

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 PYTHON НА JAVASCRIPT**

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  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Лемеш В. Е.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Денисенко С. Д.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ма Шибо

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ше С. В.

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

г. Владивосток  
2023

Оглавление

[1. Разработка плана проекта 3](#_Toc124872711)

[2. Разработка регламента проведения инспекции 4](#_Toc124872712)

[3. Разработка модели состояний задач 6](#_Toc124872713)

[4. Разработка презентации проекта 7](#_Toc124872714)

[5. Разработка требований к проекту 9](#_Toc124872715)

[6. Разработка архитектуры проекта 12](#_Toc124872716)

[7. Разработка программы измерений проекта 13](#_Toc124872717)

[8. Разработка перечня задач проекта 14](#_Toc124872718)

[9. Разработка рекомендаций по кодированию 15](#_Toc124872719)

[10. Разработка плана тестирования проекта 18](#_Toc124872720)

[11. Матрица покрытия тестами требований 24](#_Toc124872721)

[12. Тестирование проекта 25](#_Toc124872722)

[Заключение 30](#_Toc124872723)

[13. Список литературы 31](#_Toc124872724)

# Разработка плана проекта

Исполнители проекта:

* Team Leader – Денисенко Семён;
* Coder 1 – Денисенко Семён;
* Coder 2 – Лемеш Владислав;
* Coder 3 – Ма Шибо;
* Coder 4 – Ше Сергей;
* Build Engineer – Денисенко Семён;
* Technical Writer – Лемеш Владислав.

Перечень задач и сроки их выполнения представлены на рисунке 1.

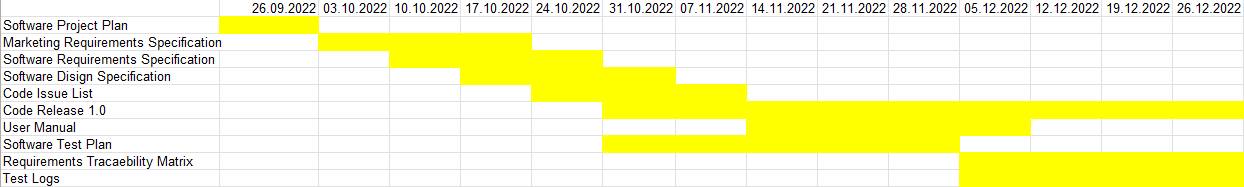


Рисунок 1 – Календарный план работы над проектом

# Разработка регламента проведения инспекции

* Критерии инспекции для различных типов рабочих продуктов

|  |  |
| --- | --- |
| **Рабочий продукт** | **Размер продукта или изменения в нем не менее** |
| Требования | 10% рабочего продукта |
| Документы по дизайну | 5% рабочего продукта |
| Не комментированный код | 20 NCLOC |
| Тесты | 25% рабочего продукта |

* Роли участников инспекции

*Обязанности автора* – оповестить председателя инспекции о готовности инспектируемого программного продукта. Провести обзорное собрание.

*Обязанности председател*я – убедиться в готовности продукта к формальной инспекции и определить её необходимость. Оповестить всех участников о необходимости проведения инспекции.

*Обязанности инспектора* – провести анализ работоспособности инспектируемого продукта.

* Участники инспекции

*Автор*: Ше Сергей, Ма Шибо.

*Инспектор*: Лемеш Владислав.

*Председатель*: Денисенко Семён.

* Этапы инспекции
  + Формальный этап
    - Оповестить команду и выбрать дату проведения
    - Провести инспекцию
    - Утвердить коллективное решение
    - Завершение инспекции
  + Неформальный этап
    - Оповестить команду
    - Провести инспекцию
* Порядок организации

Инспектируемый продукт выгружать в отдельную ветку на GitHub, информацию о командном сборе публиковать в telegram, видеосвязь по discord.

* Перечень статусов и степени важности замечаний
  + Очень важно
  + Важно
  + Выполнено
* Метрики, характеризующие эффективность инспекции

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>.

