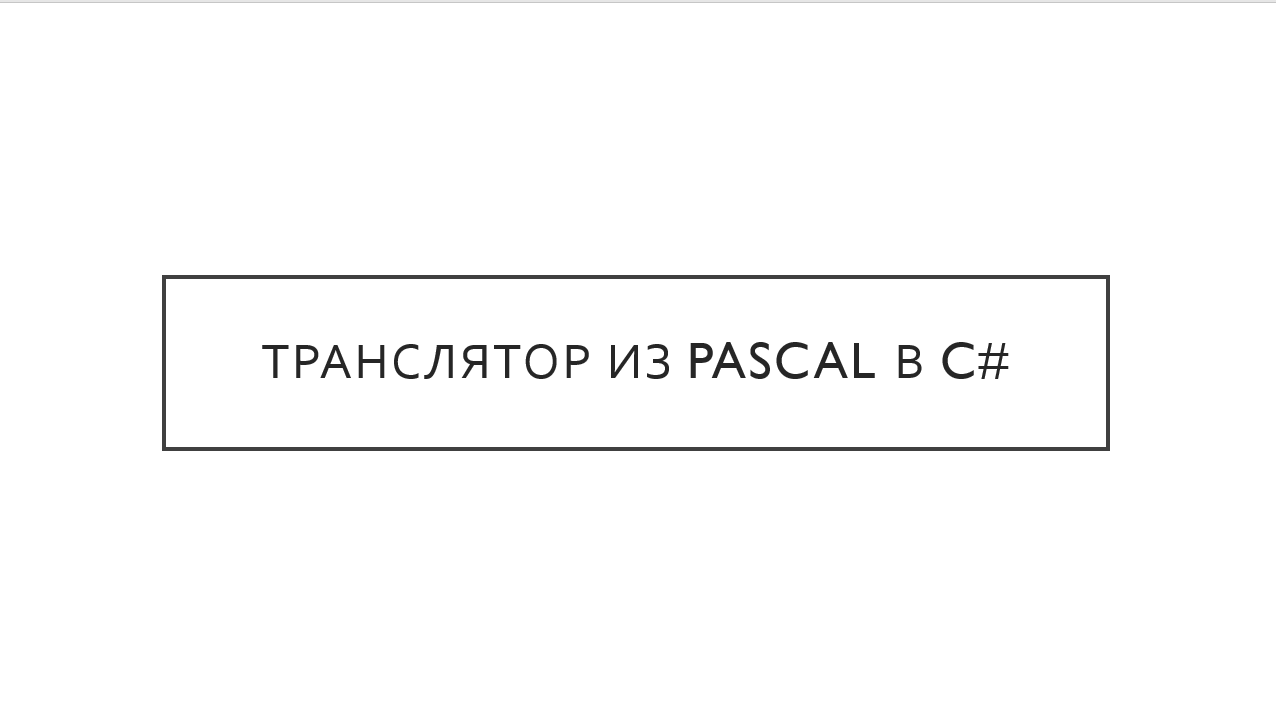


Рисунок 2 – Титульная страница

Актуальность разрабатываемого средства представлена на рисунке 3.

