

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**

Транслятор языка программирования PyJsik

**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. Б8119-09.03.04прогин

\_\_\_\_\_\_\_\_\_\_\_\_\_ Чуйко М. Ю.

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_ Губенко И. Г.

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_ Пархоменко А. В.

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_ Аксеныч С. Р.

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_ Туров С. Е.

(подпись)

Руководитель: В. М. Гриняк

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

(подпись)

Защищён с оценкой \_\_\_\_\_\_\_\_

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

(подпись) И.О. Фамилия

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

г. Владивосток

2022

Оглавление

[План проекта 3](#_Toc94562048)

[Регламент инспекции 3](#_Toc94562049)

[Модель состояний задач 6](#_Toc94562050)

[Презентация продукта 8](#_Toc94562051)

[Требования к проекту 10](#_Toc94562052)

[Требования к подсистеме «Ввода-вывода» 10](#_Toc94562053)

[Требования к подсистеме «Модуль лексического анализатора» 10](#_Toc94562054)

[Требования к подсистеме «Модуль синтаксического анализатора» 11](#_Toc94562055)

[Требования к подсистеме «Модуль кодогенирации» 11](#_Toc94562056)

[Архитектура проекта 11](#_Toc94562057)

[Метрики 14](#_Toc94562058)

[Перечень задач проекта 15](#_Toc94562059)

[Рекомендации по кодированию 17](#_Toc94562060)

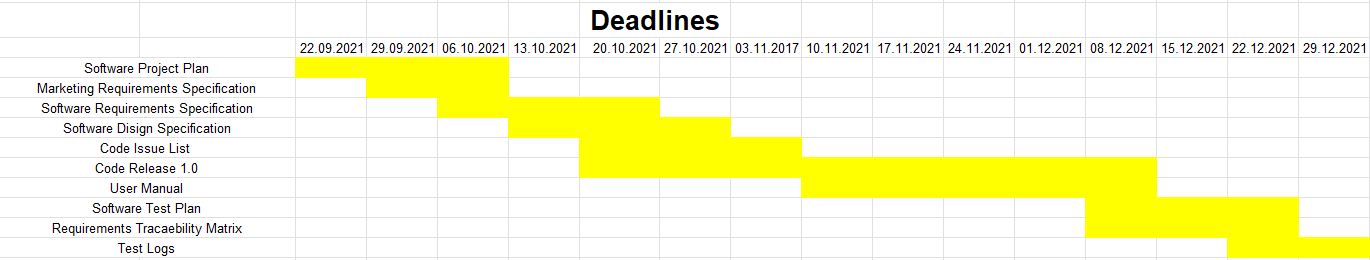
[План тестирования 20](#_Toc94562061)

[Матрица тестирования 24](#_Toc94562062)

[Заключение 25](#_Toc94562063)

# План проекта

|  |  |
| --- | --- |
| **Project Team:** |  |
| Team Leader | Пархоменко Александр |
| Coder 1 | Пархоменко Александр |
| Coder 2 | Губенко Иван |
| Coder 3 | Чуйко Михаил |
| Coder 4 | Туров Степан |
| Coder 5 | Аксёныч Семён |
| Build Engineer | Туров Степан |
| Technical Writer 1 | Губенко Иван |
| Technical Writer 2 | Чуйко Михаил |



# Регламент инспекции

1. **Критерии отнесения к формальной / неформальной инспекции для различных типов рабочих продуктов (требования, документы дизайна, код, тесты).**

Формальная инспекция проводится:

- после завершения разработки некоторого модуля;

- после полного завершения работы над проектом.

1. **Перечень ролей участников инспекции и их обязанности, число участников в зависимости от объёма рабочего продукта**

Автор – сотрудник создавший продукт или изменивший некоторую его часть.

Инспектор – сотрудник (или группа) ответственный за проведение инспекции.

Стенографист – сотрудник ответственный за ведение документации по инспекции.

Связной – сотрудник, который назначает место и уведомляет других сотрудников.