# Разработка модели состояний задач

* Перечень возможных состояний задач и их интерпретация состояний
  + TODO – задача в очереди на выполнение
  + IN ASSIGN – задача доведена до исполняющего
  + IN PROGRESS – в данный момент исполнитель занят решением задачи
  + CHECKOUT – исполнитель закончил задачу и нуждается в инспекции
  + DONE – задача полностью выполнена
* Правила создания новой задачи
  + Задачи (TODO, IN ASSIGN, DONE) назначает лидер команды
  + Задачи (IN PROGRESS, CHECKOUT) назначает исполнитель
* Правила перехода из состояния в состояние
  + TODO переходит в IN ASSIGN когда лидер доносит информацию до исполнителя, что ему назначена задача
  + IN ASSIGN переходит в IN PROGRESS когда исполнитель берется за выполнение задачи
  + IN PROGRESS переходит в CHECKOUT когда исполнитель закончил задачу и нуждается в проведении инспекции
  + CHECOUT переходит в DONE в случае когда была проведена инспекция с результатом положительно
  + CHECKOUT переходит в IN PROGRESS в случае когда была проведена инспекция с результатом отрицательно

# Разработка презентации проекта

Первый слайд презентации содержит логотип и название web-приложения (рисунок 2)

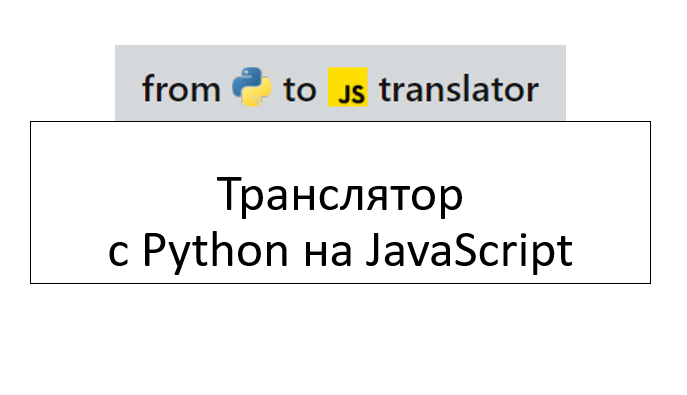


Рисунок 2 - Первый слайд

Второй слайд содержит краткое описание проекта(рисунок 3)

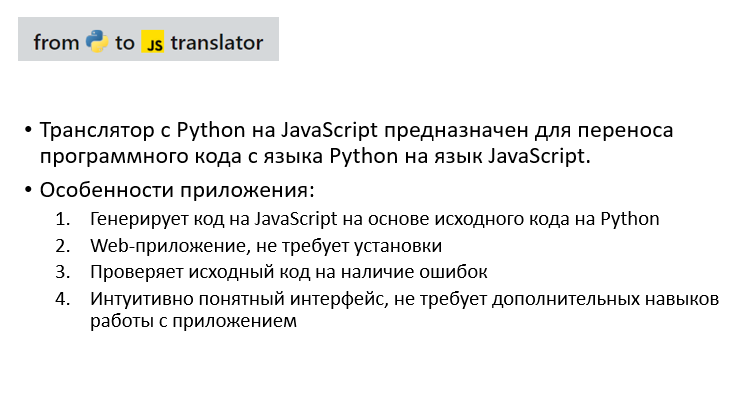


Рисунок 3 - второй слайд

На третьем слайде представлен интерфейс приложения и инструкция для пользователя(рисунок 4)

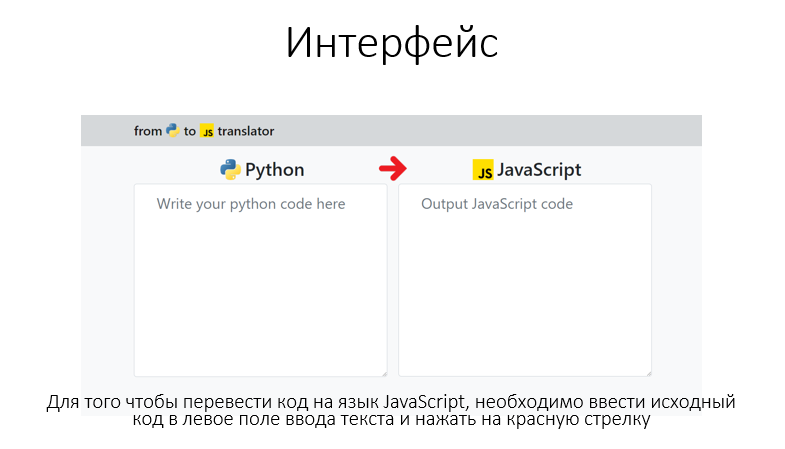


Рисунок 4 - третий слайд

# Разработка требований к проекту

Программный продукт *Транслятор Python -> JavaScript* предназначен для трансляции (перевода) с языка программирования Python на язык JavaScript.

