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, ПЕРЕВОДЯЩЕГО ПОДМНОЖЕСТВО ЯЗЫКА PYTHON В ЭКВИВАЛЕНТНОЕ ПОДМНОЖЕСТВО ЯЗЫКА LUA

**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. Б9118-09.03.04прогин

\_\_\_\_\_\_\_\_\_\_\_ Аксененко О.Ю.

\_\_\_\_\_\_\_\_\_\_\_ Говоров В.Р.

\_\_\_\_\_\_\_\_\_\_\_ Жлуткин Р.В.

\_\_\_\_\_\_\_\_\_\_\_ Рева С.Б.

\_\_\_\_\_\_\_\_\_\_\_ Шешин М.С.

Руководитель: д. т. н., доцент Гриняк В. М.

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

(подпись)

Защищена с оценкой \_\_\_\_\_\_\_\_

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

(подпись) И.О. Фамилия

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2022 г.

г. Владивосток

2022 г.

Оглавление

[Введение 3](#_Toc90460831)

[1 Software Project Plan (календарный план работы над проектом) 4](#_Toc90460832)

[2 Inspection Requlations (регламент проведения инспекции) 4](#_Toc90460833)

[3 Tasks States Model (модель состояний задач) 6](#_Toc90460834)

[4 Marketing Requirements Specification (маркетинговые требования к проекту) 8](#_Toc90460835)

[5 Software Requirements Specification (технические требования к продукту) 12](#_Toc90460836)

[6 Software Design Specification (архитектурные требования к проекту) 14](#_Toc90460837)

[7 Metrics Regulations (программа измерений проекта) 20](#_Toc90460838)

[8 Code Issue List (перечень задач проекта) 21](#_Toc90460839)

[9 Coding Recommendations (рекомендации по кодированию) 22](#_Toc90460840)

[10 Software Test Plan (план тестирования проекта) 24](#_Toc90460841)

[11 Requirements Traceability Matrix (матрица покрытия тестами требований) 27](#_Toc90460842)

[12 Software Test Logs (результаты выполнения тестов) 27](#_Toc90460843)

[Заключение 41](#_Toc90460844)

# Введение

Курс «Технология коллективной промышленной разработки информационных систем» посвящён знаниям и умениям, необходимым для работы в программистских коллективах: как небольших (2-5 человек), сидящих в одном офисе, так и больших (сотни разработчиков), территориально распределённых по всему миру. Полученные в рамках курса знания необходимы для позиции Инженер-программист и Инженер по тестированию ПО. Последняя, как правило, подразумевает коллективную работу в тесном контакте с собственно разработчиками и клиентами.

Цель курсовой работы: получение знаний и умений в коллективной разработке программного средства.

Разрабатываемым программным средством является транслятор, который преобразует программу, содержащую подмножество языка Python, в программу, содержащую подмножество языка Lua, генерируя эквивалентный исходный код.

При этом воспроизводятся следующие этапы разработки ПО:

* Формирование команды. В команде выделается руководитель проекта (team leader), кодировщики и технический писатель.
* Формулировка идеи проекта и разработка требований.
* Разработка дизайна проекта (конструирование). Описывается архитектура проекта. Проект детально разбивается на подзадачи (issues), составляется график работ.
* Реализация проекта с использованием систем контроля версий и систем трекинга.
* Составление плана тестирования, разработка тестов и их реализация.

# 1 Software Project Plan (календарный план работы над проектом)

*Название проекта:* LuPy

*Команда проекта:*

*Руководитель проекта:* Говоров Владислав

*Кодировщик 1:* Аксененко Олег

*Кодировщик 2:* Жлуткин Рома

*Кодировщик 3:* Шешин Максим

*Технический писатель:* Рева Светлана

*Сроки выполнения проекта (deadlines):*

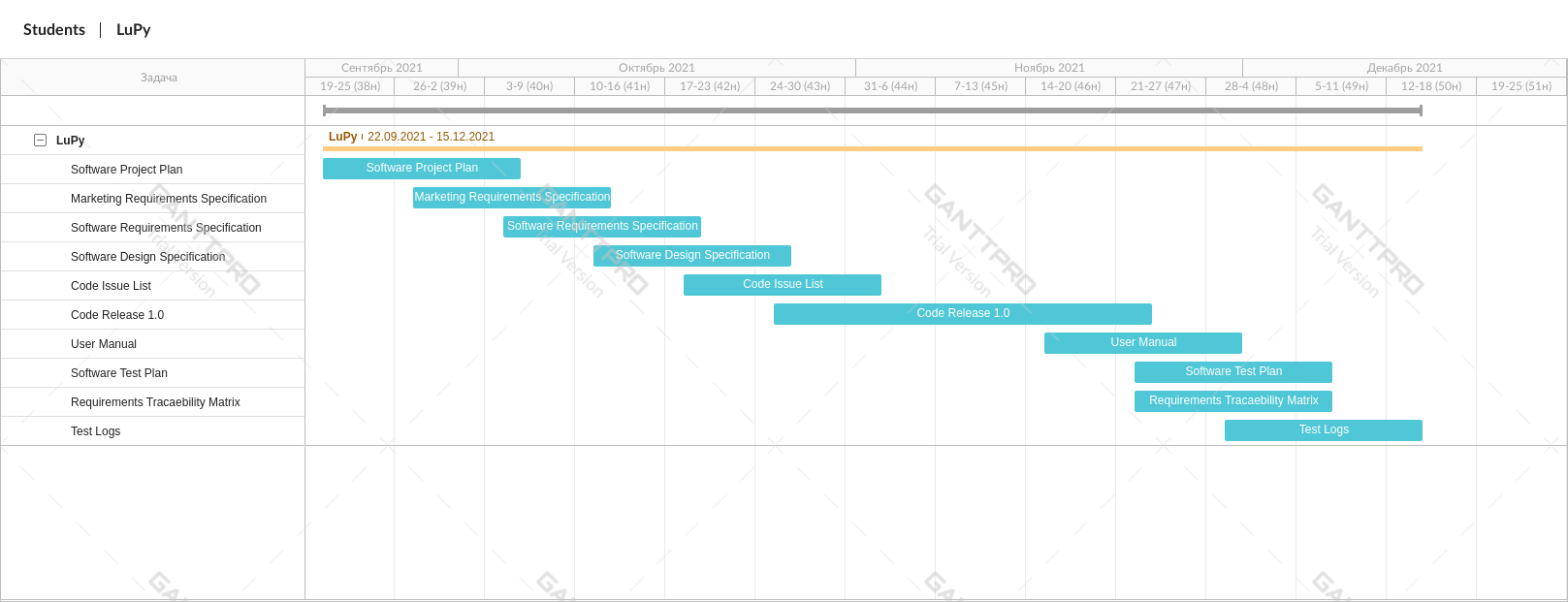


Рисунок 1 – Сроки выполнения проекта (deadlines)

# 2 Inspection Requlations (регламент проведения инспекции)

1. *Критерии отнесения к формальной/неформальной инспекции для различных типов рабочих продуктов*

Инспекция является формальной в случае слияния веток. В случае внесения изменений в рабочую ветку инспекция является неформальной.

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

Роли: автор, модератор (принимает изменения, назначает степень важности замечаний, закрывает замечания), инспектор (комментирование изменений, создание замечаний).

Число участников: 5 (вся команда участвует в инспекциях).

1. *Этапы инспекции*
2. Назначение инспекции;
3. Обзорное (overview) собрание;
4. Исследование изменений;
5. Собрание по инспекции (Summary);
6. Завершение инспекции (распространение результатов, переработка рабочего продукта, проверка исправления недостатков в рабочем продукте).
7. *Порядок организации (кто куда что выгружает, кому что рассылает, кого приглашает и т.д.)*

Изменения выгружаются в отдельную ветку с соответствующим задаче названию. Сообщение о внесении изменений отправляется в командный чат, приглашая всех участников к проведению инспекции.

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

Проведение юнит и интеграционных тестов. В случае слияния производится предварительная проверка совместимости веток.

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

Автор – внесение комментариев и пояснений.

Модератор – назначение степени важности замечаний, закрытие замечаний, принятие изменений, выполнение обязанностей инспектора.

Инспектор – комментирование изменений, создание замечаний, закрытие собственных замечаний.

1. *Перечень статусов и степени важности замечаний*

Статус:

* дефект (defect);
* ошибка (error);
* комментарий (comment).

Степень важности:

* критическая (critical);
* особо важная (major);
* средняя (moderate);
* мелкая, незначительная (minor);
* другие (other).

1. *Порядок верификации учёта замечаний*

Верификацию производит модератор вручную.

1. *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>.
* Inspection Preparation Rate (IPR) IPR = (Количество инспекторов \* Размер продукта)/Общее время подготовки; Стратегическая цель метрики – повысить качество разрабатываемого ПО. Изучаемый объект метрики – подготовка к инспекции, измеряемый атрибут – производительность подготовки к инспекции. Единица измерения – <страница, требование, LOC, тест>/ час.
* Inspection Rate (IR) IR = Размер продукта/Общее время инспектирования; Стратегическая цель метрики – повысить качество разрабатываемого ПО.

# 3 Tasks States Model (модель состояний задач)

*Перечень возможных состояний задачи и их интерпретация:*

* New – новая задача.
* Analysis – в процессе анализа. В это состояние задачу переводит сотрудник после того, как был начат анализ.
* Terminated – прерванная задача. В это состояние задача переводится после анализа в случае принятия решения о необходимости отмены.
* Coding – кодирование. В это состояние задача переводится после начала этапа кодирования программы.
* Tested – протестировано. В это состояние задача переводится сотрудником, осуществляющим тестирование изменений рабочего продукта.
* Inspected – проинспектировано. В это состояние задача переводится инспектором после инспектирования изменений рабочего продукта.
* Closed – закрыто. В это состояние задача переводится при успешном ее выполнении в ходе прохождения всех необходимых состояний.

*Правила перехода задачи из состояния в состояние:*

New → Analysis – переход осуществляется после создания формального описания требований задачи.

Analysis → Terminated – переход осуществляется при выявленном в ходе анализа несоответствии составленного формального описания задачи бизнес-целям проекта, либо при неосуществимости данного функционала.

Analysis → Coding – переход осуществляется после анализа соответствия составленного формального описания задачи бизнес-целям проекта и осуществимости данного функционала.

Coding → Tested – переход осуществляется после завершения этапа кодирования и передачи составленного кода команде тестировщиков.

Tested → Inspected – переход осуществляется после проверки кода инспектором и передачи составленного кода на инспекцию квалифицированному сотруднику.

Inspected → Coding – переход осуществляется после решения о необходимости внесения изменений в код после проведения инспекции.

Inspected → Closed – переход осуществляется после успешного прохождения инспекции.

