

# Лабораторная работа по дисциплине

## «Автоматизация научных исследований»

Соломатов Александр Денисович

Гр. 5040102/50201

December 2025

## 1 Цель работы

Разработать и проанализировать диаграмму UML (а или б) для информационной системы по выбору студента.

## 2 Задание

- Часть а. Диаграмма вариантов использования
  1. Выбрать предметную область для проектирования информационной системы
  2. Определить основных действующих лиц системы
  3. Выявить основные функциональные требования
  4. Построить диаграмму вариантов использования с учетом следующих требований:
    - Отобразить все действующие лица
    - Определить основные и вспомогательные варианты использования
    - Установить отношения между вариантами использования (include, extend)
    - Добавить границу системы
    - Обеспечить логическую целостность диаграммы
- Часть б. Диаграмма классов
  1. Выбрать предметную область для проектирования информационной системы
  2. Определить основных действующих лиц системы
  3. Выявить основные функциональные требования
  4. Построить диаграмму вариантов использования с учетом следующих требований:
    - На основе выбранной предметной области разработать диаграмму классов

- Определить основные классы системы
- Установить иерархию наследования
- Определить типы связей между классами
- Указать атрибуты классов с типами данных
- Добавить кратности ассоциаций

### 3 Требования к выполнению

1. Использовать PlantUML для генерации диаграммы
2. Соблюдать стандарты UML
3. Обеспечить логическую целостность диаграммы
4. Предоставить комментарии к ключевым элементам диаграммы
5. Проверить работоспособность сгенерированного кода
6. Провести исследование для 1-2 моделей ИИ

### 4 Критерии оценки

1. Корректность синтаксиса PlantUML
2. Соответствие стандартам UML
3. Полнота отображения функциональных требований
4. Качество визуализации
5. Логическая целостность диаграмм

### 5 Описание моделируемой системы

Информационная система **DSLTools** представляет собой инструментальную среду разработки языков. Система предназначена для автоматизации процесса создания интерпретаторов предметно-ориентированных языков (DSL) на основе формальных спецификаций.

### 6 Описание моделируемой системы

Информационная система **DSLTools** представляет собой Language Workbench — среду для проектирования и выполнения предметно-ориентированных языков. Система позволяет описывать синтаксис на языке EBNF и реализовывать семантику на основе паттерна Visitor. Особенностью системы является разделение между стадией проектирования (интерпретация метаданных) и стадией выполнения целевых программ.

## 6.1 Действующие лица и варианты использования

1. **Пользователь системы (BaseUser)** — базовый актор, представляющий любого субъекта, взаимодействующего с интерфейсом среды.
  - Выполнить программу на DSL (RunDSL) — запуск кода на целевом языке, что автоматически требует подгрузки сгенерированного интерпретатора.
2. **Пользователь DSL (TargetUser)** — конечный потребитель созданного языка, использующий систему как готовую среду исполнения.
  - Наследует все возможности базового пользователя, фокусируясь на прикладном применении созданных инструментов.
3. **Разработчик DSL (Creator)** — основной специалист, отвечающий за создание архитектуры нового языка.
  - Задать описание синтаксиса (EBNF) — написание текстового файла с правилами грамматики.
  - Задать семантику (Visitor) — определение логики обхода дерева (вручную или через ИИ-генератор).
  - Сгенерировать интерпретатор (Build) — инициирование сборки, которая включает внутренние процессы интерпретации грамматики и семантики.
  - Визуализировать CST/AST дерево — диагностический инструмент для анализа структуры целевого-языка или целевого кода.
4. **Сильный разработчик (Strong Creator)** — эксперт, имеющий доступ к низкоуровневым настройкам самой системы DSLTools.
  - Модифицировать модуль ядра — внесение изменений в базовые алгоритмы токенизации, построения CST, трансляции в AST или в саму последовательность шагов интерпретации.

## 6.2 P.S

Это описание написала Gemini по Use-Case Diagram, правильной на мой взгляд (ну почти). Мы ее делали вместе, точнее нейросеть боролась с PlantUML, а я боролся с Gemini и UML.

Вот выстраданный код:

### Листинг

```
@startuml  
' Жесткое управление слоями
```

```

    top to bottom direction
    skinparam actorStyle human
    skinparam shadowing false

    ' --- АКЫТОР ---
    actor Пользователь" системы" as BaseUser
    actor "Strong Creator" as StrongCreator
    actor Разработчик" DSL" as Creator
    actor Пользователь" DSL" as TargetUser

    ' Иерархия (UML_5)
    StrongCreator -up-> Creator
    Creator -up-> BaseUser
    TargetUser -up-> BaseUser

    ' Фиксация позиций Верх( / Низ)
    BaseUser -[hidden]down- DSLTools
    DSLTools -[hidden]down- TargetUser

rectangle Система" DSLTools" as DSLTools {

    ' --- ЭШЕЛОН 1 Входы() ---
    usecase UC_Build as Сгенерировать" интерпретатор
    --
    extension points:
    Structure_Analysis"
    usecase UC_RunDSL as Выполнить" программу на DSL
    --
    extension points:
    Debug_View"

    ' --- ЭШЕЛОН 2 (Include) ---
    usecase UC_InterpGrammarDescr as Проинтерпретировать"
    заданную грамматику"
    usecase UC_InterpSemanic as Проинтерпретировать" семантику"
    usecase UC_LoadDSLInterpreter as Подгрузить" интерпретатор
    DSL"
    usecase UC_Grammar as Задать" синтаксис + лексику (EBNF)"
    usecase UC_Semantics as Задать" семантику (Visitor)"

    ' --- ЭШЕЛОН 3 (Base Definitions) ---
    usecase UC_Grammar as Задать" синтаксис + лексику (EBNF)"
    usecase UC_Semantics as Задать" семантику (Visitor)"
}