Программный продукт *Транслятор Python -> JavaScript* 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 продукта должен содержать:

* Поле ввода исходного кода на языке Python
* Поле вывода транслированного кода на языке JavaScript
* Кнопка трансляции языка
* Поле вывода синтаксического дерева
* Поле вывода токенов

Требование REQ\_UI\_002

При допущении пользователем синтаксической ошибки, при написании в поле ввода исходного кода и нажатии на кнопку трансляции, над кнопкой должно выводиться сообщение об ошибке.

Требование REQ\_UI\_003

Кнопка трансляции языка должна иметь вид стрелки (См. рисунок 1), направленной в сторону поля вывода и располагаться между полями ввода и вывода.



Рисунок 5

Требование REQ\_UI\_004

Поля ввода и вывода исходного и транслированного языка соответственно, должны располагаться на одном уровне, причем поле ввода должно находиться левее.

Требование REQ\_UI\_005

Логотип приложения (См. рисунок 2) должен располагаться в левом верхнем углу страницы сайта.



Рисунок 6

**Требования к подсистеме «Лексический анализатор»**

Требование REQ\_LA\_001

На вход подсистеме поступает исходный код на языке Python. На выход подсистемы поступает множество токенов, либо ошибка.

Требование REQ\_LA\_002

Каждый токен представляет собой четверку <тип, значение, номер строки, номер столбца>.

**Требования к подсистеме «Синтаксический анализатор»**

Требование REQ\_SYNTA\_001

Синтаксический анализатор получает на вход массив токенов от лексического анализатора. На его основе строится абстрактное синтаксическое дерево разбора, если синтаксических ошибок нет, иначе – выдаётся ошибка.

**Требования к подсистеме «Семантический анализатор»**

Требование REQ\_SEMANTA\_001

Семантический анализатор получает на вход дерево разбора от синтаксического анализатора и проверяет его на семантические ошибки.

**Требования к подсистеме «Генератор кода»**

Требование REQ\_GEN\_001

Генератор кода получает на вход дерево разбора от синтаксического анализатора. На выходе получаем код программы на языке JavaScript.

# Разработка архитектуры проекта

Программный продукт *Транслятор Python -> JavaScript* состоит из следующих подсистем:

* Пользовательский интерфейс
* Лексический анализатор
* Синтаксический анализатор
* Семантический анализатор
* Генератор кода

Диаграмма потоков данных:

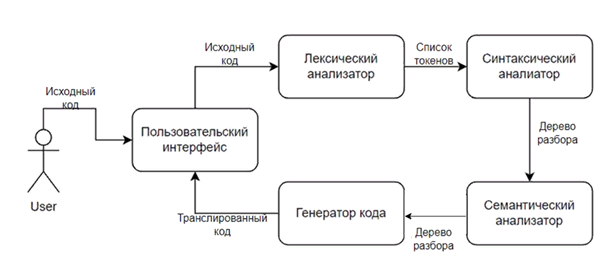
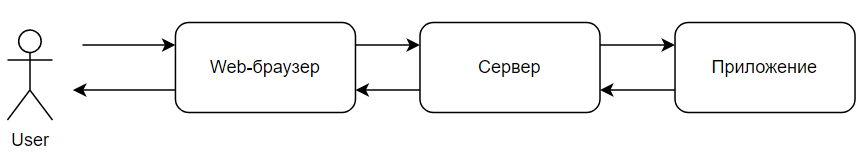


Схема работы пользовательского интерфейса:



# Разработка программы измерений проекта

**Метрики для оценки эффективности процесса разработки**

project\_size – размер разрабатываемого, сумма количества документов дизайна, количества требований, количества тестов и количества строк кода без комментариев.

error\_count – общее количество найденных ошибок в процессе разработки.

errors – общее количество найденных ошибок на этапе разработки

missed\_errors общее количество не найденных ошибок на этапе разработки

**Метрика качества программного продукта**

c\_req – количество требований, e\_req – количество требований в которых допущена ошибка.

c\_design – количество документов дизайна (диаграмм, схем и т.д.), e\_design – количество документов дизайна в которых допущена ошибка.

e\_coding – количество ошибок допущенных в коде.

c\_test – количество тестов, e\_test -количество тестов в которых допущена ошибка.

Каждое из слагаемых может выступать в роли отдельной метрики.