Сотрудник не может одновременно сочетать роли автора и инспектора.

1. **Этапы инспекции**

Назначение – инициация места, времени, способа проведения, целей, сроков и ролей инспекции

Проведение – непосредственно проведение инспекции группой сотрудников в соответствие с назначенным порядком, ведение документации о ходе инспекции.

Завершение – подведение итогов, рассылка документации инспекции сотрудникам.

1. **Порядок организации (кто куда что выгружает, кому что рассылает, кого приглашает и т.д.)**

Сотрудник сообщает в групповом чате о необходимости проведения инспекции. Руководитель проекта назначает время, место и способ сбора, оглашает роли. Задокументированные итоги проведённой инспекции рассылаются в групповой чат и выгружаются в github.

1. **Порядок подготовки к инспекции (сроки, время и т.п.)**

В течение суток после заявления о необходимости проведения инспекции согласно порядку организации, осуществляется подготовка и её проведение. Длительность инспекции не должна превышать одного часа.

1. **Порядок проведения инспекции (функции каждого участника, сроки, время)**

Функции автора

Автор – знакомится с порядком предоставления своего продукта, готовит продукт к инспекции, сообщает о необходимости инспекции.

Функции инспектора

Инспектор – изучить порядок проведения инспекции, требования к инспектируемому продукту, ознакомиться со списком тестов, формой отчётности инспекции. Заполнить форму отчётности и отослать её связному, прокомментировать итоги инспекции, ознакомить автора с обнаруженными недочётами.

Функции стенографиста

Стенографист – распространение среди участников инспекции форм отчётности и предоставления продукта, требований к инспектируемому продукта, проверка правильности заполненных форм, обработка результатов инспекции, заполнение формы итогов инспекции, распространение итогов инспекции согласно порядку организации.

Функции связного

Связной – получение от руководителя проекта списка участника инспекции, проверка готовности участников к инспекции, проверка выполнения ими их функций, организация встречи согласно порядку организации.

1. **Перечень статусов и степени важности замечаний**

Ошибка – проблема, найденная на стадии проведения инспекции.

Комментарий – предложение архитектурных подходов, оформления, организации модулей, функций и т. д.

1. **Порядок верификации учёта замечаний**

Коллективное исправление, если возможно во время встречи, иначе исправление сотрудником, с последующим покрытием необходимыми тестами.

1. **Метрики, характеризующие эффективность инспекций (предполагаемые)**

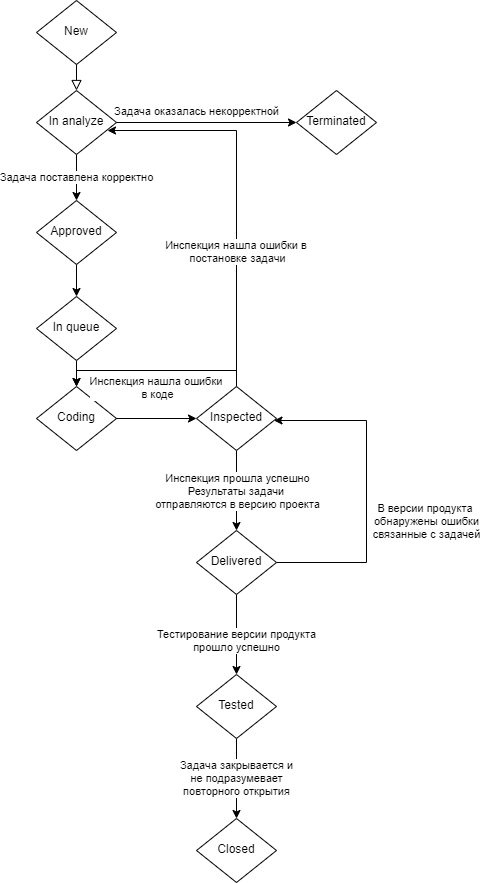
1 ошибка на 50 строк – хорошо;

2-4 на 50 строк – плохо;