# 4 Marketing Requirements Specification (маркетинговые требования к проекту)



Рисунок 2 – Титульный слайд

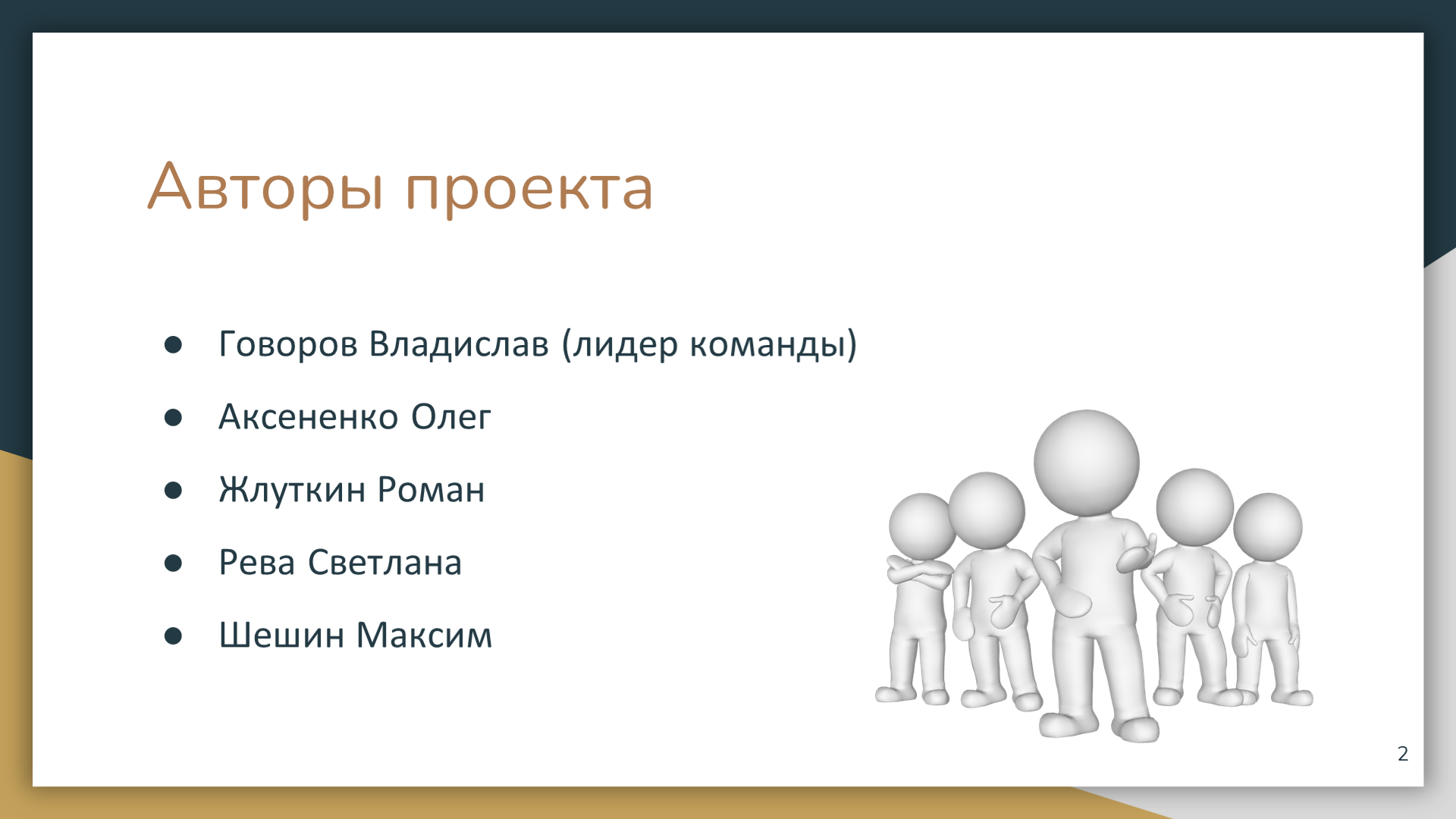


Рисунок 3 – Авторы проекта

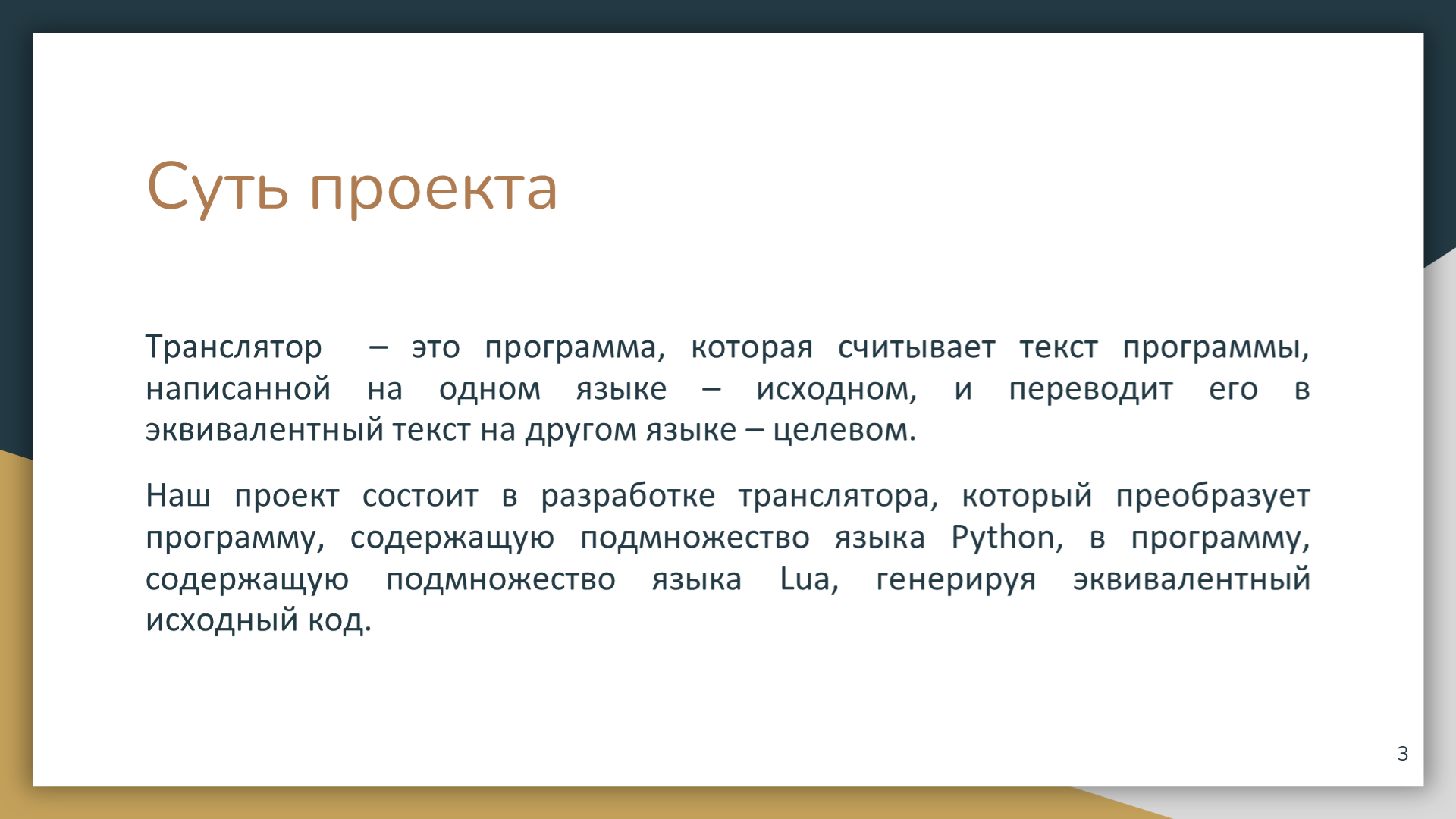


Рисунок 4 – Суть проекта

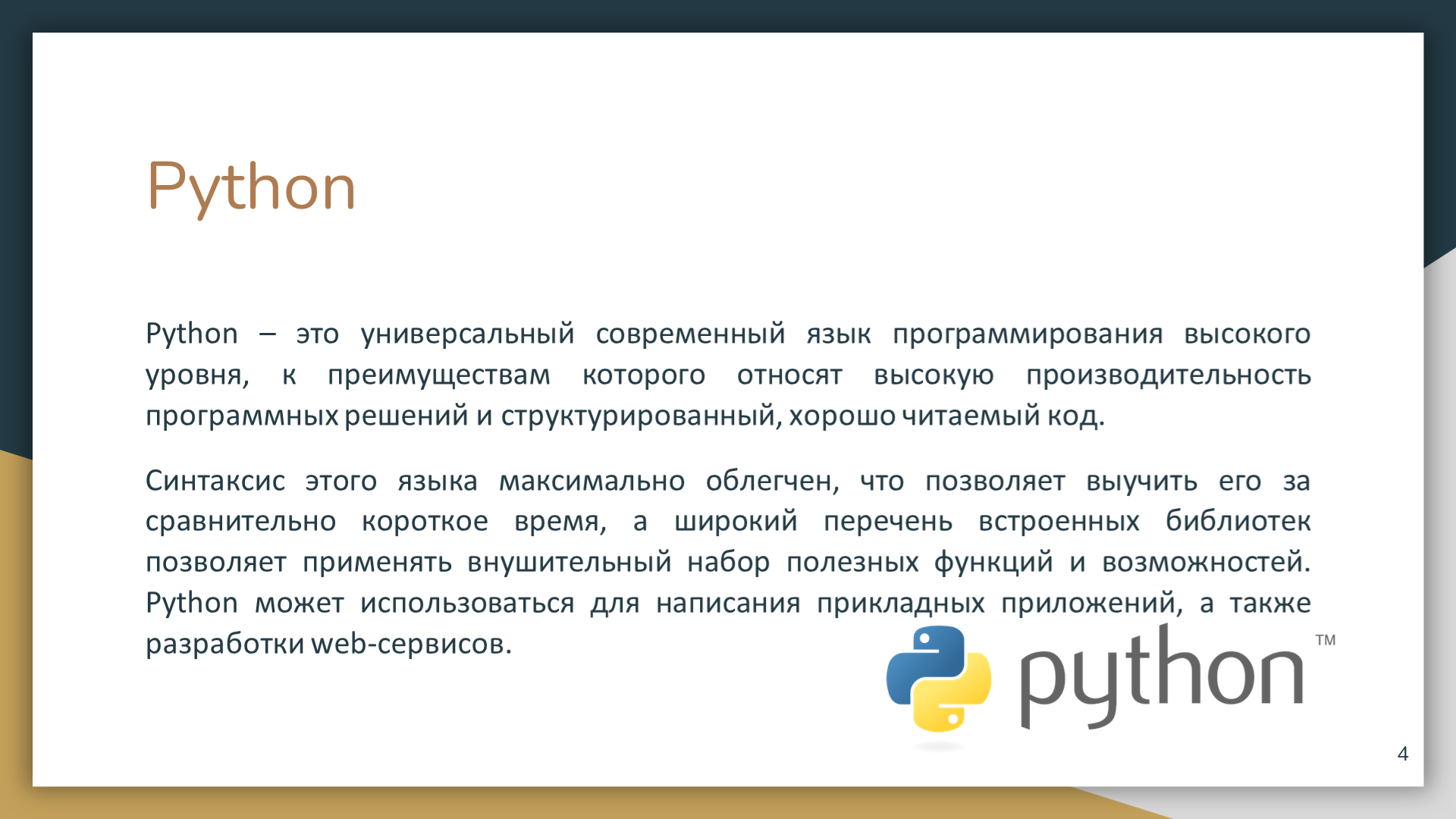


Рисунок 5 – Python

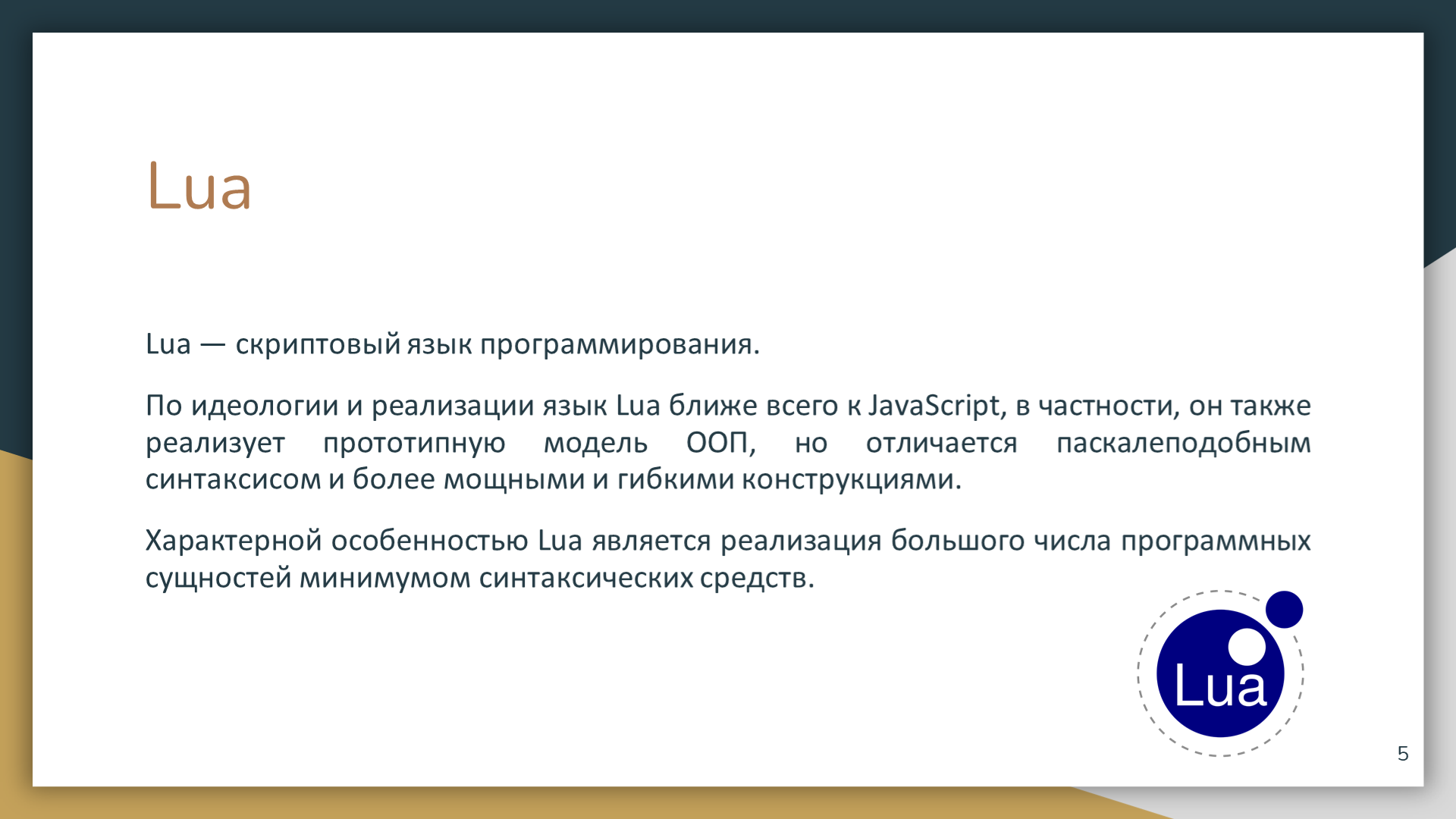


Рисунок 6 – Lua