# Разработка перечня задач проекта

**Подсистема «Пользовательский интерфейс»**

**Task\_UI\_1**

Реализовать интерфейс поля ввода-вывода.

Реализовать часть интерфейса, содержащую поле ввода и поле вывода, программа, записанная в поле ввода при нажатии на кнопку трансляции языка, передаётся на вход лексическому анализатору. В поле вывода записывается результат работы функции генерации кода

**Task\_UI\_2**

Реализовать кнопку трансляции кода.

Реализовать часть интерфейса, содержащую кнопку трансляции кода, при нажатии на которую, содержимое поля ввода передаётся на вход функции лексического анализатора.

**Подсистема «Лексический анализатор»**

**Task\_tokenizing**

Реализовать лексический анализатор.

Необходимо реализовать функцию, на вход которой подаётся исходный код, из исходного кода выделяются токены, которые представляют собой структуру <тип токена, значение токена, строка, столбец>. На выходе - упорядоченный список токенов.

**Подсистема «Синтаксический анализатор»**

**Task\_parser**

Реализовать синтаксический анализатор.

Необходимо реализовать функцию, на вход которой подаётся упорядоченный список токенов (результат работы лексического анализатора), на основе которого формируется синтаксического дерево и подаётся на выход.

**Подсистема «Семантический анализатор»**

**Task\_semantic**

Реализовать семантический анализатор.

Необходимо реализовать функцию, на вход которой подаётся синтаксическое дерево. Производится проверка областей видимости переменных и синтаксических конструкций. На выходе сообщение о том, что всё корректно или сообщение об ошибке.

**Подсистема «Генератор кода»**

**Task\_code\_gen**

Реализовать генератор кода.

Реализовать функцию, на вход которой поступает дерево синтаксического разбора. На выходе код на целевом языке.

# Разработка рекомендаций по кодированию

***Правила написания кода на ЯП Python***

При написании кода на языке программирования python вы должны руководствоваться стандартом написания кода PEP8. Этот документ описывает соглашение о том, как писать код для языка python, включая стандартную библиотеку, входящую в состав python.

Ключевая идея такова: код читается намного больше раз, чем пишется. Собственно, рекомендации о стиле написания кода направлены на то, чтобы улучшить читаемость кода и сделать его согласованным между большим числом проектов.

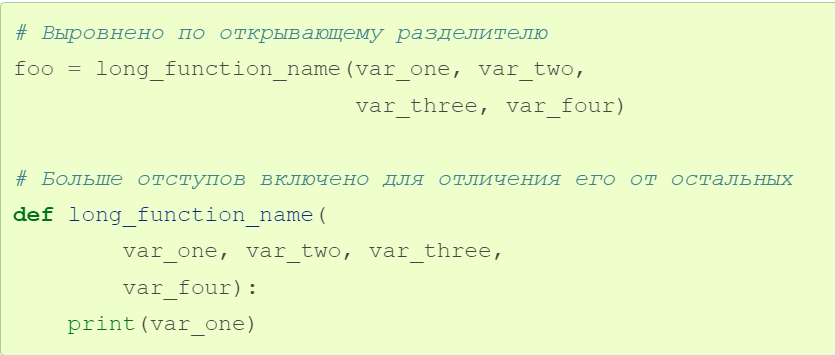
Ниже перечислены некоторые пункты, для полного ознакомления вам необходимо прочитать полное руководство по написанию кода на python.

***Внешний вид кода***

**Отступы**

Используйте 4 пробела на каждый уровень отступа. Продолжительные строки должны выравнивать обернутые элементы либо вертикально, используя неявную линию в скобках (круглых, квадратных или фигурных), либо с использованием висячего отступа. При использовании висячего отступа следует применять следующие соображения: на первой линии не должно быть аргументов, а остальные строки должны четко восприниматься как продолжение линии.

Правильно



Неправильно

Изображение выглядит как текст

Автоматически созданное описание

**Табуляция или пробелы?**

Разрешается использовать либо пробелы, либо табуляцию, но Python 3 запрещает смешивание табуляции и пробелов в отступах. Поэтому договоримся использовать только табуляцию.

**Максимальная длина строки**

Ограничьте длину строки максимум 79 символами. Для более длинных блоков текста с меньшими структурными ограничениями (строки документации или комментарии), длину строки следует ограничить 72 символами. Ограничение необходимой ширины окна редактора позволяет иметь несколько открытых файлов бок о бок, и хорошо работает при использовании инструментов анализа кода, которые предоставляют две версии в соседних столбцах.