Более 5 ошибок на 50 строк – …

# Модель состояний задач

1. Перечень возможных состояний задач и их интерпретация
   1. New – новая подзадача
   2. In analyze – в процессе анализа и утверждения задачи
   3. Terminated – задача не прошла этап анализа, оказалась не нужной или неполной в рамках проекта.
   4. In queue – задача одобрена, но ей не был назначен исполнитель. Она находится в очереди, пока ей не будет назначен человек.
   5. Coding – задача находится непосредственно в стадии выполнения
   6. Under inspection – результат выполнения задачи находится на стадии инспекции. Если инспекция устанавливает неточность в коде, то задача возвращается в состояние Coding. Если была обнаружена ошибка или неточность в формулировке задачи, то она возвращается в состояние In analyze.
   7. Delivered – результаты выполнения задачи прошли этап инспекции и теперь внедрён в текущий вариант продукта
   8. Tested – продукт с внедрёнными изменениями прошёл этап тестирования. Если же продукт не прошёл тестирование, то задача возвращается в стадию инспекции с информацией об обнаруженных ошибках.
   9. Closed – задача полностью выполнена и закрыта.
2. Правила создания новой задачи
   1. Производится сбор команды разработки во главе с командиром команды.
   2. Ведётся обсуждение плана работ на ближайшую рабочую неделю, распределение нагрузки. С учётом нужд команды производится декомпозиция крупных задач проект и её распределение между участниками команды
   3. После согласования распределения задач и сроков их выполнения, все распределённые задачи переходят в состояние New и сразу же попадает в состояние In analyze, так как анализ происходит сразу на сборе команды.
   4. В течение собрания задача также проходит все состояния вплоть до Coding
3. Правила перехода задачи из состояния в состояние



# Презентация продукта





# Требования к проекту

Программный продукт PyJsik предназначен для трансляции элементарного кода на языке Python в код на языке JavaScript.

Программный продукт PyJsik состоит из следующих подсистем:

1. Система ввода-вывода
2. Модуль лексического анализатора
3. Модуль синтаксического анализатора
4. Модуль семантического анализатора
5. Модуль кодогенирации

# Требования к подсистеме «Ввода-вывода»

Требование REQ\_IO\_001

Функция ввода кода должна быть доступна пользователю при помощи указания пути к текстовому файлу с кодом или перемещении этого файла в папку продукта.

Требование REQ\_IO\_002

Функция вывода кода должна производиться автоматически по завершению работы продукта в ту же папку, где находится файл с исходным кодом.

Требование REQ\_IO\_003

При возникновении ошибки сообщение об этом должно выводиться в консоль, а работа продукта возвращаться к стадии указания пути к исходному коду.

# Требования к подсистеме «Модуль лексического анализатора»

Требование REQ\_LA\_001

Результаты работы данного модуля должны отображаться в консоль перед началом работы следующего модуля.

Требование REQ\_LA\_002

У пользователя должна быть возможность сохранить результаты работы данного модуля в виде текстового файла в папку, где находится файл исходного кода

Требование REQ\_IO\_003

При возникновении ошибки сообщение об этом должно выводиться в консоль с указанием позиции ошибки в тексте кода. а работа продукта возвращаться к стадии указания пути к исходному коду.

# Требования к подсистеме «Модуль синтаксического анализатора»

Требование REQ\_SINTA\_001

Результаты работы данного модуля должны отображаться в консоль перед началом работы следующего модуля.

Требование REQ\_SINTA\_002

У пользователя должна быть возможность сохранить результаты работы данного модуля в виде текстового файла в папку, где находится файл исходного кода

Требование REQ\_IO\_003

При возникновении ошибки сообщение об этом должно выводиться в консоль с указанием позиции ошибки в тексте кода. а работа продукта возвращаться к стадии указания пути к исходному коду.

# Требования к подсистеме «Модуль кодогенирации»

Требование REQ\_CG \_001

Результаты работы данного модуля должны отображаться в консоль перед началом работы следующего модуля.