```

```

' --- ЭШЕЛОН 4 (Sub-methods / Core) ---
usecase UC_Manual as Ручная" регистрация функций"
usecase UC_AI as Автогенерация" через ИИ"

usecase UC_Modify as Модифицировать" модуль ядра"
usecase UC_Tokenizer_Algho as Алгоритм" токенизатора"
usecase UC_CSTBuilder_Algho as Алгоритм" построения дерева
разбора"
usecase UC_ASTTranslator_Algho as Алгоритм" трансляции в AST"
usecase UC_Sequence_Edit as Расширение" шагов интерпретации"

' --- ЭШЕЛОН 5 ---
usecase UC_ViewTree as Визуализировать" CST/AST дерево"

' --- ВЕРСТКА Скрытые( связи для соосности) ---
UC_Build -[hidden]right- UC_RunDSL
UC_Build -[hidden]down- UC_InterpGrammarDescr
UC_InterpGrammarDescr -[hidden]down- UC_Grammar
UC_Grammar -[hidden]down- UC_Manual
UC_Semantics -[hidden]down- UC_Modify

' --- ЛОГИКА ---
UC_Build ..> UC_InterpGrammarDescr : <<include>>
UC_Build ..> UC_InterpSemantic : <<include>>
UC_RunDSL ..> UC_LoadDSLInterpreter : <<include>>

UC_Manual -up-|> UC_Semantics
UC_AI -up-|> UC_Semantics

UC_Tokenizer_Algho -up-|> UC_Modify
UC_CSTBuilder_Algho -up-|> UC_Modify
UC_ASTTranslator_Algho -up-|> UC_Modify
UC_Sequence_Edit -up-|> UC_Modify

UC_ViewTree .up.> UC_Build : <<extend>>
note on link
condition: {inspect meta}
extension point: Structure_Analysis
end note

UC_ViewTree .up.> UC_RunDSL : <<extend>>
note on link

```

```

condition: {inspect target}
extension point: Debug_View
end note
}

' --- СВЯЗИ АКТОРОВ ---
' Привязываем BaseUser Верх()
BaseUser -down- UC_RunDSL

' Разработчики Слева( -> Направо)
StrongCreator -right- UC_Modify
Creator -right- UC_Semantics
Creator -right- UC_Grammar
Creator -right- UC_Build

@enduml

```

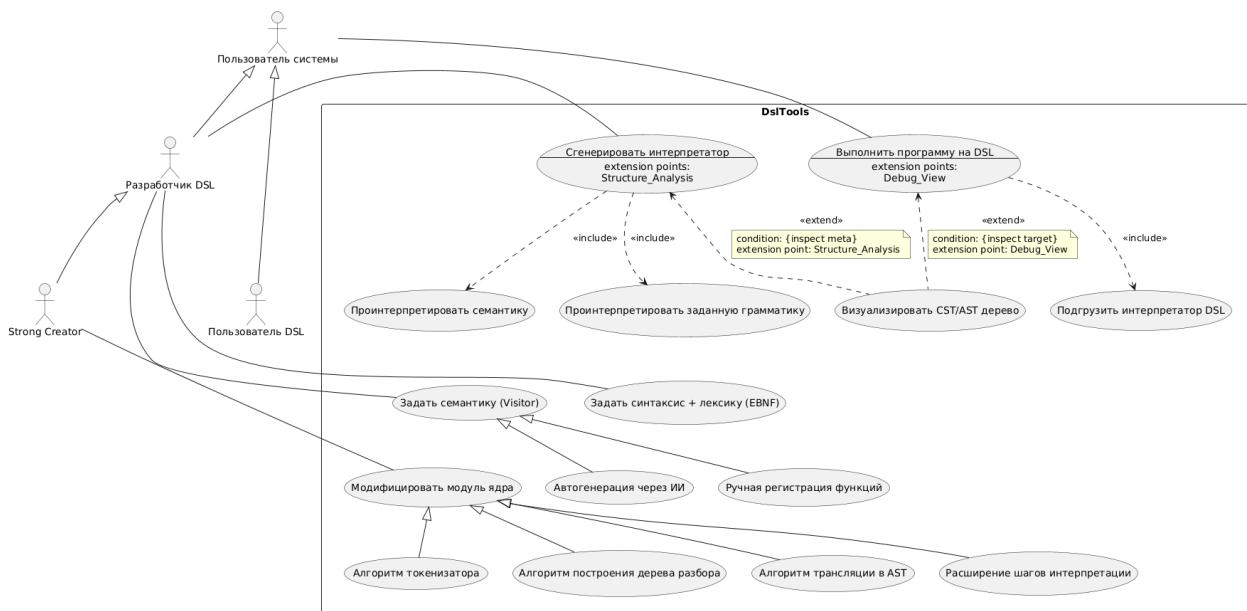


Figure 1: Описание диаграммы использования для DSLTools.

Итак, по поводу использования нейросети в таких вещах заранее внесу гипотезу о том, что ничего не выйдет, во всяком случае для цепочек запросов с итерационным изменением картинки, однако если есть на руках почти полное описание системы то все может удачно выйти!

## 7 Промты

### 7.1 Промт 1

#### Листинг

Промт №1 Нулевой( уровень - "Basic"): Сгенерируй PlantUML код для Use Case диаграммы системы DSLTools. В системе есть Разработчик и Пользователь. Разработчик создает целевую грамматику и собирает интерпретатор, а Пользователь запускает программы на целевом языке. Также должна быть возможность смотреть деревья AST/CST.

### 7.2 Промт 2

#### Листинг

Промт №2 Первый( уровень - "Structured"): Напиши PlantUML код диаграммы вариантов использования.

Акторы: BaseUser, TargetUser, Creator, StrongCreator.

Реализуй иерархию акторов.

Основные функции: запускает программы на целевом языке, описание EBNF грамматики-, задание семантики Visitor, сборка интерпретатора и выполнение DSLкода-.

StrongCreator может менять алгоритмы ядра.

Используй rectangle для границы системы.

### 7.3 Промт 3

#### Листинг

Промт №3 Второй( уровень - "Natural Language Descriptive"): Сгенерируй PlantUML код для Use Case диаграммы системы разработки языков DSLTools.

Пользователь системы: запускает программы на целевом языке, что всегда требует автоматической подгрузки интерпретатора.

Пользователь DSL: человек - обобщение над пользователем системы.