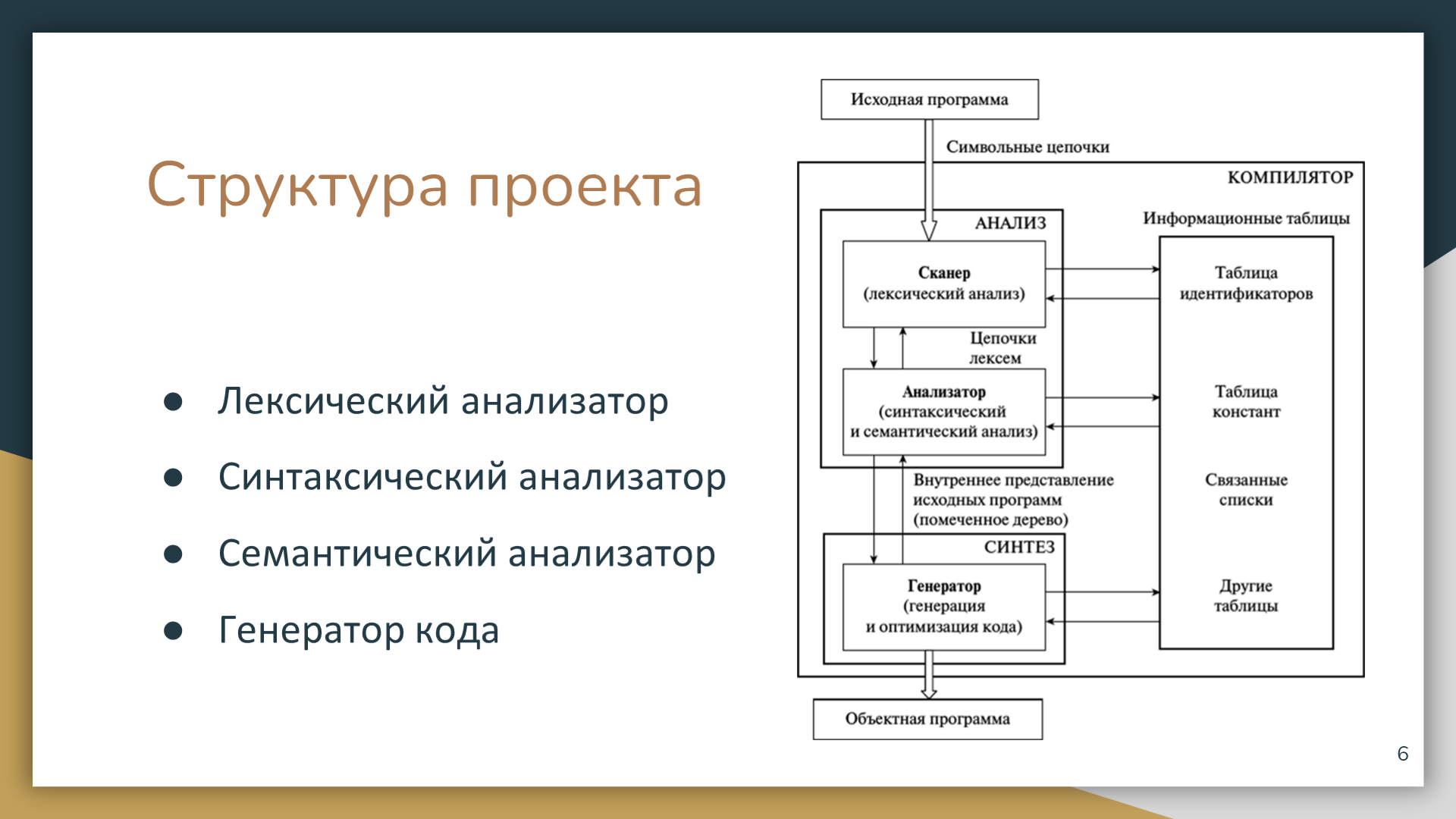


Рисунок 7 – Структура проекта

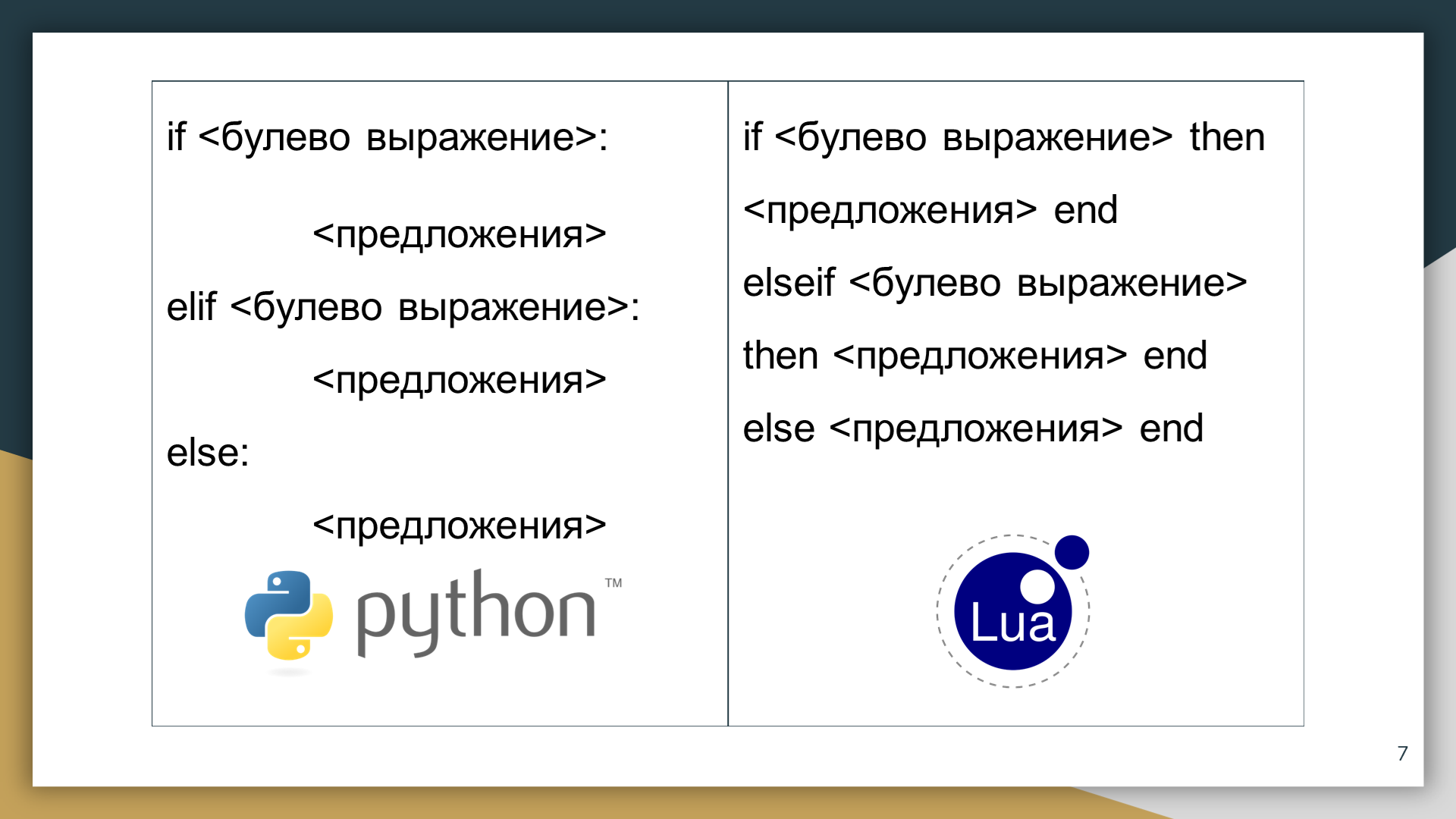


Рисунок 8 – Фрагмент таблицы соответствия Python и Lua

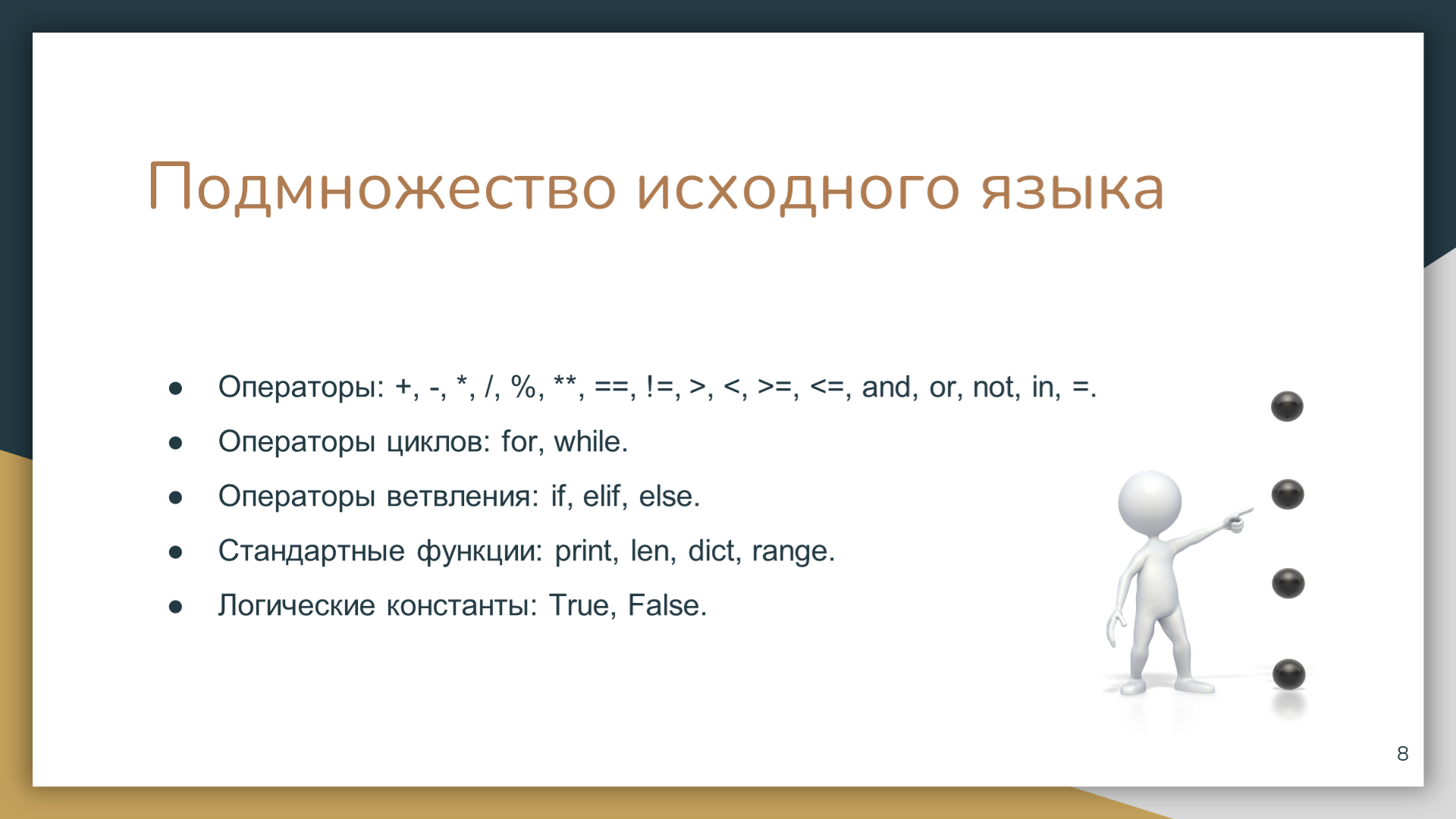


Рисунок 9 – Подмножество исходного языка

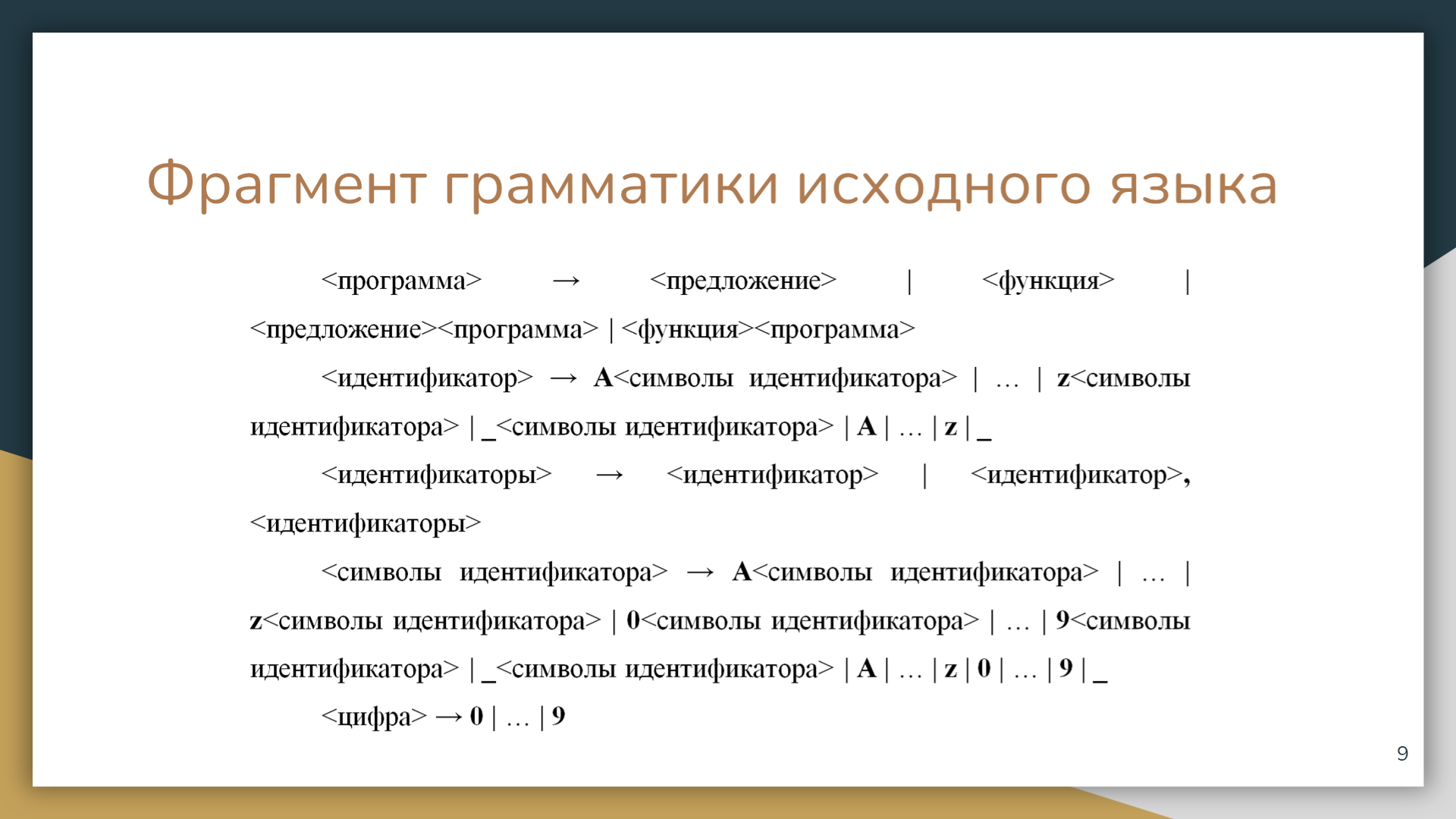


Рисунок 10 – Фрагмент грамматики исходного языка

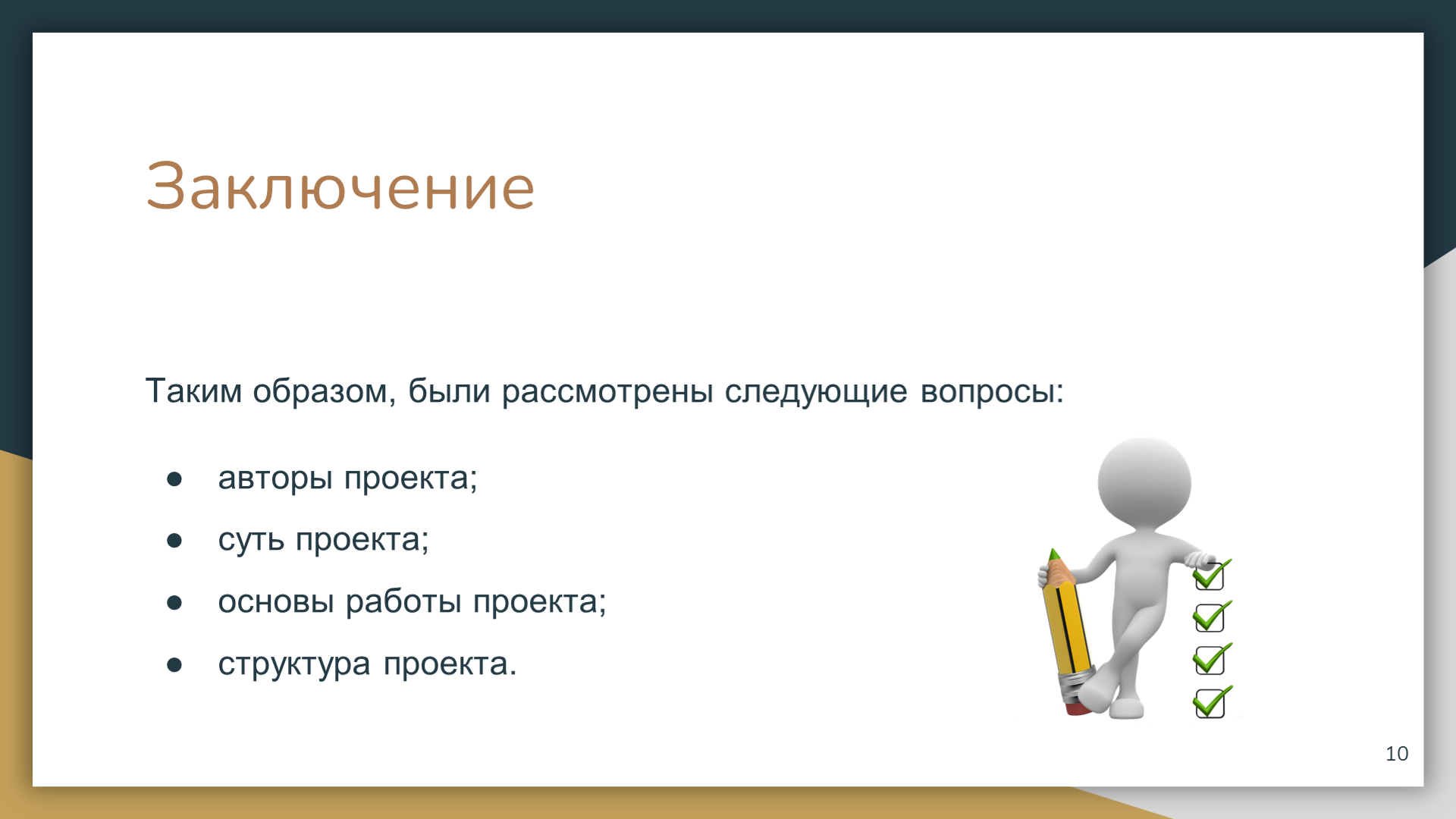


Рисунок 11 – Заключение

# 5 Software Requirements Specification (технические требования к продукту)

Программный продукт *LuPy* предназначен для осуществления процесса трансляции, в ходе которого происходит преобразование программы, содержащей подмножество языка Python, в программу, содержащую подмножество языка Lua, а также генерируется эквивалентный исходный код.