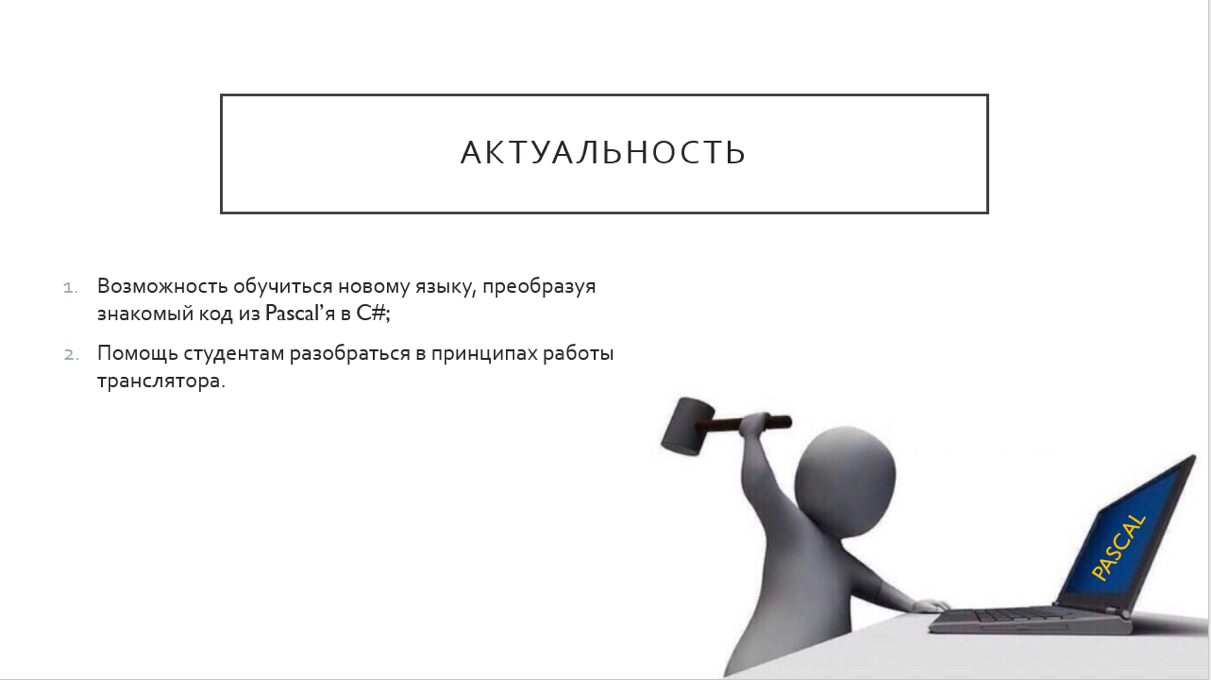


Рисунок 3 – Актуальность

На рисунке 4 демонстрируется как код транслируется из Pascal в C#.



Рисунок 4 – Принцип работы

Особенности разрабатываемого средства представлены на рисунке 5.



Рисунок 5 – Особенности

# 5 Разработка требований к проекту

Программный продукт *Транслятор из Pascal в C#* предназначен для трансляции программного кода из языка Pascal в язык C#.

Программный продукт *Транслятор из Pascal в C#* состоит из следующих подсистем:

1. Пользовательский интерфейс;
2. Лексический анализатор;
3. Синтаксический анализатор;
4. Семантический анализатор;
5. Генератор кода.

## Требования к подсистеме «Пользовательский интерфейс»

*Требование REQ\_UI\_001*

Пользовательский интерфейс содержит два окна: окно ввода и окно вывода. Окно ввода принимает текст программы на языке Pascal. Окно вывода отображает текст программы на языке C#.

*Требование REQ\_UI\_002*

В случае некорректной работы программы в окне вывода отображается сообщение об ошибке.

*Требование REQ\_UI\_003*

При нажатии на кнопку «Выберите файл» пользователь выбирает файл формата «.txt» или «.pas», содержащий код программы на языке Pascal. Название выбранного файла отображается справа от кнопки, а его содержимое представлено в окне ввода.

*Требование REQ\_UI\_004*

При нажатии на кнопку «Download» пользователь сохраняет на устройство файл формата «.cs», содержащий код программы на языке C#. Если окно вывода пустое или содержит ошибку, кнопка неактивна.

*Требование REQ\_UI\_005*

Пользовательский интерфейс содержит кнопку «Transpile», при нажатии на которую происходит трансляция кода из Pascal в C#.

*Требование REQ\_UI\_006*

Пользовательский интерфейс содержит кнопку «Clear», при нажатии на которую происходит очистка окна ввода, очистка окна вывода, если ранее была нажать кнопка «Transpile», и в случае, если код был вставлен из файла, очищается и строка с названием файла справа от кнопки «Выберите файл».