Разработчик DSL: проектировщик целевого языка. Он задает грамматику с помощью (EBNF) и регистрирует правила семантики в Visitor. При запуске им команды Сгенерировать" интерпретатор" система должна обязательно

проинтерпретировать описание грамматики и семантики. Реализуй Задание "семантики" через два варианта: ручной ввод или ИИгенерацию-.

Сильный разработчик: эксперт, который может менять алгоритмы ядра токенизатор(, CSTбилдер-, ASTтранслятор-, а также расширить шаги, которые происходят при интерпретации целевого языка).

Добавь возможность визуализации деревьев (CST/AST) как optionalное расширение для процессов сборки интерпретатора и выполнения программ.

Связывай акторов только с основными кейсами, внутренние процессы интерпретации оставь без прямых связей с людьми.

Заключи всё в границу системы.

## 7.4 Промт 4

### Листинг

Промт

№4 Архитектурный( мастер):

Действуй как Senior Software Architect и эксперт по UML 2.5.

Сгенерируй безупречный PlantUML код диаграммы Use Case для системы DSLTools.

Прежде чем выдать код, сформируй аналитическую сводку в формате LaTeX по следующему шаблону:

```
% ===== НАЧАЛО ВЫВОДА В ФОРМАТЕ LaTeX =====
\section{Аналитическая}{сводка проектирования архитектуры}

\subsection{1. Цепочка мыслей (Reasoning Chain)}
\begin{longtable}{p{0.20\linewidth}|p{0.75\linewidth}}
\toprule
\textrm{Этап}{ анализа} & \textrm{Логическое}{\\
обоснование и принятые архитектурные решения} \\
\midrule
\textrm{Роли}{} & Обоснование[ иерархии акторов и\\
экспертности StrongCreator] \\
\textrm{Инкапсуляция}{} & Обоснование[ отделения\\
внутренних ивентов системы от действий акторов через include] \\
\textrm{Связи}{} & Обоснование[ использования\\
Generalization для алгоритмов и extend для диагностики] \\
\end{longtable}
```

```

\textrm{Визуализация}{} & Логика[ эшелонирования и
управления направлением связей для чистоты верстки] \\%
\bottomrule
\end{longtable}

\subsection*{2. Принятые решения по верстке}
\begin{itemize}
\item \textbf{Эшелонирование}: Распределение кейсов
по 5 уровням для исключения пересечений.
\item \textbf{Позиционирование}: Принудительное
закрепление акторов по осям Верх(, Лево, Низ).
\end{itemize}
\end{itemize}
% ===== КОНЕЦ ВЫВОДА В ФОРМАТЕ LaTeX =====

```

Требования к диаграмме описание( логики):

1. Акторы: StrongCreator наследует Creator, Creator наследует BaseUser, TargetUser наследует BaseUser.
2. Верстка: Сверху - BaseUser связь( вниз), Снизу - TargetUser связь( вверх), Слева - Creator и StrongCreator связи( вправо). Используй 'top to bottom direction'.
  - Эшелон 1 Входы(): Сгенерировать" интерпретатор" и Выполнить" программу на DSL".
  - Эшелон 2 Системные( процессы): Проинтерпретировать" грамматику", Проинтерпретировать" семантику", Подгрузить" интерпретатор".
  - Эшелон 3 Определения(): Задать" синтаксис + лексику", Задать" семантику (Visitor)".
  - Эшелон 4 Реализации(): Ручная" регистрация" и Автогенерация" через ИИ" наследуются( от Задать" семантику").
  - Эшелон 5 Ядро(): Модифицировать" модуль ядра" база( для наследования алгоритмов токенизатора, CSTбилдера-, ASTтранслятора- и шагов интерпретации).
3. Отношения:
  - Сгенерировать интерпретатор включает (include) процессы интерпретации грамматики и семантики.
  - Выполнить программу включает (include) подгрузку интерпретатора.
  - Визуализировать дерево расширяет (extend) Сгенерировать и Выполнить добавь( Extension Points в базовые кейсы и условия condition в заметках на линиях).
4. СТРОГИЙ ЗАПРЕТ: Не рисуй ассоциации от акторов к любым событиям, которые являются частью отношений <<include>>, <<extend>> или являются наследниками потомками() в отношениях обобщения. Акторы работают только с верхушками"" и базами определений.

5. Выравнивание: Используй невидимые связи - [hidden]down- для удержания эшелонов друг под другом.

## 8 Промт 1

### 8.1 DeepSeek

Листинг

```
@startuml
left to right direction
skinparam packageStyle rect

actor Разработчик
actor Пользователь

rectangle DSLTools {
    usecase Создать( грамматику) as UC1
    usecase Собрать( интерпретатор) as UC2
    usecase Запустить( программу) as UC3
    usecase Просмотреть( AST) as UC4
    usecase Просмотреть( CST) as UC5
}

Разработчик --> UC1
Разработчик --> UC2
Пользователь --> UC3
Пользователь --> UC4
Пользователь --> UC5

UC1 .> UC2 : включает
UC3 .> UC4 : может потребовать
UC3 .> UC5 : может потребовать

note right of UC4
Абстрактное синтаксическое
дерево (Abstract Syntax Tree)
end note

note right of UC5
```