Требование REQ\_CG\_002

У пользователя должна быть возможность сохранить результаты работы данного модуля в виде текстового файла в папку, где находится файл исходного кода

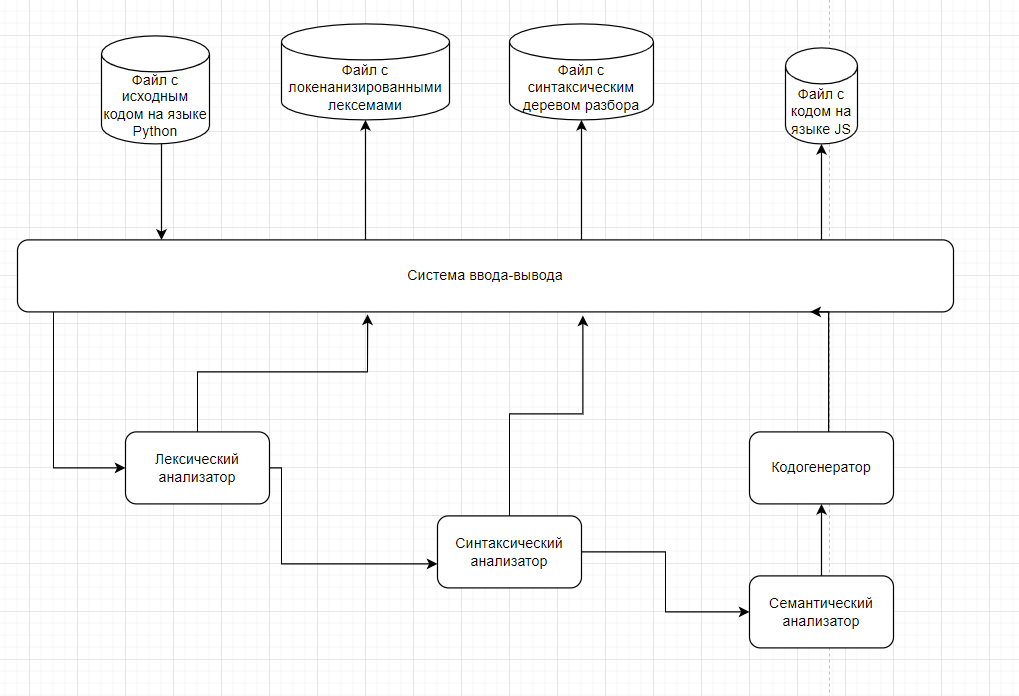
# Архитектура проекта

Программный продукт *PyJsok* предназначен для трансляции кода на языке Python в код на языке JS

Настоящий документ содержит спецификации дизайна, включающие в себя:

1. Архитектурно-контекстную диаграмму системы в целом
2. Архитектурно-контекстную диаграммы подсистем

**1. Архитектурно-контекстная диаграмма системы**

****

Программный продукт *название* состоит из следующих подсистем:

1. Система ввода-вывода
2. Модуль лексического анализатора
3. Модуль синтаксического анализатор
4. Модуль кодогенератора

Система ввода-вывода представляет из себя 1 класс с 1 функцией ввода и 3 функциями вывода. Все 4 функции элементарны и не требуют диаграммы.

**2.1 Диаграмма классов «Модуля лексического анализатора»**

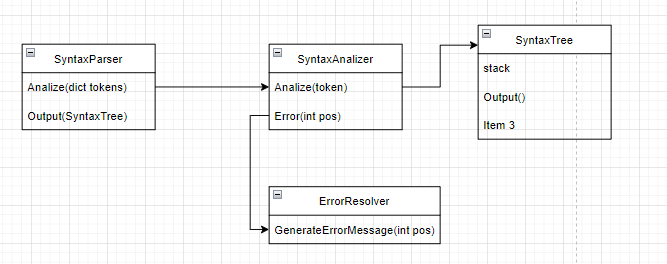
Использовался подход ModelViewController, который облегчит тестирование