## Требования к подсистеме «Лексический анализатор»

*Требование REQ\_LA\_001*

На вход поступает код программы на языке Pascal. На выходе получаем множество токенов, иначе, если анализатор обнаружил ошибки, то программа уведомляет об этом. Структура токена представлена на рисунке 6.

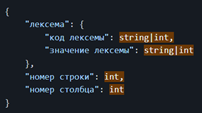


Рисунок 6 – Структура токена

## Требования к подсистеме «Синтаксический анализатор»

*Требование REQ\_SYNA\_001*

На вход поступает массив токенов, полученный в результате лексического анализа. На выходе получаем CST (конкретное синтаксическое дерево), или сообщение об ошибке, если такова была обнаружена.

## Требования к подсистеме «Семантический анализатор»

*Требование REQ\_SEMA\_001*

На вход поступает дерево разбора, полученное от синтаксического анализатора. На выходе получаем дерево, проверенное на семантические ошибки, а если таковы есть, то выводится сообщение об ошибке.

## Требования к подсистеме «Генератор кода»

*Требование REQ\_CG\_001*

На вход поступает дерево разбора, полученное от семантического анализатора. На выходе получаем код программы на языке C#.

# 6 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.

На рисунке 7 представлена диаграмма потоков данных между подсистемами разрабатываемого транслятора. На вход интерфейс принимает код на языке Pascal или файл, содержащий код на языке Pascal. 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 7 – Диаграмма потов данных

# 7 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).

## Метрика эффективности процесса производства

* c (completed tests) – покрытые тесты;
* t (time) – время одного рабочего цикла.
* p (planed tests) – запланированные тесты.
* E (efficiency) – эффективность процесса производства. Единица измерения – тесты/час.
* Сумма покрытых и запланированных тестов является работой.

Отношение проделанной работы ко времени является эффективностью процесса производства.

## Метрика качества продукта

* Q (quality) – качество продукта.

Метрику можно использовать как для оценки качества всего проекта, так и для оценки кода написанного в течении одного рабочего цикла.

# 8 Разработка перечня задач проекта

# Задачи для подсистемы «Пользовательский интерфейс»

*Задача TASK\_UI\_001*

Название: сделать веб-интерфейс.

Описание: основываясь на требованиях, разработать веб-интерфейс. Минимальный функционал: окно ввода, окно вывода, кнопка для трансляции, кнопка для выбора файла, кнопка для очистки окон.

# Задачи для подсистемы «Лексический анализатор»

*Задача TASK\_LA\_001*

Название: добавить поддержку комментариев в лексер.

Описание: в паскале есть два вида комментариев:

* однострочный: //
* многострочный: { comments }

Логика должна быть примерно следующая:

* если встречаем символы //, то ищем следующий символ \n (перехода строки)
* если встречаем символ {, то ищем следующий символ }

# Задачи для подсистемы «Семантический анализатор»

*Задача TASK\_SEMA\_001*

Название: реализовать сематический анализатор

Описание: используем LL(1) грамматику и, соответственно, LL(1) парсер. Метод SyntaxAnalyzer.parse() отдает объект дерева ParseTree, которое является так называемым конкретным синтаксическим деревом. Возможно, понадобится преобразовать его в AST - Abstract Syntax Tree - абстрактное синтаксическое дерево.

# Задачи для подсистемы «Генератор кода»

*Задача TASK\_CG\_001*

Название: сделать вывод C# кода в файл.

Описание: при запуске transpiler вместо принта полученного кода писать его в файл с тем же названием и класть в examples. Добавить в .gitignore .cs файлы.

*Задача TASK\_CG\_002*

Название: реализовать генератор кода.

Описание: сделать таблицу сопоставления терминалов, покрыть тестами код, содержащий циклы, условия, присвоение значений переменным, функции. Собрать итоговый код на C#. В семантическом анализаторе хранить vars\_dict, где ключи будут областями видимости, состоящими из цифры области видимости и uuid. В генераторе кода надо хранить текущую область видимости, словарь, в котором ключи будут цифрами области видимости, которые уже встречались, а значения сколько раз уже встречалась эта область видимости. Инкрементируем current\_scope, смотрим в словарь с ключом по current\_scope и инкрементируем значение.