Конкретное синтаксическое дерево (Concrete Syntax Tree)  
end note  
@enduml

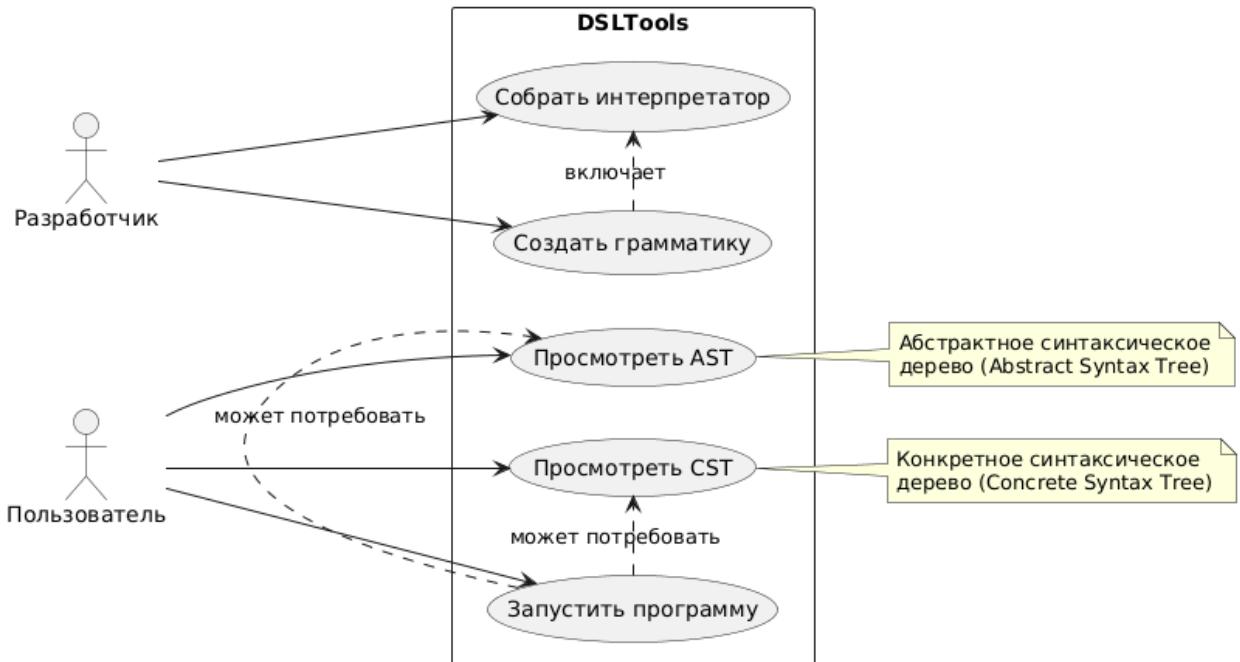


Figure 2

## 8.2 Gemini

### Листинг

```

@startuml
left to right direction
skinparam packageStyle rectangle

actor Разработчик"" as Dev
actor Пользователь"" as User

rectangle Система" DSLTools" {
    usecase Создать" целевую грамматику" as UC1
    usecase Собрать" интерпретатор" as UC2
    usecase Запустить" программу на целевом языке" as UC3
    usecase Просмотреть" деревья AST/CST" as UC4
}