LexicalAnalizer – View, Tokenizer – Contreller, Tokens - Model

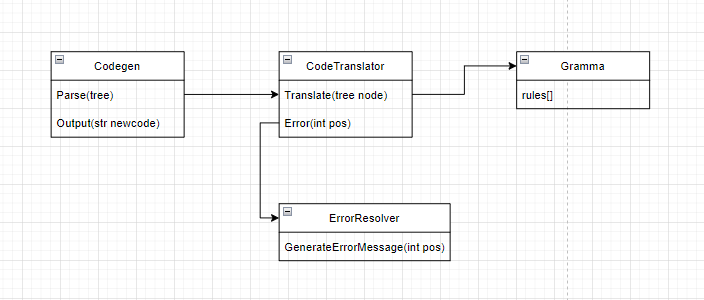
Изображение выглядит как текст, внутренний

Автоматически созданное описание

**2.2 Диаграмма классов «Модуля Синтаксического анализатора»**



**2.4 Диаграмма классов «Модуля кодогенератор»**

****

# Метрики

В качестве основных метрик процесса разработки в нашем проекте были выбраны:

**Productivity**

Productivity = LOC / Рабочее время, затраченное на проект.

Стратегическая цель метрики – повысить производительность труда.

Единица измерения – LOC / человеко-час.

**Problem Resolution Rate (PRR)**

PRR = Количество дней на обработку задачи.

Стратегическая цель метрики – сократить сроки выполнения проектов по разработке ПО.

Единица измерения – день.

Основные метрики оценки качества ПО:

**Inspection Faults Rate**

Inspection Faults Rate = (Количество ошибок, обнаруженных при инспекции)/LOC.

Стратегическая цель метрики – повысить качество разрабатываемого ПО и качество инспектирования.

Единица измерения – неполадка / LOC.

# Перечень задач проекта

Нумерация происходит следующим образом. Каждый архитектурный модуль имеет своё кодовое название

IO

LA

SYNA

CG

У каждой задачи в рамках модуля будет своё наименование. Проект сравнительно небольшой, потому каждая задача будет относиться к разработке целого класса

IO-001

Разработать систему ввода-вывода. Система должна соответствовать всем требованиям REQ\_IO.

LA-001

Разработка класса data класса Token. Класс представляет из себя модель токенизированной лексемы. Он хранит тип лексемы и её позицию в исходном коде. Работа данного класса должна соответствовать всем требованиям REQ\_LA

LA-002

Разработать data класс Tokens, который представляет из себя хранилище типов токенов, регулярных выражений каждого типа и итоговый список токенов исходного текста. Работа данного класса должна соответствовать всем требованиям REQ\_LA

LA-003

Разработать класс Tokenizer, который реализует алгоритм токенизации, выводит в консоль ошибку, если встречена лексема неизвестного типа и возвращает список токенов, в случае успешной обработки исходного кода. Работа данного класса должна соответствовать всем требованиям REQ\_LA.

LA-004

Разработать класс LexicalAnalizer, который принимает на вход текст исходного кода и возвращает массив токенов. Данный класс должен выводить в консоль результат своей работы и предлагать сохранять его в текстовом файле. Работа данного класса должна соответствовать всем требованиям REQ\_LA.

SYNA-001

Разработать класс-представление SyntaxTree. Класс реализует дерево синтаксического разбора в соответствии с правилами языка Python. Работа данного класса должна соответствовать всем требованиям REQ\_SINTA.

SYNA-002

Реализовать класс SyntaxAnalizer, реализующий обработку токенов из списка токенов и выводящий в консоль ошибку, если была обнаружена синтаксическая ошибка. Работа данного класса должна соответствовать всем требованиям REQ\_SINTA.

SYNA-003

Разработать класс SyntaxParser, реализующий приём списка токенов и возвращающий объект класса SyntaxTree. Данный класс должен выводить в консоль результат своей работы и предлагать сохранять его в текстовом файле. Работа данного класса должна соответствовать всем требованиям REQ\_SINTA.