# Задачи для системы в целом

*Задача TASK\_SYS\_001*

Название: собрать образ docker.

Описание: для взаимодействия с веб-интерфейсом нужен контейнер, содержащий в себе все элементы транслятора. Предоставить возможность увидеть интерфейс на 8000 порту.

# 9 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.

## Перечень запретов, рекомендаций и требований для оформления исходных кодов проекта

* Код написан по стандарту PEP8. Наибольшее внимание следует уделить следующим пунктам из этого стандарта:
  + Названия функций состоят только из строчных букв, если название функции составное к нему применяется стиль написания Snake case;
  + Названия классов должны начинаться с заглавной буквы, если название класса составное к нему применяется стиль написания Pascal case;
  + Все константы должны быть записываться заглавными буквами, если название константы составное, то слова должны разделять символами подчеркивания.
* Именовать переменные и функции в соответствии с их предназначением;
* Любая новая написанная функция должна сопровождаться новым тестом для этой функции;
* Перед однострочным комментарием должна быть одна пустая строка;
* Однострочные комментарии должны иметь один пробел перед знаком # и самим комментарием.

## Чек-лист для проверки корректности кода

* Написанный код проходит все тесты;
* Названия новых переменных и функций отображают их содержание;
* Код написан по стандарту PEP8;
* Любая новая написанная функция должна сопровождаться новым тестом для этой функции;
* Перед однострочным комментарием есть одна пустая строка;
* Однострочные комментарии имеют один пробел перед знаком # и самим комментарием;
* Flake8 не выдал ошибок.

# 10 Разработка плана тестирования проекта

## Тесты для тестирования подсистемы «Пользовательский интерфейс»

*Тест TEST\_UI\_001*

Тестируемые требования: *REQ\_UI\_001*.

Описание теста: нажать на окно ввода, начать вводить любой текст с клавиатуры.

Ожидаемый результат: печатаемый текст отображается в окне ввода.

*Тест TEST\_UI\_002*

Тестируемые требования: *REQ\_UI\_001.*

Описание теста: нажать на окно вывода, начать вводить любой текст с клавиатуры.

Ожидаемый результат: печатаемый текст не отображается в окне вывода.

*Тест TEST\_UI\_003*

Тестируемые требования: *REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_005, REQ\_SYNA\_001*.

Описание теста: нажать на окно ввода, ввести «test», нажать на кнопку «Transpile».

Ожидаемый результат: в окне вывода отобразилось сообщение об ошибке вида «SyntaxError: test at line 1. Expected ' START '».

*Тест TEST\_UI\_004*

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

Описание теста: нажать на кнопку «Выберите файл», выбрать любой файл формата «.txt» или «.pas».

Ожидаемый результат: в окне ввода отобразилось содержимое выбранного файла, справа от кнопки «Выберите файл» написано название файла.

*Тест TEST\_UI\_005*

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

Описание теста: нажать на окно ввода, ввести «test», нажать на кнопку «Clear».

Ожидаемый результат: окно ввода очищено.

## Тесты для тестирования подсистемы «Лексический анализатор»

*Тест TEST\_LA\_001*

Тестируемые требования: *REQ\_LA\_001, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004, REQ\_UI\_005*.

Описание теста:

1. Нажать на окно ввода.
2. Ввести

begin  
{  
var4: string := 'some string1';  
end.

1. Нажать на кнопку «Transpile».

Ожидаемый результат:

1. Окно вывода содержит сообщение о лексической ошибке вида «UnexpectedTokenError: { at line 2».
2. Кнопка «Download» недоступна для нажатия.

*Тест TEST\_LA\_002*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_CG\_001, REQ\_UI\_001, REQ\_UI\_004, REQ\_UI\_005*.

Описание теста:

1. Нажать на окно ввода.
2. Ввести

begin  
println('Hello, World');  
end.

1. Нажать на кнопку «Transpile».