```

Dev --> UC1

Dev --> UC2

Dev --> UC4

User --> UC3

User --> UC4

@enduml

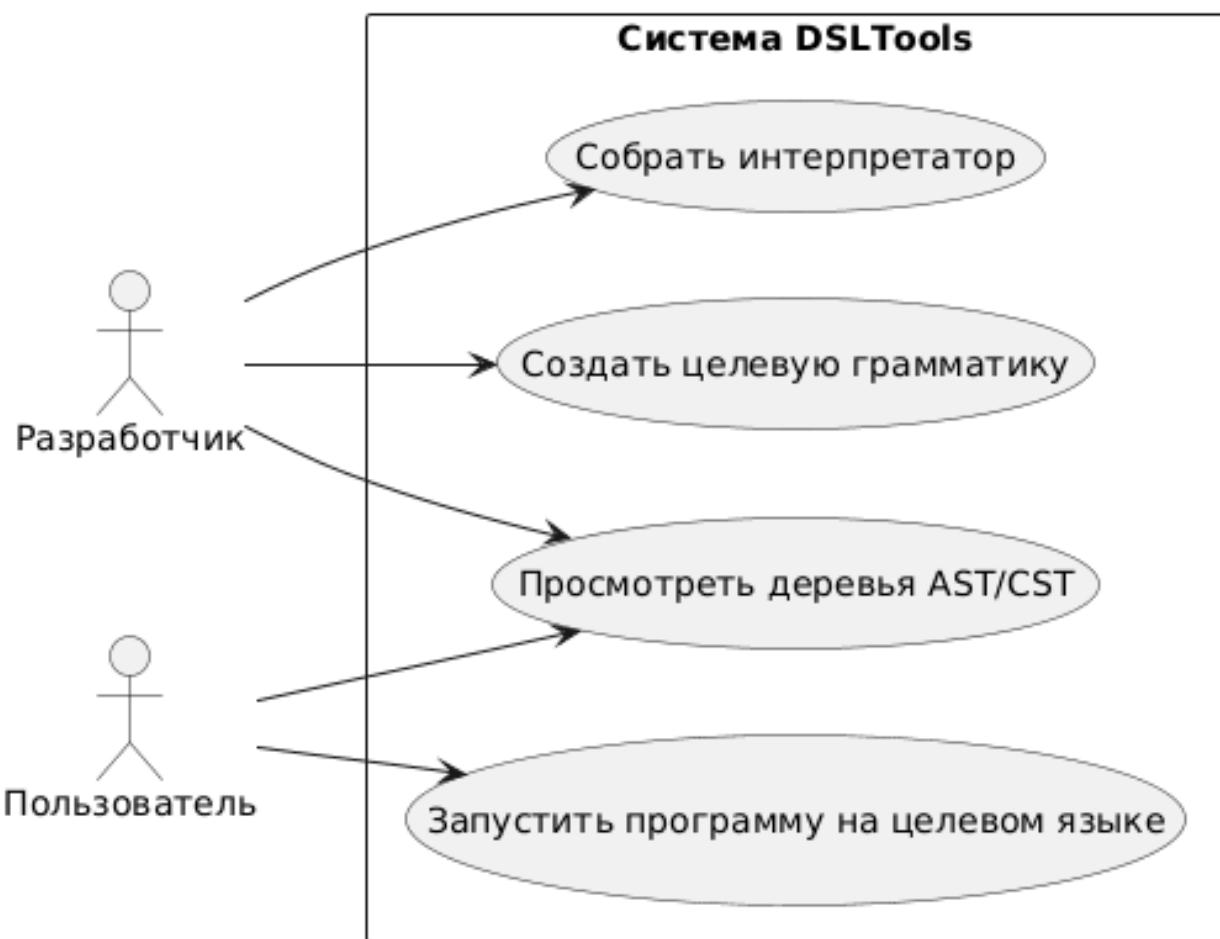


Figure 3

## 9 Промт 2

### 9.1 DeepSeek

До исправления

## Листинг

```
@startuml
left to right direction
skinparam actorStyle awesome

rectangle Система {

    Запускает( программы на целевом языке) as (use1)
    Описание( EBNFграмматики-) as (use2)
    Задание( семантики через Visitor) as (use3)
    Сборка( интерпретатора) as (use4)
    Выполнение( DSLкода-) as (use5)
    Изменение( алгоритмов ядра) as (use6)

}

actor BaseUser
actor TargetUser Пользователь<<>>
actor Creator Работчик<<>>
actor StrongCreator Расширенный<< разработчик>>

BaseUser <|-- TargetUser
BaseUser <|-- Creator
Creator <|-- StrongCreator

BaseUser --> (use1)

TargetUser --> (use1)
TargetUser --> (use5)

Creator --> (use2)
Creator --> (use3)
Creator --> (use4)

StrongCreator --> (use6)
@enduml
```

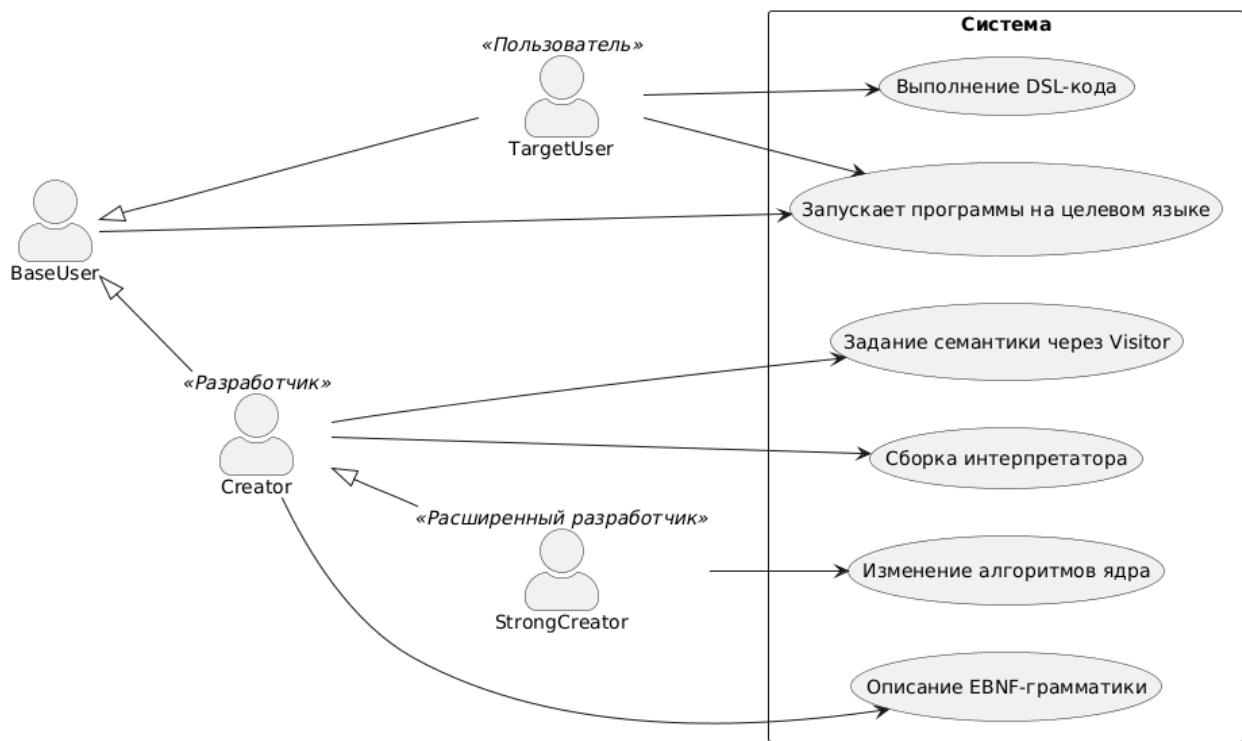


Figure 4

## После исправления

### Листинг

Неправильно, исправь человечков согласно стандарту UML  
Ассоциация должна не иметь стрелки на конце

### Листинг

```
@startuml
left to right direction

rectangle Система {
    Запускает( программы на целевом языке) as (use1)
    Описание( EBNFграмматики-) as (use2)
    Задание( семантики через Visitor) as (use3)
    Сборка( интерпретатора) as (use4)
    Выполнение( DSLкода-) as (use5)
    Изменение( алгоритмов ядра) as (use6)
}
```

```

actor BaseUser
actor TargetUser Пользователь<<>>
actor Creator Развработчик<<>>
actor StrongCreator Расширенный<< разработчик>>

BaseUser <|-- TargetUser
BaseUser <|-- Creator
Creator <|-- StrongCreator

BaseUser -- (use1)

TargetUser -- (use1)
TargetUser -- (use5)

Creator -- (use2)
Creator -- (use3)
Creator -- (use4)

StrongCreator -- (use6)
@enduml