**Пустые строки**

Отделяйте функции верхнего уровня и определения классов двумя пустыми строками. Определения методов внутри класса разделяются одной пустой строкой. Дополнительные пустые строки возможно использовать для разделения различных групп похожих функций. Пустые строки могут быть опущены между несколькими связанными однострочниками (например, набор фиктивных реализаций). Используйте пустые строки в функциях, чтобы указать логические разделы.

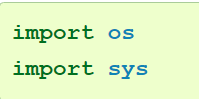
**Кодировка исходного файла**

Кодировка Python должна быть UTF-8.

**Импорты**

Каждый импорт, как правило, должен быть на одной строке.

Правильно



Неправильно

Изображение выглядит как текст

Автоматически созданное описание

В то же время можно писать так:

Изображение выглядит как текст

Автоматически созданное описание

Импорты всегда помещаются в начале файла, сразу после комментариев к модулю и строк документации, и перед объявлением констант. Импорты должны быть сгруппированы в следующем порядке:

* импорты из стандартной библиотеки
* импорты сторонних библиотек
* импорты модулей текущего проекта

Вставляйте пустую строку между каждой группой импортов.

Рекомендуется абсолютное импортирование, так как оно обычно более читаемо и ведет себя лучше (или, по крайней мере, даёт понятные сообщения об ошибках).

**Именование переменных и функций**

При именовании переменных и функций следует руководствоваться исключительно назначением соответствующей переменной или функцией. При именовании использовать прописные латинские буквы в формате snake\_case. Если функция используется для получения какого-то значения использовать префикс get, если устанавливает какое-то значение – set. Для объявления констант использовать UPPER\_SNAKE\_CASE.

**Именование классов**

Для именования классов использовать CapWords, классы именовать в единственном числе.

# Разработка плана тестирования проекта

**Тесты для 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\_003.

Описание теста: нажать на окно ввода, ввести код на языке Python, например print("hello"), нажать на кнопку трансляции языка.

Ожидаемый результат: В окне вывода отобразился код на JavaScript console.log("hello");

**Тесты для подсистемы «Лексический анализатор»**

Тест TEST\_LA\_001

Тестируемые требования: REQ\_LA\_001, REQ\_UI\_001, REQ\_UI\_002.

Описание теста:

1. нажать на окно ввода.
2. Ввести

{

a=5

1. Нажать на кнопку трансляции языка.

Ожидаемый результат: окно вывода содержит сообщение о лексической ошибке вида <class ‘SyntaxError'>: unexpected '{' on line 1

Тест TEST\_LA\_002

Тестируемые требования: REQ\_LA\_001, REQ\_SYNTA\_001, REQ\_SEMANTA\_001, REQ\_GEN\_001, REQ\_UI\_001, ,

Описание теста:

1. Нажать на окно ввода
2. Ввести print("hello, world")
3. Нажать на кнопку трансляции языка.

Ожидаемый результат: Окно вывода содержит код на языке JavaScript console.log("hello, world")

Тест TEST\_LA\_003

Тестируемые требования: REQ\_LA\_001, REQ\_LA\_001, REQ\_SYNTA\_001, REQ\_SEMANTA\_001, REQ\_GEN\_001, REQ\_UI\_001.

Описание теста:

1. Перейти в файл code/test.py
2. Ввести

while 4 + 4:

if True:

a = 0

else:

while True:

a = 0

1. Запустить тестовый файл.

Ожидаемый результат: вывод токенов в консоли.

Token(type='NEWLINE', value='\n', line=2, column=0)

Token(type='BLOCK', value='while', line=2, column=0)

Token(type='NUMBER', value='4', line=2, column=6)

Token(type='OPERATOR', value='+', line=2, column=8)

Token(type='NUMBER', value='4', line=2, column=10)

Token(type='COLON', value=':', line=2, column=11)

Token(type='NEWLINE', value='\n', line=3, column=12)

Token(type='TAB', value=' ', line=3, column=0)

Token(type='BLOCK', value='if', line=3, column=4)

Token(type='BOOL', value='True', line=3, column=7)

Token(type='COLON', value=':', line=3, column=11)

Token(type='NEWLINE', value='\n', line=4, column=12)

Token(type='TAB', value=' ', line=4, column=0)

Token(type='TAB', value=' ', line=4, column=4)

Token(type='VARIABLE', value='a', line=4, column=8)