Ожидаемый результат:

1. Окно вывода содержит код на языке C#

using System;  
  
namespace Transpiler  
{  
 internal class Program  
 {  
  
 public static void Main(string[] args)  
 {  
 Console.WriteLine("Hello, World");  
 }  
  
 }  
}

1. Кнопка «Download» доступна для нажатия.

## Тесты для тестирования подсистемы «Синтаксический анализатор»

*Тест TEST\_SYNA\_001*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004, REQ\_UI\_005.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

beg$in  
end.

1. Нажать на кнопку «Transpile»

Ожидаемый результат:

1. Окно вывода содержит сообщение о синтаксической ошибке вида «SyntaxError: beg at line 1. Expected ' START '».
2. Кнопка «Download» недоступна для нажатия.

*Тест TEST\_SYNA\_002*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004, REQ\_UI\_005.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

begin  
println('Hello, World')  
end.

1. Нажать на кнопку «Transpile».

Ожидаемый результат:

1. Окно вывода содержит сообщение о синтаксической ошибке вида «SyntaxError: end at line 3. Expected 'semicolon'».
2. Кнопка «Download» недоступна для нажатия.

*Тест TEST\_SYNA\_003*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_CG\_001, REQ\_UI\_001, REQ\_UI\_004, REQ\_UI\_005.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

var test: boolean := true;  
begin  
if (2 < 5) or (test) then  
begin  
 test := false;  
end;  
end.

1. Нажать на кнопку «Transpile».

Ожидаемый результат:

1. Окно вывода содержит код на языке C#

using System;  
  
namespace Transpiler  
{  
 internal class Program  
 {  
 static bool test = true;  
  
 public static void Main(string[] args)  
 {  
 if ((2 < 5) || (test))  
 {  
 test = false;  
 }  
 }  
  
 }  
}

1. Кнопка «Download» доступна для нажатия.

## Тесты для тестирования подсистемы «Семантический анализатор»

*Тест TEST\_SEMA\_001*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004, REQ\_UI\_005.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

var test: boolean := true;  
begin  
test := 2;  
end.

1. Нажать на кнопку «Transpile».

Ожидаемый результат:

1. Окно вывода содержит сообщение о семантической ошибке вида «SemanticError: 2 at line 3 - expression is not compatible with type Boolean».
2. Кнопка «Download» недоступна для нажатия.

*Тест TEST\_ SEMA \_002*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004, REQ\_UI\_005.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

begin  
var a: integer := 15;  
var a: integer;  
end.

1. Нажать на кнопку «Transpile».

Ожидаемый результат:

1. Окно вывода содержит сообщение о семантической ошибке вида «SemanticError: a at line 3 - variable is already defined».
2. Кнопка «Download» недоступна для нажатия.

## Тесты для тестирования подсистемы «Генератор кода»

*Тест TEST\_CG\_001*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_CG\_001, REQ\_UI\_001, REQ\_UI\_004, REQ\_UI\_005.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

var a: real := 2.0;  
begin  
var b: real := a \* a;  
println(b);  
end.

1. Нажать на кнопку «Transpile».

Ожидаемый результат:

1. Окно вывода содержит код на языке C#

using System;  
  
namespace Transpiler  
{  
 internal class Program  
 {  
 static double a = 2.0;  
  
 public static void Main(string[] args)  
 {  
 double b = a \* a;  
 Console.WriteLine(b);  
 }  
  
 }  
}

1. Кнопка «Download» доступна для нажатия.

## Тесты для тестирования системы в целом

*Тест TEST\_SYS\_001*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_CG\_001, REQ\_UI\_001, REQ\_UI\_004, REQ\_UI\_005, REQ\_UI\_006.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

begin  
println('Hello, World');  
end.

1. Нажать на кнопку «Transpile».
2. Нажать на кнопку «Download» и сохранить файл.
3. Нажать на кнопку «Clear».