```

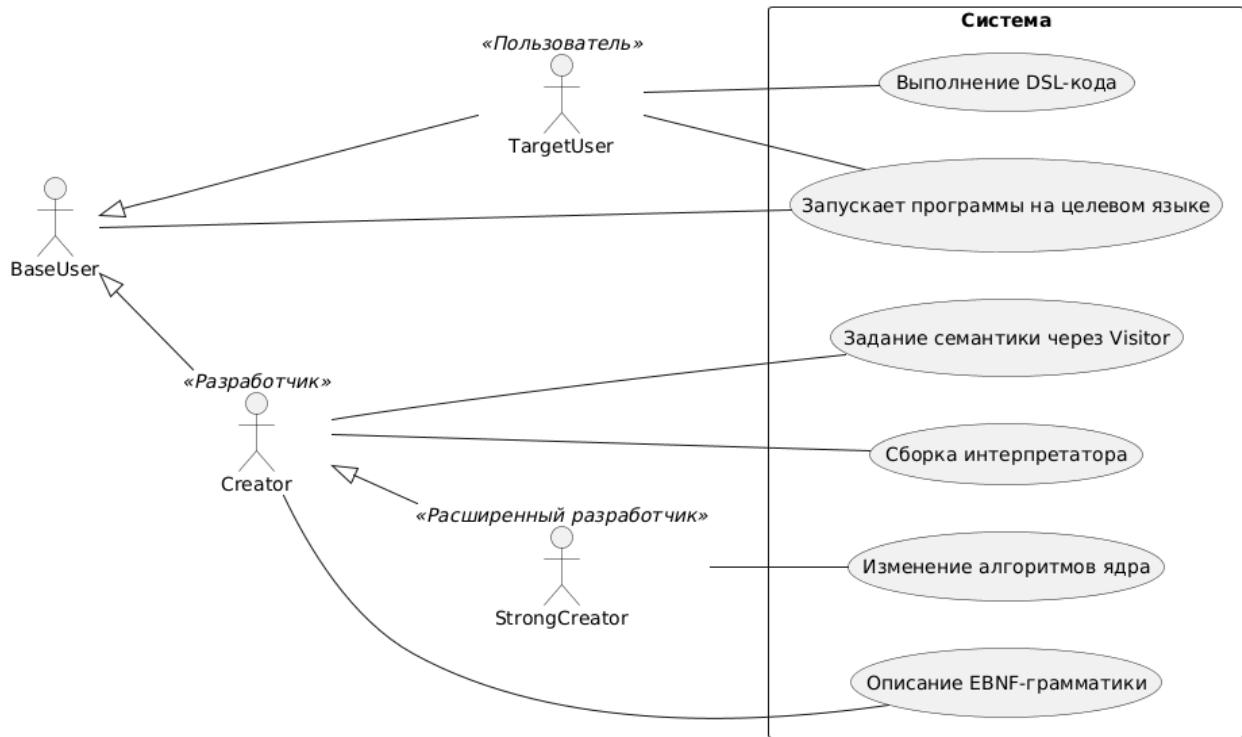


Figure 5

*Комментарий:* Тут конечно стоило написать, что лучше чтобы теге авторов не отображались как их имена, а имена отображались снизу, однако это связано с тем, как ИИ решил объявить actors. Кажется, что при дополнительном промте, модель может запутаться и я решил не проверять.

## 9.2 Gemini

## Листинг

```
@startuml
left to right direction
skinparam packageStyle rectangle

actor "BaseUser" as BU
actor "TargetUser" as TU
actor "Creator" as C
actor "StrongCreator" as SC

' Иерархия акторов
BU <|-- TU
BU <|-- C
C <|-- SC

rectangle "DSL Development & Execution System" {
    usecase Запуск" программ на целевом языке" as UC1
    usecase Выполнение" DSLкода-" as UC2
    usecase Описание" EBNFграмматики-" as UC3
    usecase Задание" семантики Visitor" as UC4
    usecase Сборка" интерпретатора" as UC5
    usecase Изменение" алгоритмов ядра" as UC6
}

' Связи акторов с функциями
TU -- UC1
TU -- UC2

C -- UC3
C -- UC4
C -- UC5

SC -- UC6
@enduml
```