CG-001

Разработать класс-грамматику Gramma, содержащий правила грамматики для языка JavaScript. Правила должны быть реализованы в виде функций, принимающих на вход узел дерева синтаксического разбора и возвращающие текст на языке JavaScript. Работа данного класса должна соответствовать всем требованиям REQ\_CG.

CG-002

Разработать класс CodeTranlator, который принимает на вход узел дерева синтаксического разбора, передаёт грамматике и формирует конечный текст. Работа данного класса должна соответствовать всем требованиям REQ\_CG.

CG-003

Разработать класс Codegen, который принимает на вход дерево синтаксического разбора и возвращает конечный текст программы на языке JavaScript. Данный класс должен выводить в консоль результат своей работы и сохранять его в текстовом файле. Работа данного класса должна соответствовать всем требованиям REQ\_CG.

ERR-001

Разработать и реализовать класс обработки ошибок, возникающий при работе всех вышеперечисленных классов. Дополнять функционал класса по мере разработки и тестирования продукта.

Дополнение задач:

CG-001

ISSUE ID 01

Грамматику необходимо дополнить правилами описания условных операторов

SYNA-002

ISSUE ID 02

Настроить модуль на работу с обновлённой грамматикой.

CG-002

ISSUE ID 03

Добавить функционал для работы с условными операторами. Реализовать обработку узла синтаксического дерева, обозначающего условный оператор.

CG-003

ISSUE ID 04

Добавить функционал для работы с условным оператором.

Данный перечень задач реализован в среде командной разработке Trello

# Рекомендации по кодированию

Все описанные ниже правила разделены на три категории: запреты, требования, рекомендации.

**Запреты** описывают положения, которые нельзя нарушать ни при каких условиях.

**Требования** описывают то, что должно выполняться всегда и в любом проекте. В каждом конкретном проекте лидер проекта может принять решение, что определённые требования могут не соблюдаться. При этом должны быть проделаны следующие шаги: следует доказать необходимость исключений, следует описать, где и по какой причине были сделаны исключения.

**Рекомендации** описывают то, что может существенно облегчить жизнь разработчикам. Пренебрежение рекомендациями не приведёт к провалу проекта, но может осложнить его сопровождение или взаимодействие между разработчиками.

1. Запреты

* Рекомендуется использовать 4 пробела на каждый уровень отступа. Python 3 запрещает смешивание табуляции и пробелов в отступах. Код, в котором используются и те, и другие типы отступов, должен быть исправлен так, чтобы отступы в нем были расставлены только с помощью пробелов.

Хорошо

def•no\_tab\_using():

••••no\_tab•=•'Using•4•spaces'

Плохо

def•use\_tab():

→| one\_tab\_using•=•'Ugly'

* Когда вы генерируете исключение, пишите raise ValueError('message') вместо старого синтаксиса raise ValueError, message.

Старая форма записи запрещена в python 3.

Такое использование предпочтительнее, потому что из-за скобок не нужно использовать символы для продолжения перенесенных строк, если эти строки длинные или если используется форматирование.

* Двойные подчеркивания (в начале и конце имен) зарезервированы для языка.

Хорошо

my\_variable = 'Variable'

Плохо

\_\_myvariable\_\_ = 'Variable'

1. Требования

* Ограничьте длину строки максимум 79 символами.

Стандартная библиотека Python консервативна и требует ограничения длины строки в 79 символов (а строк документации/комментариев в 72).

Предпочтительный способ переноса длинных строк является использование подразумеваемых продолжений строк Python внутри круглых, квадратных и фигурных скобок. Длинные строки могут быть разбиты на несколько строк, обернутые в скобки. Это предпочтительнее использования обратной косой черты для продолжения строки.

* Комментарии, противоречащие коду, хуже, чем отсутствие комментариев. Всегда исправляйте комментарии, если меняете код!

Комментарии должны являться законченными предложениями. Если комментарий — фраза или предложение, первое слово должно быть написано с большой буквы, если только это не имя переменной, которая начинается с маленькой буквы (никогда не изменяйте регистр переменной!).