Программный продукт *LuPy* 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*

Функция загрузки текста программы на исходном языке Python должна быть доступна для пользователя ввод в соответствующее текстовое поле.

*Требование REQ\_UI\_002*

Функция трансляции текста программы на исходном языке Python на язык Lua должна быть доступна для пользователя через нажатие кнопки «Трансляция».

**Требования к подсистеме «Лексический анализатор»**

*Требование REQ\_LEX\_001*

Подсистема «Лексический анализатор» должна выполнять преобразование исходной программы в последовательность лексем.

*Требование REQ\_LEX\_002*

Подсистема «Лексический анализатор» должна позволять пользователю получать сообщения о возникших ошибках.

**Требования к подсистеме «Синтаксический анализатор»**

*Требование REQ\_SINT\_001*

Подсистема «Синтаксический анализатор» должна определять структуру программы и представлять ее в виде дерева вывода.

*Требование REQ\_SINT\_002*

Подсистема «Синтаксический анализатор» должна позволять пользователю получать сообщения о возникших ошибках.

**Требования к подсистеме «Семантический анализатор»**

*Требование REQ\_SEM\_001*

Подсистема «Семантический анализатор» должна проверять текст исходной программы на выполнение контекстных условий (о правилах описания идентификаторов; использования идентификаторов в своей области действия; соответствии видов величин, входящих в синтаксические конструкции программ)

*Требование REQ\_SEM\_002*

Подсистема «Семантический анализатор» должна позволять пользователю получать сообщения о возникших ошибках.

**Требования к подсистеме «Генератор кода»**

*Требование REQ\_CG\_001*

Подсистема «Генератор кода» должна конструировать программу на выходном языке из промежуточного представления, полученного в результате работы синтаксического анализа.

# 6 Software Design Specification (архитектурные требования к проекту)

Программный продукт *LuPy* предназначен для осуществления процесса трансляции, в ходе которого происходит преобразование программы, содержащей подмножество языка Python, в программу, содержащую подмножество языка Lua, а также генерируется эквивалентный исходный код.

Настоящий раздел содержит спецификации дизайна, включающие в себя:

1. Архитектурно-контекстную диаграмму системы в целом
2. Диаграммы классов (по подсистемам)

**1. Архитектурно-контекстная диаграмма системы**

Программный продукт LuPy состоит из следующих подсистем:

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

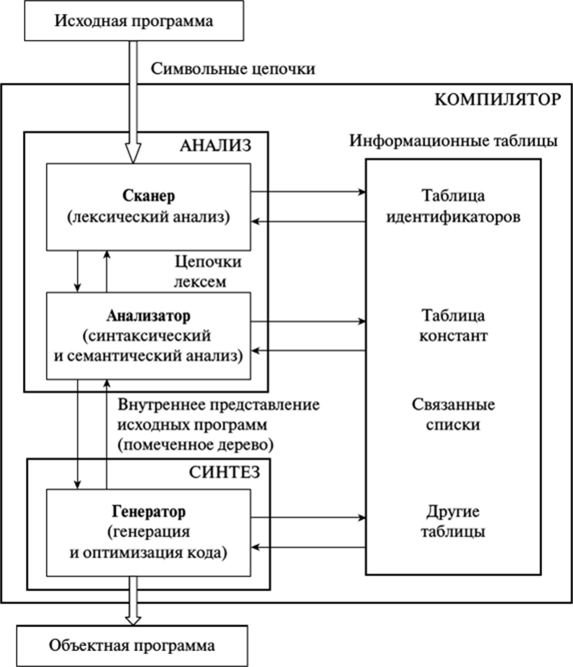


Рисунок 12 – Архитектурно-контекстная диаграмма системы

**2.1. Диаграмма классов подсистемы «Пользовательский интерфейс»**

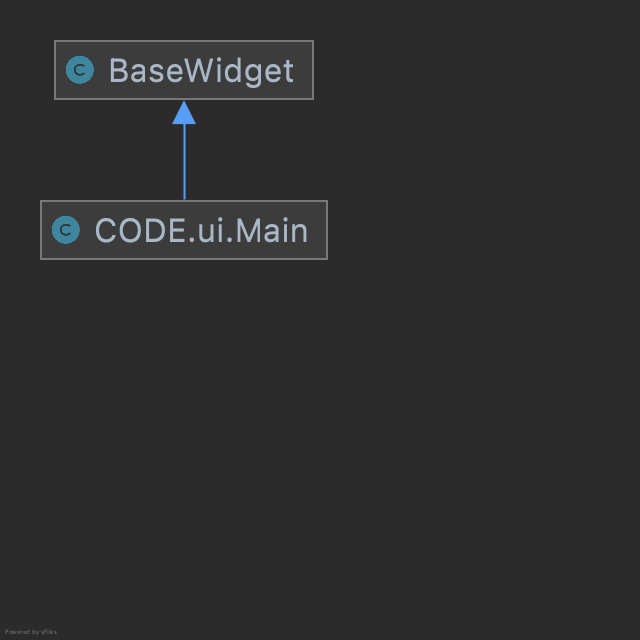


Рисунок 13 – Классы подсистемы «Пользовательский интерфейс»

**3.1. Диаграмма классов подсистемы «Лексический анализатор»**

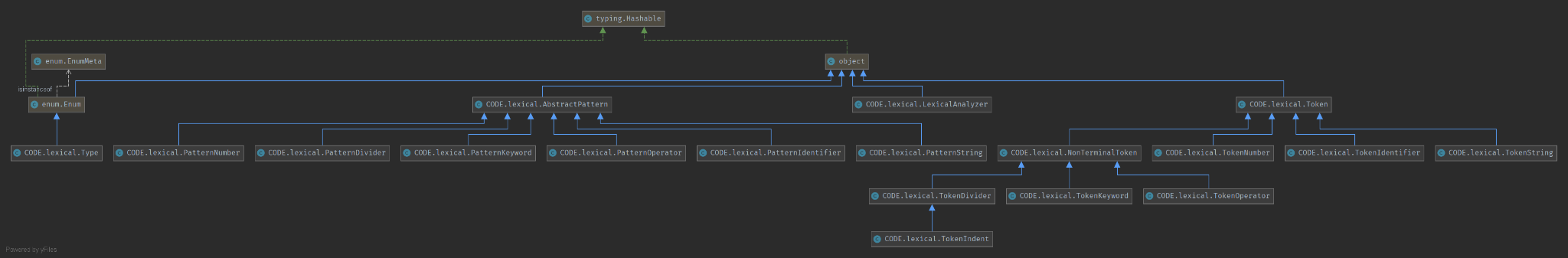
****

Рисунок 14 – Классы подсистемы «Лексический анализатор»

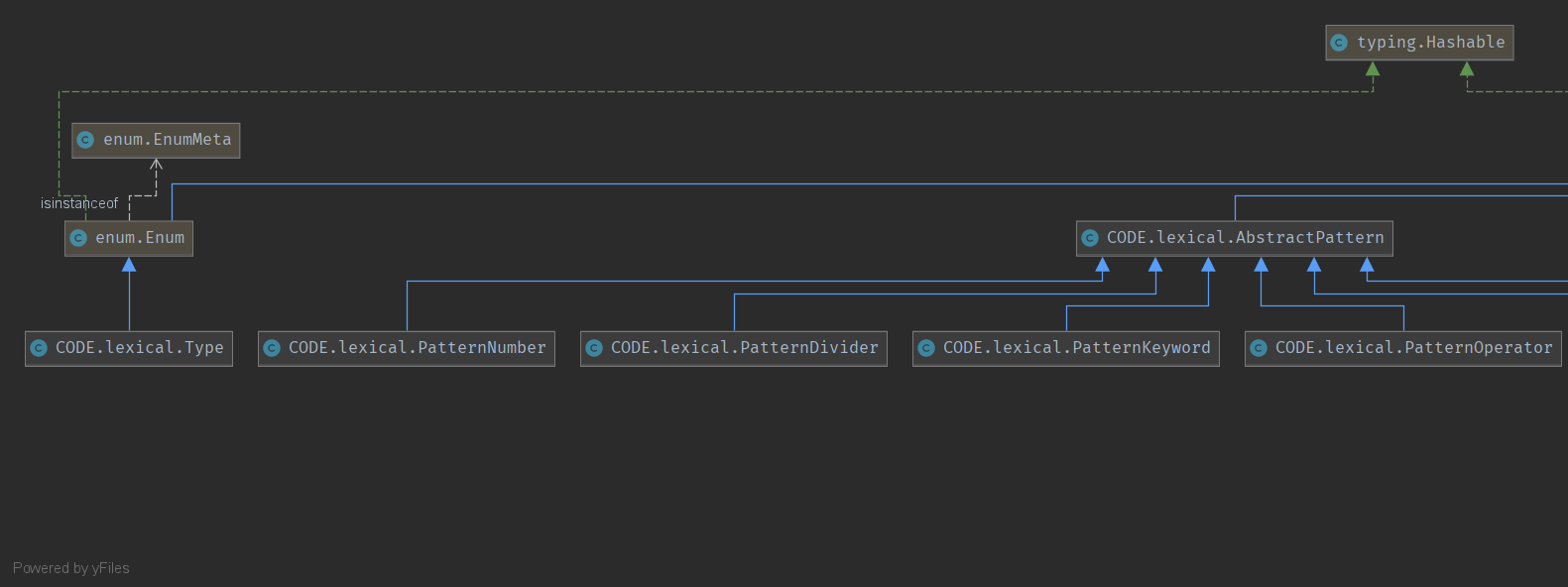


Рисунок 15 – Классы подсистемы «Лексический анализатор» (левый фрагмент)

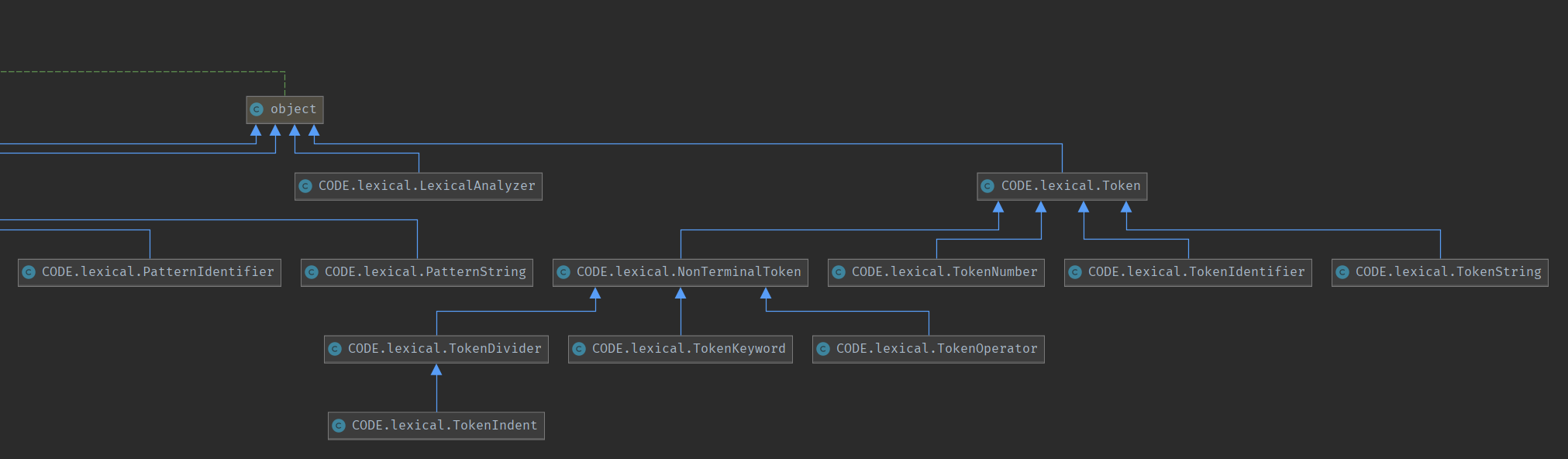


Рисунок 16 – Классы подсистемы «Лексический анализатор» (правый фрагмент)