Ожидаемый результат:

1. Окно вывода содержит код на языке C#

using System;  
  
namespace Transpiler  
{  
 internal class Program  
 {  
  
 public static void Main(string[] args)  
 {  
 Console.WriteLine("Hello, World");  
 }  
  
 }  
}

1. Файл сохранен на устройство.
2. После нажатия на кнопку «Clear» будет очищено окно ввода и вывода.

*Тест TEST\_SYS\_002*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_CG\_001, REQ\_UI\_001, REQ\_UI\_003, REQ\_UI\_005, REQ\_UI\_006.*

Описание теста:

1. Нажать на кнопку «Выберите файл» и выбрать файл формата «.txt» или «.pas».
2. Ввести

// условное содержание файла  
begin  
println('Hello, World');  
end.

1. Нажать на кнопку «Transpile».
2. Нажать на кнопку «Clear».

Ожидаемый результат:

1. После выбора файла окно ввода содержит код на языке Pascal и рядом с кнопкой «Выберите файл» отображено название выбранного файла.
2. Окно вывода содержит код на языке C

using System;  
  
namespace Transpiler  
{  
 internal class Program  
 {  
  
 public static void Main(string[] args)  
 {  
 Console.WriteLine("Hello, World");  
 }  
  
 }  
}

1. После нажатия на кнопку «Clear» будет очищено окно ввода, окно вывода и строка, содержащее название выбранного файла.

*Тест TEST\_SYS\_003*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004, REQ\_UI\_005.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

begin  
var a: integer := 15;  
var a: integer;  
end.

1. Нажать на кнопку «Transpile».
2. Нажать на кнопку «Download» и сохранить файл.

Ожидаемый результат:

1. Окно вывода содержит сообщение о семантической ошибке вида «SemanticError: a at line 3 - variable is already defined».
2. Кнопка «Download» недоступна для нажатия.

## Матрица покрытия тестами требований

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

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

Данный тестовый артефакт является неотъемлемой частью тестирования.

В соответствии с написанными требованиями и тестами на рисунке 8 представлена матрица покрытия тестами требований.

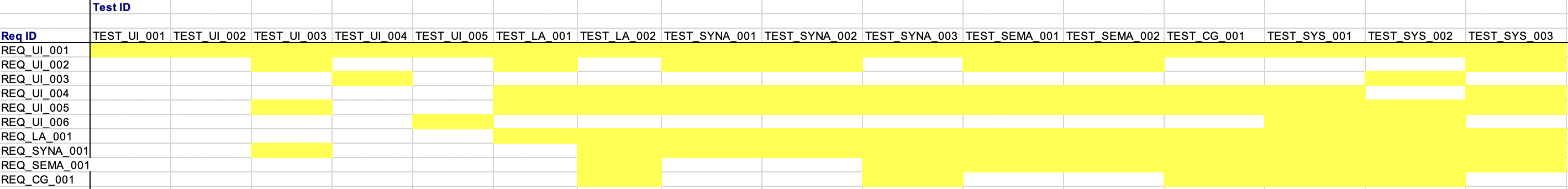


Рисунок 8 – Матрица покрытия тестами требований

# 

# 11 Тестирование проекта

## Тесты для тестирования подсистемы «Пользовательский интерфейс»

*Тест TEST\_UI\_001*

Тестируемые требования: *REQ\_UI\_001*.

Описание теста: нажать на окно ввода, начать вводить любой текст с клавиатуры.

Ожидаемый результат: печатаемый текст отображается в окне ввода.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

*Тест TEST\_UI\_002*

Тестируемые требования: *REQ\_UI\_001.*

Описание теста: нажать на окно вывода, начать вводить любой текст с клавиатуры.

Ожидаемый результат: печатаемый текст не отображается в окне вывода.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

*Тест TEST\_UI\_003*

Тестируемые требования: *REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_005, REQ\_SYNA\_001*.

Описание теста: нажать на окно ввода, ввести «test», нажать на кнопку «Transpile».

Ожидаемый результат: в окне вывода отобразилось сообщение об ошибке вида «SyntaxError: test at line 1. Expected ' START '».