Если комментарий короткий, можно опустить точку в конце предложения. Блок комментариев обычно состоит из одного или более абзацев, составленных из полноценных предложений, поэтому каждое предложение должно оканчиваться точкой.

Ставьте два пробела после точки в конце предложения.

Программисты, которые не говорят на английском языке, пожалуйста, пишите комментарии на английском, если только вы не уверены на 120%, что ваш код никогда не будут читать люди, не знающие вашего родного языка.

1. Рекомендации

* Не разделяйте ваши строки с помощью точек с запятой и не используйте точки с запятой для разделения команд, находящихся на одной строке.

Хорошо

a = 'String'

b = 15

c = 7.2

Плохо

a = 'String';

b = 15; c = 7.2;

* Не ставьте пробелы внутри каких-либо скобок (обычных, фигурных и квадратных).

Хорошо

pineapple(pine[1], {apple: 2})

Плохо

pineapple( pine[ 1 ], { apple: 2 } )

* Если есть необходимость пройти в цикле по ряду чисел, то метод range будет намного приемлемее, как минимум потому, что этот метод потребляет намного меньше памяти, чем вариант в блоке "Плохо". А представьте, что у вас ряд из трёх миллиардов последовательных чисел!

Хорошо

for i in range(6):

print(i\*\*2)

Плохо

for i in [0, 1, 2, 3, 4, 5]:

print(i\*\*2)

**Чек-лист для проверки корректности кода**

**Appearance**

1. Обновляется история изменений (версия, данные, автор и комментарии к изменениям) и информация об авторских правах.
2. Все комментарии написаны на английском языке.
3. Размещение скобок, отступы, интервалы, соглашение об именовании и длина строки соответствуют стандарту кодирования.

**Miscellaneous**

1. Каждая строка кода содержит не более одного оператора.

**Constants**

1. Все константы записаны заглавными буквами, а слова разделены символами подчёркивания.