**4.1. Диаграмма классов подсистемы «Синтаксический анализатор»**

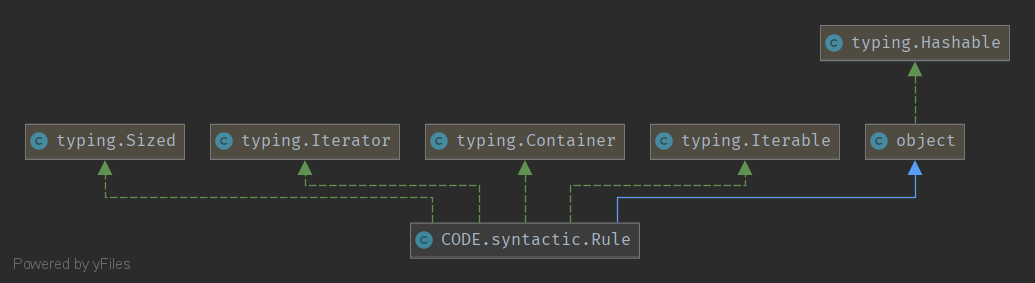


Рисунок 17 – Класс Rule подсистемы «Синтаксический анализатор»

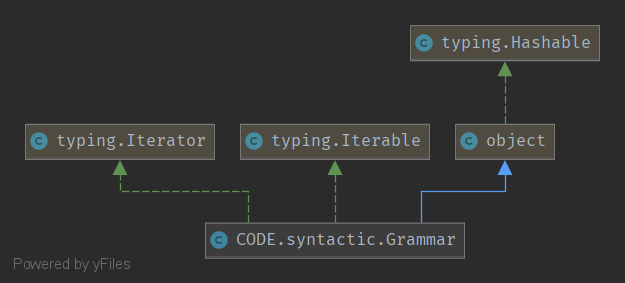


Рисунок 18 – Класс Grammar подсистемы «Синтаксический анализатор»

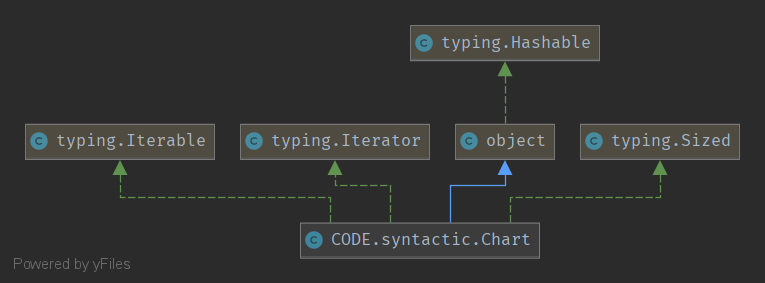


Рисунок 19 – Класс Chart подсистемы «Синтаксический анализатор»

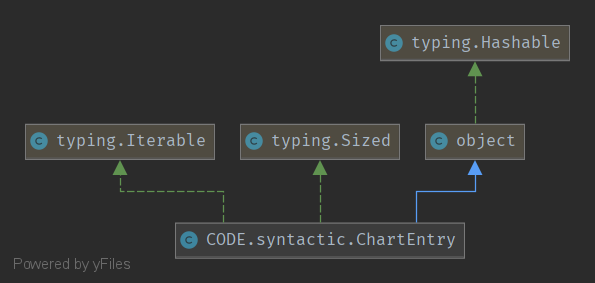


Рисунок 20 – Класс ChartEntry подсистемы «Синтаксический анализатор»

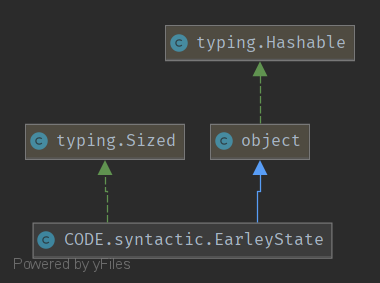


Рисунок 21 – Класс EarleyState подсистемы «Синтаксический анализатор»

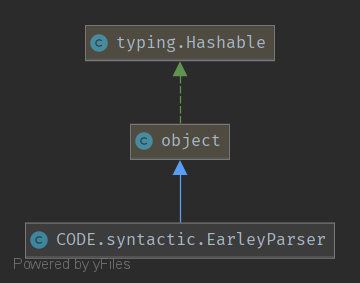


Рисунок 22 – Класс EarleyParser подсистемы «Синтаксический анализатор»

**5.1. Диаграмма классов подсистемы «Семантический анализатор»**

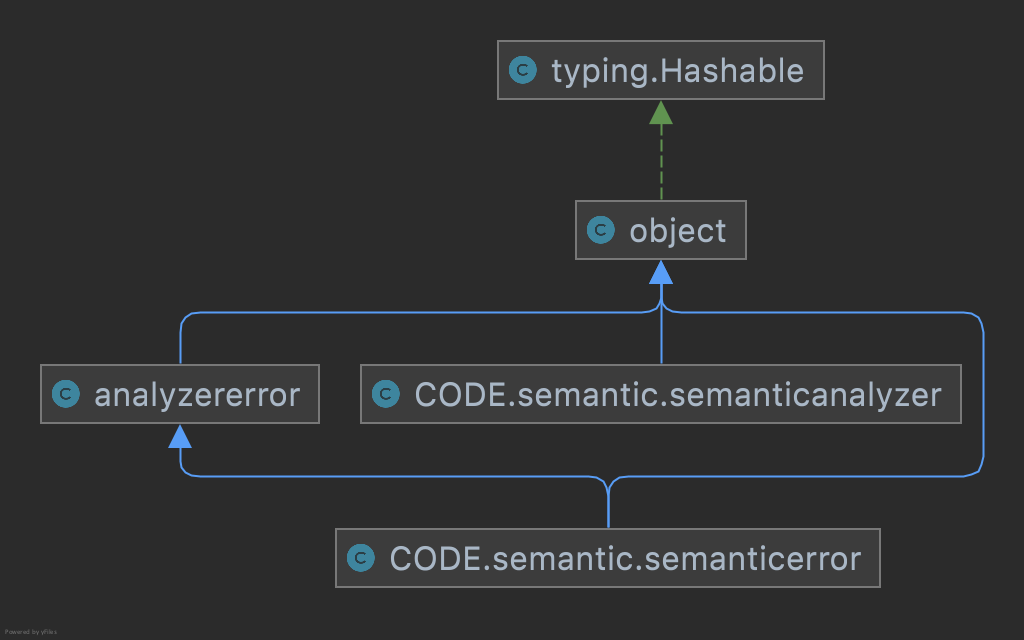


Рисунок 23 – Классы подсистемы «Семантический анализатор»

**6.1. Диаграмма классов подсистемы «Генератор кода»**

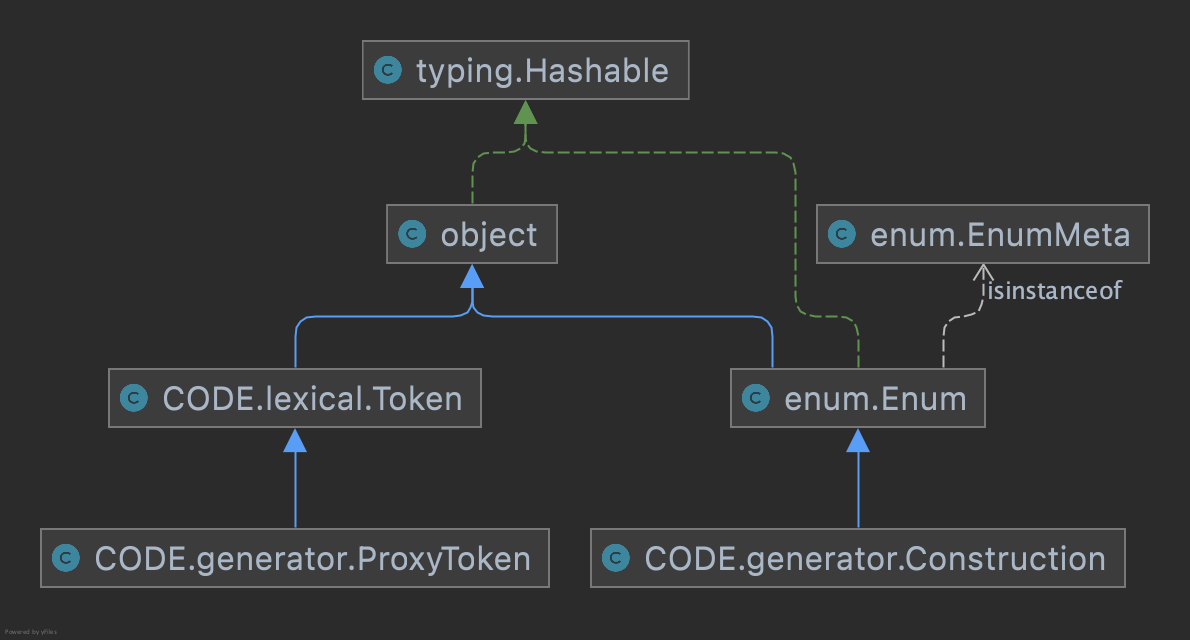
****

Рисунок 24 – Классы подсистемы «Генератор кода»

# 7 Metrics Regulations (программа измерений проекта)

Программа измерений проекта – комплекс мероприятий, направленных на количественную оценку эффективности работы над проектом и позволяющих обоснованно принимать решения в соответствии с полученной оценкой.

Поскольку в рамках проекта ведется разработка программы, основным объектом для оценки является ее код. За единицу измерения размера кода будем принимать число строк кода – Line Of Code (LOC). При подсчете строк кода не учитываются пустые строки, а также комментарии.

1. *Система метрик для оценки эффективности процесса разработки*

* Метрика «On project % time» (OPPT)

Процент времени, затрачиваемый на работу по проекту.

Рассчитывается как OPPT = (Рабочее время, затраченное на задачу / Общее время, выделенное на задачу по плану) \* 100%.

* Метрика «Productivity»

Производительность труда на проекте в LOC / человеко-час.

Рассчитывается как Productivity = LOC / Рабочее время, затраченное на проект.

* Метрика «Problem Resolution Rate» (PRR)

Количество дней, затрачиваемое на обработку (решение) задачи.