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

*Тест TEST\_UI\_004*

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

Описание теста: нажать на кнопку «Выберите файл», выбрать любой файл формата «.txt» или «.pas».

Ожидаемый результат: в окне ввода отобразилось содержимое выбранного файла, справа от кнопки «Выберите файл» написано название файла.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

*Тест TEST\_UI\_005*

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

Описание теста: нажать на окно ввода, ввести «test», нажать на кнопку «Clear».

Ожидаемый результат: окно ввода очищено.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

## Тесты для тестирования подсистемы «Лексический анализатор»

*Тест TEST\_LA\_001*

Тестируемые требования: *REQ\_LA\_001, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004, REQ\_UI\_005*.

Описание теста:

1. Нажать на окно ввода.
2. Ввести

begin  
{  
var4: string := 'some string1';  
end.

1. Нажать на кнопку «Transpile».

Ожидаемый результат:

1. Окно вывода содержит сообщение о лексической ошибке вида «UnexpectedTokenError: { at line 2».
2. Кнопка «Download» недоступна для нажатия.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

*Тест TEST\_LA\_002*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_CG\_001, REQ\_UI\_001, REQ\_UI\_004, REQ\_UI\_005*.

Описание теста:

1. Нажать на окно ввода.
2. Ввести

begin  
println('Hello, World');  
end.

1. Нажать на кнопку «Transpile».

Ожидаемый результат:

1. Окно вывода содержит код на языке C#

using System;  
  
namespace Transpiler  
{  
 internal class Program  
 {  
  
 public static void Main(string[] args)  
 {  
 Console.WriteLine("Hello, World");  
 }  
  
 }  
}

1. Кнопка «Download» доступна для нажатия.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

## Тесты для тестирования подсистемы «Синтаксический анализатор»

*Тест TEST\_SYNA\_001*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004, REQ\_UI\_005.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

beg$in  
end.

1. Нажать на кнопку «Transpile»

Ожидаемый результат:

1. Окно вывода содержит сообщение о синтаксической ошибке вида «SyntaxError: beg at line 1. Expected ' START '».
2. Кнопка «Download» недоступна для нажатия.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

*Тест TEST\_SYNA\_002*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004, REQ\_UI\_005.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

begin  
println('Hello, World')  
end.

1. Нажать на кнопку «Transpile».

Ожидаемый результат:

1. Окно вывода содержит сообщение о синтаксической ошибке вида «SyntaxError: end at line 3. Expected 'semicolon'».
2. Кнопка «Download» недоступна для нажатия.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

*Тест TEST\_SYNA\_003*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_CG\_001, REQ\_UI\_001, REQ\_UI\_004, REQ\_UI\_005.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

var test: boolean := true;  
begin  
if (2 < 5) or (test) then  
begin  
 test := false;  
end;  
end.

1. Нажать на кнопку «Transpile».

Ожидаемый результат:

1. Окно вывода содержит код на языке C#

using System;  
  
namespace Transpiler  
{  
 internal class Program  
 {  
 static bool test = true;  
  
 public static void Main(string[] args)  
 {  
 if ((2 < 5) || (test))  
 {  
 test = false;  
 }  
 }  
  
 }  
}

1. Кнопка «Download» доступна для нажатия.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

## Тесты для тестирования подсистемы «Семантический анализатор»

*Тест TEST\_SEMA\_001*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004, REQ\_UI\_005.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

var test: boolean := true;  
begin  
test := 2;  
end.

1. Нажать на кнопку «Transpile».

Ожидаемый результат:

1. Окно вывода содержит сообщение о семантической ошибке вида «SemanticError: 2 at line 3 - expression is not compatible with type Boolean».
2. Кнопка «Download» недоступна для нажатия.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

*Тест TEST\_ SEMA \_002*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004, REQ\_UI\_005.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

begin  
var a: integer := 15;  
var a: integer;  
end.

1. Нажать на кнопку «Transpile».