# План тестирования

**Тесты для подсистемы «Ввода-вывода»**

Тест TEST\_IO\_001

Тестируемые требования: REQ\_IO\_001

Вход: путь к файлу с кодом на языке python

Выход: сообщение об удачной загрузки исходного кода

Описание: тестирование функции ввода кода

Резюме: Тест пройден

Тест TEST\_IO\_002

Тестируемые требования: REQ\_IO\_002

Вход: код на языке java script

Выход: сообщение об удачном сохранении полученного кода

Описание: тестирование функции сохранения кода

Резюме: Тест пройден

Тест TEST\_IO\_003

Тестируемые требования: REQ\_IO\_003

Вход: путь к файлу, с ошибкой

Выход: сообщение с описанием ошибки

Описание: тестирование функции вывода ошибок процессов ввода вывода

Резюме: Тест пройден

**Тесты для подсистемы «Лексического анализатора»**

Тест TEST\_LA\_001

Тестируемые требования: REQ\_LA\_001

Вход: результаты работы модуля

Выход: сообщение о результатах работы модуля

Описание: тестирование функции вывода результатов работы модуля

Резюме: Тест пройден

Тест TEST\_LA\_002

Тестируемые требования: REQ\_LA\_002

Вход: результаты работы модуля

Выход: сообщение об удачном сохранении результатов

Описание: тестирование функции сохранения результатов работы модуля

Резюме: Тест пройден

Тест TEST\_LA\_003

Тестируемые требования: REQ\_LA\_003

Вход: файл с ошибкой

Выход: сообщение об ошибке и месте локализации

Описание: тестирование функции вывода ошибок в ходе работы модуля

Результат: Тест пройден

**Тесты для подсистемы «Синтаксического анализатора»**

Тест TEST\_SINTA\_001

Тестируемые требования: REQ\_ SINTA \_001

Вход: результаты работы модуля

Выход: сообщение о результатах работы модуля

Описание: тестирование функции вывода результатов работы модуля

Резюме: Тест пройден

Тест TEST\_ SINTA \_002

Тестируемые требования: REQ\_ SINTA \_002

Вход: результаты работы модуля

Выход: сообщение об удачном сохранении результатов

Описание: тестирование функции сохранения результатов работы модуля

Резюме: Тест пройден

Тест TEST\_ SINTA \_003

Тестируемые требования: REQ\_ SINTA \_003

Вход: файл с ошибкой

Выход: сообщение об ошибке и месте локализации

Описание: тестирование функции вывода ошибок в ходе работы модуля

Резюме: Тест пройден

**Тесты для подсистемы «Кодогенерации»**

Тест TEST\_CD\_001

Тестируемые требования: REQ\_CD\_001

Вход: результаты работы модуля

Выход: сообщение о результатах работы модуля

Описание: тестирование функции вывода результатов работы модуля

Резюме: Тест не пройден

Тест TEST\_CD\_002

Тестируемые требования: REQ\_CD\_002

Вход: результаты работы модуля

Выход: сообщение об удачном сохранении результатов

Описание: тестирование функции сохранения результатов работы модуля

Резюме: Тест пройден

# Матрица тестирования

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Test ID** |  |  |  |  |  |  |  |  |  |  |
| **Req ID** | TEST\_IO\_001 | TEST\_IO\_002 | TEST\_IO\_003 | TEST\_LA\_001 | TEST\_LA\_002 | TEST\_LA\_003 | TEST\_SINTA\_001 | TEST\_ SINTA \_002 | TEST\_ SINTA \_003 | TEST\_CD\_001 | TEST\_CD\_002 |
| REQ\_IO\_001 |  |  |  |  |  |  |  |  |  |  |  |
| REQ\_IO\_002 |  |  |  |  |  |  |  |  |  |  |  |
| REQ\_IO\_003 |  |  |  |  |  |  |  |  |  |  |  |
| REQ\_LA\_001 |  |  |  |  |  |  |  |  |  |  |  |
| REQ\_LA\_002 |  |  |  |  |  |  |  |  |  |  |  |
| REQ\_LA\_003 |  |  |  |  |  |  |  |  |  |  |  |
| REQ\_SINTA\_001 |  |  |  |  |  |  |  |  |  |  |  |
| REQ\_SINTA\_002 |  |  |  |  |  |  |  |  |  |  |  |
| REQ\_SINTA\_003 |  |  |  |  |  |  |  |  |  |  |  |
| REQ\_CD\_001 |  |  |  |  |  |  |  |  |  |  |  |
| REQ\_CD\_002 |  |  |  |  |  |  |  |  |  |  |  |

# Заключение

Подсчёт метрик проекта:

**Productivity**

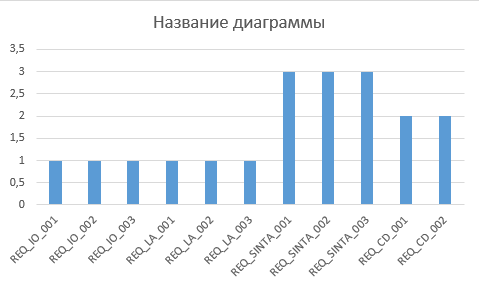
Рабочее время, затраченное на проект = 62 чел. часов

LOC = 994

Productivity = 16

**PRR**

Количество дней на решение задачи



**Inspection Faults Rate**

Количество ошибок, обнаруженных при инспекциях = 37

LOC = 994

IFR = 0.04

Это примерно одна ошибка на тридцать строк кода

Вывод.

Глядя на значения метрик, можно сделать вывод, что непосредственная работа над проектом шла в очень малые сроки. Об этом говорит PRR. Можно предположить, что именно это стало причиной повышения Productivity. Тем не менее инспекции выявили умеренное количество ошибок, что говорит о способности команды справиться с поставленными задачами в сжатые строки. Также стоит отметить, что количество дней на некоторые задачи значительно больше, чем у других. Это указывает на неправильное распределение задач или на их недостаточную деструктуризацию.