Рассчитывается как PRR = Количество дней на обработку задачи.

Все вышеперечисленные метрики позволяют оценить сроки выполнения проекта и указать на необходимость принятия мер по их сокращению.

1. *Система метрик для оценки качества программного продукта, создаваемого в проекте*

* Метрика «In Process Faults» (IPF)

Плотность неполадок, возникших при разработке продукта в неполадках / LOC.

Рассчитывается как IPF = Число обнаруженных ошибок до выпуска его релиза при тестировании / LOC.

Метрика позволяет повысить качество разрабатываемого в рамках проекта продукта.

Средний объем модуля составляет 200 строк кода. Нормативное значение метрики ~ 3-4/200 неполадок / LOC.

# 8 Code Issue List (перечень задач проекта)

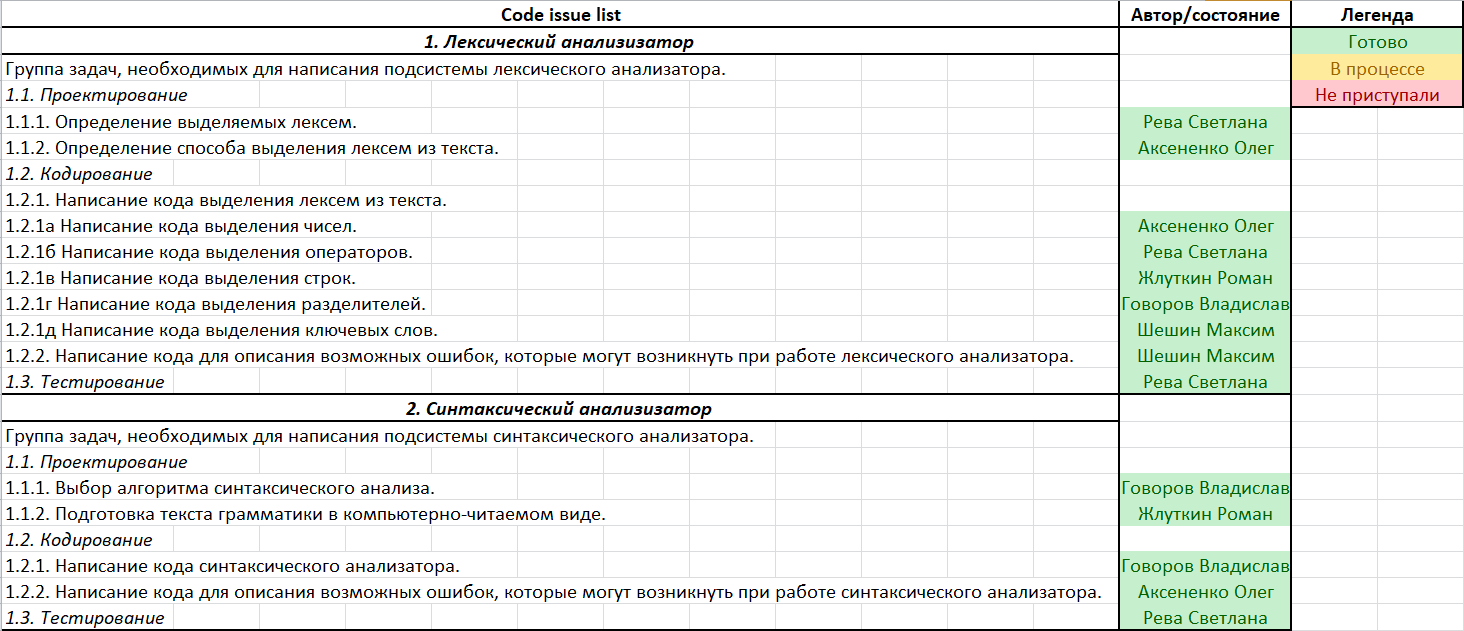


Рисунок 25 – Перечень задач проекта (начало)



Рисунок 26 – Перечень задач проекта (окончание)

# 9 Coding Recommendations (рекомендации по кодированию)

* Запреты:
* при написании комментариев запрещается использовать какой-либо язык, отличный от английского.
* Рекомендации:
* рекомендуется, чтобы все неошибочные константы перечислимых типов (enum) начинались с прописной буквы, например:

*class Construction(Enum):*

*Condition = auto()*

*Else = auto()*

*Loop = auto()*

*Function = auto();*

* рекомендуется, чтобы все ошибочные константы перечислимых типов (enum) были в верхнем регистре, например:

*class Type(Enum):*

*Identifier = 0*

*Keyword = 1*

*Operator = 2*

*Divider = 3*

*Number = 4*

*String = 5*

*ERROR = -1;*

* рекомендуется, чтобы длина строки кода не превышала 120 символов.
* Требования:
* классы должны разделяться двумя пустыми строками;
* функции класса должны разделяться пустой строкой;
* каждый файл с кодом проекта должен заканчиваться пустой строкой.

**Чек-лист**

|  |  |  |
| --- | --- | --- |
| **№** | **Контрольное выражение** | **Соответствие** |
| 1 | В коде отсутствуют комментарии на английском языке |  |
| 2 | Неошибочные константы перечислимых типов (enum) начинаются с прописной буквы |  |
| 3 | Ошибочные константы перечислимых типов (enum) находятся в верхнем регистре |  |
| 4 | В коде нет строк, длина которых превышает 120 символов |  |
| 5 | Все классы разделены двумя пустыми строками |  |
| 6 | Все функции классов разделены пустой строкой |  |
| 7 | Каждый файл с кодом проекта заканчивается пустой строкой |  |

# 10 Software Test Plan (план тестирования проекта)

**Тесты для тестирования подсистемы «Пользовательский интерфейс»**

*Тест TEST\_UI\_001*

Тестируемые требования: REQ\_UI\_001

Описание теста: загрузка текста программы в соответствующее текстовое поле

*Тест TEST\_UI\_002*

Тестируемые требования: REQ\_UI\_002

Описание теста: трансляция текста программы при нажатии на кнопку «Трансляция»

**Тесты для тестирования подсистемы «Лексический анализатор»**

*Тест TEST\_LEX\_001*

Тестируемые требования: REQ\_LEX\_001

Описание теста: программа без лексических ошибок

*Тест TEST\_LEX\_002*

Тестируемые требования: REQ\_LEХ\_001, REQ\_LEX\_002

Описание теста: программа с недопустимым символом

**Тесты для тестирования подсистемы «Синтаксический анализатор»**

*Тест TEST\_SINT\_001*

Тестируемые требования: REQ\_SINT\_001

Описание теста: программа без синтаксических ошибок

*Тест TEST\_SINT\_002*

Тестируемые требования: REQ\_SINT\_001, REQ\_SINT\_002

Описание теста: программа с синтаксической ошибкой в условии

*Тест TEST\_SINT\_003*

Тестируемые требования: REQ\_SINT\_001, REQ\_SINT\_002

Описание теста: программа с синтаксической ошибкой в цикле

*Тест TEST\_SINT\_004*

Тестируемые требования: REQ\_SINT\_001

Описание теста: программа с пустой функцией

*Тест TEST\_SINT\_005*

Тестируемые требования: REQ\_SINT\_001

Описание теста: программа с функцией с несколькими аргументами

**Тесты для тестирования подсистемы «Семантический анализатор»**

*Тест TEST\_SEM\_001*

Тестируемые требования: REQ\_SEM\_001, REQ\_SEM\_002

Описание теста: ошибка «идентификатор переменной встречен до её объявления»

*Тест TEST\_SEM\_002*

Тестируемые требования: REQ\_SEM\_001, REQ\_SEM\_002

Описание теста: ошибка «функция вызвана до её описания»

*Тест TEST\_SEM\_003*

Тестируемые требования: REQ\_SEM\_001, REQ\_SEM\_002

Описание теста: ошибка «количество формальных и фактических параметров функции не совпадает»

*Тест TEST\_SEM\_004*

Тестируемые требования: REQ\_SEM\_001, REQ\_SEM\_002

Описание теста: ошибка «в контексте вызываемой функции встречена неизвестная переменная»

*Тест TEST\_SEM\_005*

Тестируемые требования: REQ\_SEM\_001, REQ\_SEM\_002

Описание теста: ошибка «в контексте вызываемой функции встречена неизвестная функция»

**Тесты для тестирования подсистемы «Генератор кода»**

*Тест TEST\_CG\_001*

Тестируемые требования: REQ\_CG\_001

Описание теста: генерация циклов

*Тест TEST\_CG\_002*

Тестируемые требования: REQ\_CG\_001

Описание теста: генерация условий

*Тест TEST\_CG\_003*

Тестируемые требования: REQ\_CG\_001

Описание теста: генерация логических операций

*Тест TEST\_CG\_004*

Тестируемые требования: REQ\_CG\_001

Описание теста: генерация операторов

*Тест TEST\_CG\_005*

Тестируемые требования: REQ\_CG\_001

Описание теста: генерация математических операций

*Тест TEST\_CG\_006*

Тестируемые требования: REQ\_CG\_001

Описание теста: генерация функций

**Тесты для тестирования системы в целом (System Test)**

*Тест TEST\_SYS\_001*

Тестируемые требования: REQ\_UI\_001, REQ\_UI\_002, REQ\_LEX\_001, REQ\_SINT\_001

Описание теста: пустая программа

*Тест TEST\_SYS\_002*

Тестируемые требования: REQ\_UI\_001, REQ\_UI\_002, REQ\_LEX\_001, REQ\_SINT\_001, REQ\_SEM\_001, REQ\_CG\_001

Описание теста: корректная непустая программа

# 11 Requirements Traceability Matrix (матрица покрытия тестами требований)

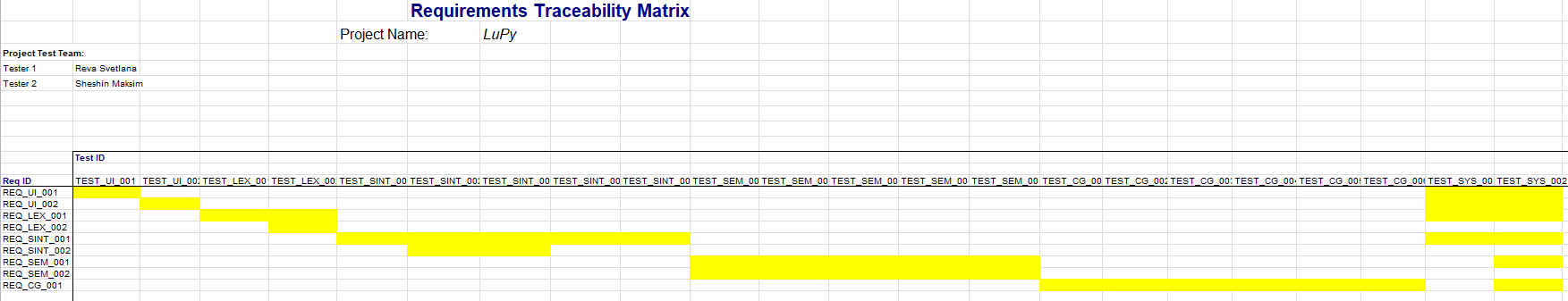


Рисунок 27 – Матрица покрытия тестами требований

# 12 Software Test Logs (результаты выполнения тестов)

**Тесты для тестирования подсистемы «Пользовательский интерфейс»**

*Тест TEST\_UI\_001*

Тестируемые требования: REQ\_UI\_001

Описание теста: загрузка текста программы в соответствующее текстовое поле

Ожидаемый результат: загрузка текста программы производится без ошибок

Видимый результат: загрузка текста программы производится без ошибок

Резюме: Тест пройден

*Тест TEST\_UI\_002*

Тестируемые требования: REQ\_UI\_002

Описание теста: трансляция текста программы при нажатии на кнопку «Трансляция»

Ожидаемый результат: при нажатии на кнопку трансляция производится без ошибок

Видимый результат: при нажатии на кнопку трансляция производится без ошибок

Резюме: Тест пройден

**Тесты для тестирования подсистемы «Лексический анализатор»**

*Тест TEST\_LEX\_001*

Тестируемые требования: REQ\_LEX\_001

Описание теста: программа без лексических ошибок

print("Hello!")