Token(type='ASSIGN', value='=', line=4, column=10)

Token(type='NUMBER', value='0', line=4, column=12)

Token(type='NEWLINE', value='\n', line=5, column=13)

Token(type='TAB', value=' ', line=5, column=0)

Token(type='BLOCK', value='else', line=5, column=4)

Token(type='COLON', value=':', line=5, column=8)

Token(type='NEWLINE', value='\n', line=6, column=9)

Token(type='TAB', value=' ', line=6, column=0)

Token(type='TAB', value=' ', line=6, column=4)

Token(type='BLOCK', value='while', line=6, column=8)

Token(type='BOOL', value='True', line=6, column=14)

Token(type='COLON', value=':', line=6, column=18)

Token(type='NEWLINE', value='\n', line=7, column=19)

Token(type='TAB', value=' ', line=7, column=0)

Token(type='TAB', value=' ', line=7, column=4)

Token(type='TAB', value=' ', line=7, column=8)

Token(type='VARIABLE', value='a', line=7, column=12)

Token(type='ASSIGN', value='=', line=7, column=14)

Token(type='NUMBER', value='0', line=7, column=16)

Token(type='NEWLINE', value='\n', line=8, column=17)

Token(type='NEWLINE', value='\n', line=9, column=0)

**Тесты для подсистемы «Синтаксический анализатор»**

Тест TEST\_SYNTA\_001

Тестируемые требования: REQ\_LA\_001, REQ\_SYNTA\_001, REQ\_UI\_001, REQ\_UI\_002.

Описание теста:

1. Нажать на окно ввода
2. Ввести priшnt(4)
3. Нажать на кнопку трансляции языка

Ожидаемый результат: окно вывода содержит сообщение о синтаксической ошибке вида <class 'SyntaxError'>: unexpected 'ш' on line 1

Тест TEST\_SYNTA\_002

Тестируемые требования: REQ\_LA\_001, REQ\_SYNTA\_001, REQ\_UI\_001, REQ\_UI\_002.

Описание теста:

1. Нажать на окно ввода
2. Ввести

if if True:

print("yes")

1. Нажать на кнопку трансляции языка.

Ожидаемый результат: Окно вывода содержит сообщение о синтаксической ошибке вида <class 'SyntaxError'>: expected NUMBER on position 1

Тест TEST\_SYNTA\_003

Тестируемые требования: REQ\_LA\_001, REQ\_SYNTA\_001, REQ\_SEMANTA\_001, REQ\_GEN\_001, REQ\_UI\_001.

Описание теста:

1. Нажать на окно ввода
2. Ввести

a=5

b=5

if a==b and b==5:

print("Yes")

1. Нажать на кнопку трансляции языка Ожидаемый результат: Окно вывода содержит код на языке JavaScript

var a, b;

a = 5;

b = 5;

if (a === b && b === 5) {

console.log("Yes");

}

**Тесты для подсистемы «Генератор кода»**

Тест TEST\_GEN\_001

Тестируемые требования: REQ\_LA\_001, REQ\_SYNTA\_001, REQ\_SEMANTA\_001, REQ\_GEN\_001, REQ\_UI\_001.

Описание теста:

1. Нажать на окно ввода

Ввести

a=5

b=6

c=a+b

print(c)

1. Нажать на кнопку трансляции языка

Ожидаемый результат: Окно вывода содержит код на языке JavaScript

var a, b, c;

a = 5;

b = 6;

c = a + b;

console.log(c);

# Матрица покрытия тестами требований

Это двумерная таблица, содержащая соответствие функциональных требований (functional requirements) продукта и подготовленных тестовых сценариев (test cases). В заголовках колонок таблицы расположены требования, а в заголовках строк — тестовые сценарии. На пересечении — отметка, означающая, что требование текущей колонки покрыто тестовым сценарием текущей строки.

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).