Ожидаемый результат:

1. Окно вывода содержит сообщение о семантической ошибке вида «SemanticError: a at line 3 - variable is already defined».
2. Кнопка «Download» недоступна для нажатия.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

## Тесты для тестирования подсистемы «Генератор кода»

*Тест TEST\_CG\_001*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_CG\_001, REQ\_UI\_001, REQ\_UI\_004, REQ\_UI\_005.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

var a: real := 2.0;  
begin  
var b: real := a \* a;  
println(b);  
end.

1. Нажать на кнопку «Transpile».

Ожидаемый результат:

1. Окно вывода содержит код на языке C#

using System;  
  
namespace Transpiler  
{  
 internal class Program  
 {  
 static double a = 2.0;  
  
 public static void Main(string[] args)  
 {  
 double b = a \* a;  
 Console.WriteLine(b);  
 }  
  
 }  
}

1. Кнопка «Download» доступна для нажатия.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

## Тесты для тестирования системы в целом

*Тест TEST\_SYS\_001*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_CG\_001, REQ\_UI\_001, REQ\_UI\_004, REQ\_UI\_005, REQ\_UI\_006.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

begin  
println('Hello, World');  
end.

1. Нажать на кнопку «Transpile».
2. Нажать на кнопку «Download» и сохранить файл.
3. Нажать на кнопку «Clear».

Ожидаемый результат:

1. Окно вывода содержит код на языке C#

using System;  
  
namespace Transpiler  
{  
 internal class Program  
 {  
  
 public static void Main(string[] args)  
 {  
 Console.WriteLine("Hello, World");  
 }  
  
 }  
}

1. Файл сохранен на устройство.
2. После нажатия на кнопку «Clear» будет очищено окно ввода и вывода.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

*Тест TEST\_SYS\_002*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_CG\_001, REQ\_UI\_001, REQ\_UI\_003, REQ\_UI\_005, REQ\_UI\_006.*

Описание теста:

1. Нажать на кнопку «Выберите файл» и выбрать файл формата «.txt» или «.pas».
2. Ввести

// условное содержание файла  
begin  
println('Hello, World');  
end.

1. Нажать на кнопку «Transpile».
2. Нажать на кнопку «Clear».

Ожидаемый результат:

1. После выбора файла окно ввода содержит код на языке Pascal и рядом с кнопкой «Выберите файл» отображено название выбранного файла.
2. Окно вывода содержит код на языке C

using System;  
  
namespace Transpiler  
{  
 internal class Program  
 {  
  
 public static void Main(string[] args)  
 {  
 Console.WriteLine("Hello, World");  
 }  
  
 }  
}

1. После нажатия на кнопку «Clear» будет очищено окно ввода, окно вывода и строка, содержащее название выбранного файла.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

*Тест TEST\_SYS\_003*

Тестируемые требования: *REQ\_LA\_001, REQ\_SYNA\_001, REQ\_SEMA\_001, REQ\_UI\_001, REQ\_UI\_002, REQ\_UI\_004, REQ\_UI\_005.*

Описание теста:

1. Нажать на окно ввода.
2. Ввести

begin  
var a: integer := 15;  
var a: integer;  
end.

1. Нажать на кнопку «Transpile».
2. Нажать на кнопку «Download» и сохранить файл.

Ожидаемый результат:

1. Окно вывода содержит сообщение о семантической ошибке вида «SemanticError: a at line 3 - variable is already defined».
2. Кнопка «Download» недоступна для нажатия.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

# Заключение

По окончанию всех работ посчитаем метрики, описанные в 7 главе. Эффективность одной рабочей сессии составила 5,25 тестов в час, а качество продукта равно 94%.

В рамках курсовой работы было разработано программное средство «Транслятор из Pascal 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 презентацию проекта;
* разработаны требования к проекту;
* разработана архитектуру проекта;
* разработаны измерения проекта;
* разработан перечь задач проекта;
* разработаны рекомендации по кодированию;
* разработан план тестирования проекта;
* протестирован проект.

Таким образом, цель данной курсовой работы была достигнута.

# Список литературы

1. Гриняк В.М. Лекции по дисциплине «Технологии коллективной промышленной разработки информационных систем». Электронный вариант.