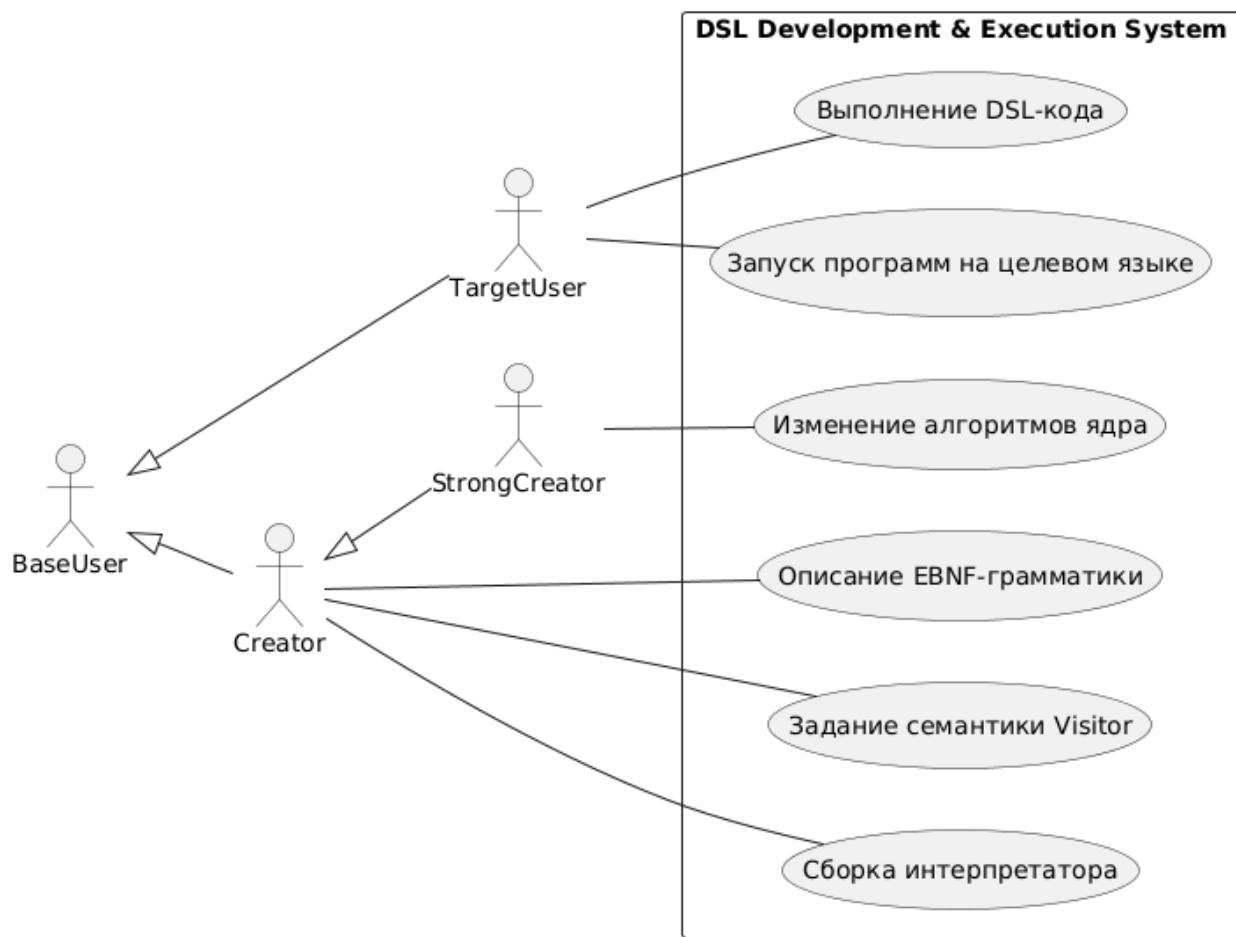


Figure 6

*Комментарий:* Все что нарисовано с точки зрения UML правильно! Но модель не стала додумывать хитросплетений с extend и include. Скорее всего это связано с промтом, поэтому его стоит доработать, либо использовать такую простую картинку.

## 10 Промт 3

### 10.1 DeepSeek

Листинг

```

@startuml
left to right direction

skinparam {
    BackgroundColor #F8F9FA
    ActorBorderColor #2E4A7E
    ActorFontColor #2E4A7E
    UseCaseBorderColor #5A6C7D
    UseCaseFontColor #2E4A7E

```

```

        UseCaseBackgroundColor #E9ECEF
        SystemBorderColor #2E4A7E
    }

    rectangle Система" DSLTools" {
        Запустить( программу на целевом языке)
        Сгенерировать( интерпретатор)
        Задать( грамматику (EBNF))
        Задать( семантику)
        Изменить( ядро интерпретатора)
        Визуализировать( деревья CST/AST)

        Запустить( программу на целевом языке) .>
        Автоподгрузка( интерпретатора) : включает
            Сгенерировать( интерпретатор) .> Интерпретировать(
        описание грамматики) : включает
            Сгенерировать( интерпретатор) .> Интерпретировать(
        описание семантики) : включает
            Задать( семантику) <|-- Задать( семантику вручную)
            Задать( семантику) <|-- Сгенерировать( семантику
        через ИИ)
            Изменить( ядро интерпретатора) .> Модифицировать(
        токенизатор) : включает
            Изменить( ядро интерпретатора) .> Модифицировать( CST
        билдер-) : включает
            Изменить( ядро интерпретатора) .> Модифицировать( AST
        транслятор-) : включает
            Изменить( ядро интерпретатора) .> Расширить( шаги
        интерпретации) : включает
            Визуализировать( деревья CST/AST) ..> Сгенерировать(
        интерпретатор) : расширяет
            Визуализировать( деревья CST/AST) ..> Запустить(
        программу на целевом языке) : расширяет
    }

    actor Пользователь" системы" as SystemUser
    actor Пользователь" DSL" as DSLUser
    actor Разработчик" DSL" as DSLDev
    actor Сильный" разработчик" as AdvancedDev

    DSLUser <|-- SystemUser
    DSLUser <|-- DSLDev

```

```

DSLDev <|-- AdvancedDev

SystemUser --> Запустить( программу на целевом языке)

DSLUser --> Визуализировать( дерева CST/AST)

DSLDev --> Задать( грамматику (EBNF))
DSLDev --> Задать( семантику)
DSLDev --> Сгенерировать( интерпретатор)

AdvancedDev --> Изменить( ядро интерпретатора)

note right of Задать( семантику)
Два варианта реализации:
1. Ручной ввод
2. Генерация-
end note

note bottom of Визуализировать( дерева CST/AST)
Опциональное расширение для процессов
сборки интерпретатора и выполнения программ
end note

@enduml

```

**Код не компилируется!**

**Результат после первого исправления**

Текст исправления:

Листинг

```
Код не компилируется. Ошибка в 17 строчке пишет после Задать
грамматику EBNF assumed diagram type: component
```

Листинг

```
@startuml
left to right direction
```

```

skinparam {
    BackgroundColor #F8F9FA
    ActorBorderColor #2E4A7E
    ActorFontColor #2E4A7E
    UseCaseBorderColor #5A6C7D
    UseCaseFontColor #2E4A7E
    UseCaseBackgroundColor #E9ECEF
    SystemBorderColor #2E4A7E
}

rectangle Система" DSLTools" {
    usecase Запустить" программу на целевом языке" as UC1
    usecase Сгенерировать" интерпретатор" as UC2
    usecase Задать" грамматику (EBNF)" as UC3
    usecase Задать" семантику" as UC4
    usecase Изменить" ядро интерпретатора" as UC5
    usecase Визуализировать" деревья CST/AST" as UC6

    usecase Автоподгрузка" интерпретатора" as UC7
    usecase Интерпретировать" описание грамматики" as UC8
    usecase Интерпретировать" описание семантики" as UC9
    usecase Задать" семантику вручную" as UC10
    usecase Сгенерировать" семантику через ИИ" as UC11
    usecase Модифицировать" токенизатор" as UC12
    usecase Модифицировать" CSTбилдер-" as UC13
    usecase Модифицировать" ASTтранслятор-" as UC14
    usecase Расширить" шаги интерпретации" as UC15

    UC1 .> UC7 : включает
    UC2 .> UC8 : включает
    UC2 .> UC9 : включает
    UC4 <|-- UC10
    UC4 <|-- UC11
    UC5 .> UC12 : включает
    UC5 .> UC13 : включает
    UC5 .> UC14 : включает
    UC5 .> UC15 : включает
    UC6 ..> UC2 : расширяет
    UC6 ..> UC1 : расширяет
}

actor Пользователь" системы" as SystemUser

```

```
actor Пользователь" DSL" as DSLUser
actor Разработчик" DSL" as DSLDev
actor Сильный" разработчик" as AdvancedDev
```

```
DSLUser <|-- SystemUser
DSLUser <|-- DSLDev
DSLDev <|-- AdvancedDev
```

```
SystemUser --> UC1
DSLUser --> UC6
DSLDev --> UC3
DSLDev --> UC4
DSLDev --> UC2
AdvancedDev --> UC5
```

```
note right of UC4
Два варианта реализации:
1. Ручной ввод
2. Генерация-
end note
```

```
note bottom of UC6
Опциональное расширение для процессов
сборки интерпретатора и выполнения программ
end note
@enduml
```