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

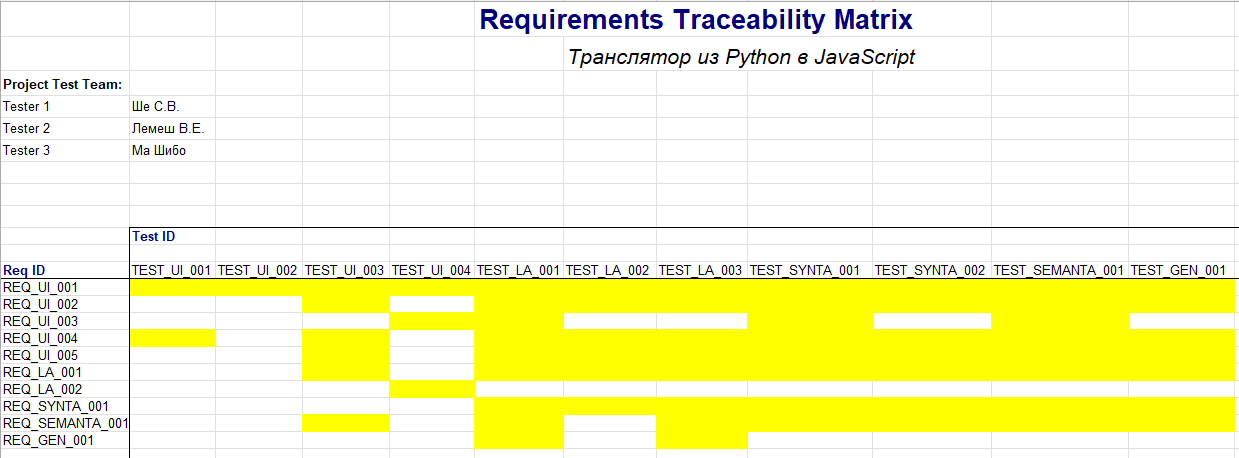


Рисунок 7-Матрица покрытия тестами требований

# Тестирование проекта

**Тесты для подсистемы «Лексический анализатор»**

Тест TEST\_LA\_001

Тестируемые требования: REQ\_LA\_001, REQ\_UI\_001, REQ\_UI\_002.

Описание теста:

1. нажать на окно ввода.
2. Ввести

{

a=5

1. Нажать на кнопку трансляции языка.

Ожидаемый результат: окно вывода содержит сообщение о лексической ошибке вида <class SyntaxError'>: unexpected '{' on line 1

Видимый результат: полностью совпадает с ожидаемым.

Резюме: Тест пройден.

Тест TEST\_LA\_002

Тестируемые требования: REQ\_LA\_001, REQ\_SYNTA\_001, REQ\_SEMANTA\_001, REQ\_GEN\_001, REQ\_UI\_001

Описание теста:

1. Нажать на окно ввода
2. Ввести print("hello, world")
3. Нажать на кнопку трансляции языка.

Ожидаемый результат: Окно вывода содержит код на языке JavaScript console.log("hello, world")

Видимый результат: полностью совпадает с ожидаемым.

Резюме: Тест пройден.

Тест TEST\_LA\_003

Тестируемые требования: REQ\_LA\_001, REQ\_LA\_001, REQ\_SYNTA\_001, REQ\_SEMANTA\_001, REQ\_GEN\_001, REQ\_UI\_001.

Описание теста:

1. Перейти в файл code/test.py
2. Ввести

while 4 + 4:

if True:

a = 0

else:

while True:

a = 0

1. Запустить тестовый файл.

Ожидаемый результат: вывод токенов в консоли.

Token(type='NEWLINE', value='\n', line=2, column=0)

Token(type='BLOCK', value='while', line=2, column=0)

Token(type='NUMBER', value='4', line=2, column=6)

Token(type='OPERATOR', value='+', line=2, column=8)

Token(type='NUMBER', value='4', line=2, column=10)

Token(type='COLON', value=':', line=2, column=11)

Token(type='NEWLINE', value='\n', line=3, column=12)

Token(type='TAB', value=' ', line=3, column=0)

Token(type='BLOCK', value='if', line=3, column=4)

Token(type='BOOL', value='True', line=3, column=7)

Token(type='COLON', value=':', line=3, column=11)

Token(type='NEWLINE', value='\n', line=4, column=12)

Token(type='TAB', value=' ', line=4, column=0)

Token(type='TAB', value=' ', line=4, column=4)

Token(type='VARIABLE', value='a', line=4, column=8)

Token(type='ASSIGN', value='=', line=4, column=10)

Token(type='NUMBER', value='0', line=4, column=12)

Token(type='NEWLINE', value='\n', line=5, column=13)

Token(type='TAB', value=' ', line=5, column=0)

Token(type='BLOCK', value='else', line=5, column=4)

Token(type='COLON', value=':', line=5, column=8)

Token(type='NEWLINE', value='\n', line=6, column=9)

Token(type='TAB', value=' ', line=6, column=0)

Token(type='TAB', value=' ', line=6, column=4)