Ожидаемый результат: Status: SUCCESS

Видимый результат: Status: SUCCESS

Резюме: Тест пройден

*Тест TEST\_LEX\_002*

Тестируемые требования: REQ\_LEХ\_001, REQ\_LEX\_002

Описание теста: программа с недопустимым символом

#print("Hello!")

Ожидаемый результат: Status: ERROR

Description: Lexical Error

Incorrect code in position 1 line 1: #print("Hello!")

↑

Видимый результат: Status: ERROR

Description: Lexical Error

Incorrect code in position 1 line 1: #print("Hello!")

↑

Резюме: Тест пройден

**Тесты для тестирования подсистемы «Синтаксический анализатор»**

*Тест TEST\_SINT\_001*

Тестируемые требования: REQ\_SINT\_001

Описание теста: программа без синтаксических ошибок

a = 2

if a > 0:

print('Ok')

Ожидаемый результат: Status: SUCCESS

Видимый результат: Status: SUCCESS

Резюме: Тест пройден

*Тест TEST\_SINT\_002*

Тестируемые требования: REQ\_SINT\_001, REQ\_SINT\_002

Описание теста: программа с синтаксической ошибкой в условии

a = 2

if a > 0

print('Ok')

Ожидаемый результат: Status: ERROR

Description: Syntactic Error

Unable to parse code using Earley Parser

Видимый результат: Status: ERROR

Description: Syntactic Error

Unable to parse code using Earley Parser

Резюме: Тест пройден

*Тест TEST\_SINT\_003*

Тестируемые требования: REQ\_SINT\_001, REQ\_SINT\_002

Описание теста: программа с синтаксической ошибкой в цикле

for i range(2):

print(i)

Ожидаемый результат: Status: ERROR

Description: Syntactic Error

Unable to parse code using Earley Parser

Видимый результат: Status: ERROR

Description: Syntactic Error

Unable to parse code using Earley Parser

Резюме: Тест пройден

*Тест TEST\_SINT\_004*

Тестируемые требования: REQ\_SINT\_001

Описание теста: программа с пустой функцией

a = 1

print(a)

def foo():

pass

Ожидаемый результат: Status: SUCCESS

Видимый результат: Status: SUCCESS

Резюме: Тест пройден

*Тест TEST\_SINT\_005*

Тестируемые требования: REQ\_SINT\_001

Описание теста: программа с функцией с несколькими аргументами

a = 1

print(a)

def foo():

pass

def baz(f, c, m):

print(f)

baz(a, a, a)

Ожидаемый результат: Status: SUCCESS

Видимый результат: Status: SUCCESS

Резюме: Тест пройден

**Тесты для тестирования подсистемы «Семантический анализатор»**

*Тест TEST\_SEM\_001*

Тестируемые требования: REQ\_SEM\_001, REQ\_SEM\_002

Описание теста: ошибка «идентификатор переменной встречен до её объявления»

print(a)

Ожидаемый результат: Status: ERROR

Description: Semantic Error

The identifier was encountered before it was announced:

Token:

token\_type = Type.Identifier

content = a

line = 1

position = 7

Видимый результат: Status: ERROR

Description: Semantic Error

The identifier was encountered before it was announced:

Token:

token\_type = Type.Identifier

content = a

line = 1

position = 7

Резюме: Тест пройден

*Тест TEST\_SEM\_002*

Тестируемые требования: REQ\_SEM\_001, REQ\_SEM\_002

Описание теста: ошибка «функция вызвана до её описания»

foo()

Ожидаемый результат: Status: ERROR

Description: Semantic Error

The function identifier was used before it was announced:

Token:

token\_type = Type.Identifier

content = foo

line = 1

position = 1

Видимый результат: Status: ERROR

Description: Semantic Error

The function identifier was used before it was announced:

Token:

token\_type = Type.Identifier

content = foo

line = 1

position = 1

Резюме: Тест пройден

*Тест TEST\_SEM\_003*

Тестируемые требования: REQ\_SEM\_001, REQ\_SEM\_002

Описание теста: ошибка «количество формальных и фактических параметров функции не совпадает»

def foo(a, b):

return a + b

foo(2)

Ожидаемый результат: Status: ERROR

Description: Semantic Error

Parameters in the declaration and function call do not match:

Token:

token\_type = Type.Identifier

content = foo

line = 3

position = 1

Видимый результат: Status: ERROR

Description: Semantic Error

Parameters in the declaration and function call do not match:

Token:

token\_type = Type.Identifier

content = foo

line = 3

position = 1

Резюме: Тест пройден

*Тест TEST\_SEM\_004*

Тестируемые требования: REQ\_SEM\_001, REQ\_SEM\_002

Описание теста: ошибка «в контексте вызываемой функции встречена неизвестная переменная»

def foo(a, b):

a = c + 2

return a + b

foo(3, 2)

Ожидаемый результат: Status: ERROR

Description: Semantic Error

When the function was called, the variable used in it was not declared:

Token:

token\_type = Type.Identifier

content = foo

line = 4

position = 1

Видимый результат: Status: ERROR

Description: Semantic Error

When the function was called, the variable used in it was not declared:

Token:

token\_type = Type.Identifier

content = foo

line = 4

position = 1

Резюме: Тест пройден

*Тест TEST\_SEM\_005*

Тестируемые требования: REQ\_SEM\_001, REQ\_SEM\_002

Описание теста: ошибка «в контексте вызываемой функции встречена неизвестная функция»

def foo(a, b):

a = foo2() + 2

return a + b

foo(3, 2)

Ожидаемый результат: Status: ERROR

Description: Semantic Error

When the function was called, the function name used in it was not declared:

Token:

token\_type = Type.Identifier

content = foo

line = 4

position = 1

Видимый результат: Status: ERROR

Description: Semantic Error

When the function was called, the function name used in it was not declared:

Token:

token\_type = Type.Identifier

content = foo

line = 4

position = 1

Резюме: Тест пройден

**Тесты для тестирования подсистемы «Генератор кода»**

*Тест TEST\_CG\_001*

Тестируемые требования: REQ\_CG\_001

Описание теста: генерация циклов

a = 0

while a < 10:

print('iteration')

a = a + 1

Ожидаемый результат:

a = 0

while a < 10 do

print('iteration')

a = a + 1

end

Видимый результат:

a = 0

while a < 10 do

print('iteration')

a = a + 1

end

Резюме: Тест пройден

*Тест TEST\_CG\_002*

Тестируемые требования: REQ\_CG\_001

Описание теста: генерация условий

b = 16

if b == 16:

b = 2

elif b == 2:

b = 1

else:

b = 0

Ожидаемый результат:

b = 16

if b == 16 then

b = 2

elseif b == 2 then

b = 1

else

b = 0

end

Видимый результат:

b = 16

if b == 16 then

b = 2

elseif b == 2 then

b = 1

else

b = 0

end

Резюме: Тест пройден

*Тест TEST\_CG\_003*

Тестируемые требования: REQ\_CG\_001

Описание теста: генерация логических операций

a = True

b = False

c = a or b

Ожидаемый результат:

a = true

b = false

c = a or b

Видимый результат:

a = true

b = false

c = a or b

Резюме: Тест пройден

*Тест TEST\_CG\_004*

Тестируемые требования: REQ\_CG\_001

Описание теста: генерация операторов

a = 1

if a != 3:

if a < 3:

a = 3

else:

a = 2

Ожидаемый результат: a = 1

if a ~= 3 then

if a < 3 then

a = 3

else

a = 2

end

end

Видимый результат:

a = 1

if a ~= 3 then

if a < 3 then

a = 3

else

a = 2

end

end

Резюме: Тест пройден

*Тест TEST\_CG\_005*

Тестируемые требования: REQ\_CG\_001

Описание теста: генерация математических операций

a = 5 \*\* 5 % 5 + 5 - 5 \* 5 / 10

Ожидаемый результат:

a = 5 ^ 5 % 5 + 5 - 5 \* 5 / 10

Видимый результат:

a = 5 ^ 5 % 5 + 5 - 5 \* 5 / 10

Резюме: Тест пройден

*Тест TEST\_CG\_006*

Тестируемые требования: REQ\_CG\_001

Описание теста: генерация функций

def func(a, b, c):

return a + b \* c

print(func(1, 2, 3))

Ожидаемый результат:

function func(a, b, c)

return a + b \* c

end

print(func(1, 2, 3))

Видимый результат:

function func(a, b, c)

return a + b \* c

end

print(func(1, 2, 3))

Резюме: Тест пройден

**Тесты для тестирования системы в целом (System Test)**

*Тест TEST\_SYS\_001*

Тестируемые требования: REQ\_UI\_001, REQ\_UI\_002, REQ\_LEX\_001, REQ\_SINT\_001

Описание теста: пустая программа

Ожидаемый результат: пустой файл с расширением .lua

Видимый результат: пустой файл с расширением .lua

Резюме: Тест пройден

*Тест TEST\_SYS\_002*

Тестируемые требования: REQ\_UI\_001, REQ\_UI\_002, REQ\_LEX\_001, REQ\_SINT\_001, REQ\_SEM\_001, REQ\_CG\_001

Описание теста: корректная непустая программа

Ожидаемый результат: отсутствие ошибок и корректная работа всех подсистем

Видимый результат: отсутствие ошибок и корректная работа всех подсистем

Резюме: Тест пройден

# Заключение

В рамках работы было разработано программное средство-транслятор, который преобразует программу, содержащую подмножество языка Python, в программу, содержащую подмножество языка Lua, генерируя эквивалентный исходный код, а также написана сопутствующая документация проекта.

Кроме того, работа над проектом была оценена в соответствии с введёнными в разделе «Программа измерений проекта» метриками:

1. Система метрик для оценки эффективности процесса разработки

* Метрика «On project % time» (OPPT)

OPPT = (Рабочее время, затраченное на задачу / Общее время, выделенное на задачу по плану) \* 100% = (4/7) \* 100% ~ 57%

* Метрика «Productivity»

Productivity = LOC / Рабочее время, затраченное на проект = 50 / 5 = 10 LOC / человеко-час

* Метрика «Problem Resolution Rate» (PRR)

PRR = 4 дня

Все вышеперечисленные метрики показывают, что сроки выполнения проекта адекватны.

1. Система метрик для оценки качества программного продукта, создаваемого в проекте

* Метрика «In Process Faults» (IPF)

IPF = Число обнаруженных ошибок до выпуска его релиза при тестировании / LOC = 5 / 200 = 0,025 неполадок / LOC

Данная метрика показывает, что разрабатываемый в рамках проекта продукт соответствует стандартам качества, поскольку её значение близко к нормативному (0,015-0,02 неполадок / LOC).