Token(type='BLOCK', value='while', line=6, column=8)

Token(type='BOOL', value='True', line=6, column=14)

Token(type='COLON', value=':', line=6, column=18)

Token(type='NEWLINE', value='\n', line=7, column=19)

Token(type='TAB', value=' ', line=7, column=0)

Token(type='TAB', value=' ', line=7, column=4)

Token(type='TAB', value=' ', line=7, column=8)

Token(type='VARIABLE', value='a', line=7, column=12)

Token(type='ASSIGN', value='=', line=7, column=14)

Token(type='NUMBER', value='0', line=7, column=16)

Token(type='NEWLINE', value='\n', line=8, column=17)

Token(type='NEWLINE', value='\n', line=9, column=0)

Видимый результат: полностью совпадает с ожидаемым.

Резюме: Тест пройден.

**Тесты для подсистемы «Синтаксический анализатор»**

Тест TEST\_SYNTA\_001

Тестируемые требования: REQ\_LA\_001, REQ\_SYNTA\_001, REQ\_UI\_001, REQ\_UI\_002.

Описание теста:

1. Нажать на окно ввода
2. Ввести priшnt(4)
3. Нажать на кнопку трансляции языка

Ожидаемый результат: окно вывода содержит сообщение о синтаксической ошибке вида <class 'SyntaxError'>: unexpected 'ш' on line 1

Видимый результат: полностью совпадает с ожидаемым.

Резюме: Тест пройден.

Тест TEST\_SYNTA\_002

Тестируемые требования: REQ\_LA\_001, REQ\_SYNTA\_001, REQ\_UI\_001, REQ\_UI\_002.

Описание теста:

1. Нажать на окно ввода
2. Ввести

if if True:

print("yes")

1. Нажать на кнопку трансляции языка.

Ожидаемый результат: Окно вывода содержит сообщение о синтаксической ошибке вида <class 'SyntaxError'>: expected NUMBER on position 1

Видимый результат: полностью совпадает с ожидаемым.

Резюме: Тест пройден.

Тест TEST\_SYNTA\_003

Тестируемые требования: REQ\_LA\_001, REQ\_SYNTA\_001, REQ\_SEMANTA\_001, REQ\_GEN\_001, REQ\_UI\_001.

Описание теста:

1. Нажать на окно ввода
2. Ввести

a=5

b=5

if a==b and b==5:

print("Yes")

1. Нажать на кнопку трансляции языка

Ожидаемый результат: Окно вывода содержит код на языке JavaScript

var a, b;

a = 5;

b = 5;

if (a === b && b === 5) {

console.log("Yes");

}

Видимый результат: полностью совпадает с ожидаемым.

Резюме: Тест пройден.

**Тесты для подсистемы «Генератор кода»**

Тест TEST\_GEN\_001

Тестируемые требования: REQ\_LA\_001, REQ\_SYNTA\_001, REQ\_SEMANTA\_001, REQ\_GEN\_001, REQ\_UI\_001.

Описание теста:

1. Нажать на окно ввода

Ввести

a=5

b=6

c=a+b

print(c)

1. Нажать на кнопку трансляции языка

Ожидаемый результат: Окно вывода содержит код на языке JavaScript

var a, b, c;

a = 5;

b = 6;

c = a + b;

console.log(c);

Видимый результат: полностью совпадает с ожидаемым.

Резюме: Тест пройден.

# Заключение

**Метрика оценки эффективности процесса разработки**

Строк кода – 552

Ошибок за все время разработки в коде – 24

Ошибок за се время разработки в требованиях – 2

Ошибок за все время разработки в тестах – 3

Ошибок за все время в документах дизайна – 0

Требований к проекту – 10

Документов дизайна – 2

Тестов – 9

Итого размер проекта – 552 + 10 + 2 + 9 = 573

Всего ошибок – 24 + 2 + 3 + 0 = 29

IDF = (29 / 573) = 0.05

**Метрика качества программного продукта**

Req = 2 / 10 = 0.2

Design = 0 / 2 = 0

Coding = 24 / 552 = 0.04

Test = 3 / 9 = 0.3

Q = 0.2 + 0 + 0.04 + 0.3 = 0.54

В рамках курсовой работы было разработано программное средство «Транслятор с Python на JavaScript» 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.
2. «Компиляторы: принципы, технологии и инструменты» — классический учебник по теории построения компиляторов под авторством Альфреда В. Ахо