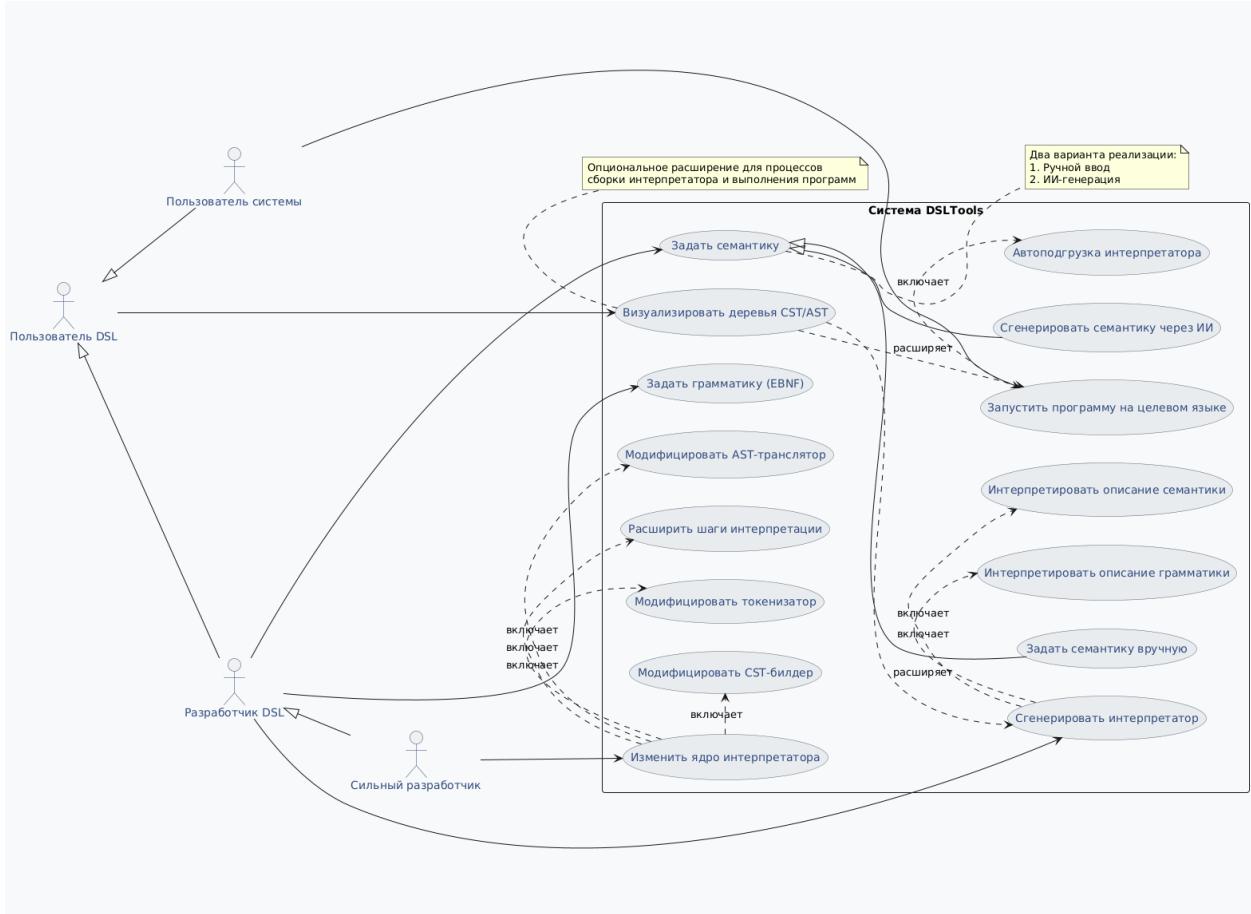


Figure 7

## После второго исправления

### Листинг

Список неправильных вещей:

Ассоциация должна не иметь стрелки на конце  
у extend согласно стандарту должно быть разделение в  
овальчике на с перечислением extend points

Далее сами подписи на стрелках должны быть на английском  
включает и расширяет соответственно должны быть написаны include и exclude

Неправильно определил связи для usecase изменить ядро -  
должна быть связь generalization

Стрелка extend согласно стандарту должна вести к  
расширяемому объекту

Также комментарий следует оформить согласно стандарту UML то  
есть для расширяющих понятий ввести condition на расширение например  
рисование AST/CST - можно сделать on\_debug в случае с расширением  
сгенерировать интерпретатор

Кроме того, ассоциация с визуализацией убрать

## Листинг

```
@startuml
left to right direction

skinparam {
    BackgroundColor #F8F9FA
    ActorBorderColor #2E4A7E
    ActorFontColor #2E4A7E
    UseCaseBorderColor #5A6C7D
    UseCaseFontColor #2E4A7E
    UseCaseBackgroundColor #E9ECEF
    SystemBorderColor #2E4A7E
}

rectangle Система" DSLTools" {
    usecase Запустить" программу на целевом языке" as UC1
    usecase Сгенерировать" интерпретатор" as UC2
    usecase Задать" грамматику (EBNF)" as UC3
    usecase Задать" семантику" as UC4
    usecase Изменить" ядро интерпретатора" as UC5
    usecase Визуализировать" деревья CST/AST" as UC6

    usecase Автоподгрузка" интерпретатора" as UC7
    usecase Интерпретировать" описание грамматики" as UC8
    usecase Интерпретировать" описание семантики" as UC9
    usecase Задать" семантику вручную" as UC10
    usecase Сгенерировать" семантику через ИИ" as UC11
    usecase Модифицировать" токенизатор" as UC12
    usecase Модифицировать" CSTбилдер-" as UC13
    usecase Модифицировать" ASTтранслятор-" as UC14
    usecase Расширить" шаги интерпретации" as UC15

    UC1 .> UC7 : include
    UC2 .> UC8 : include
    UC2 .> UC9 : include

    UC4 <|-- UC10
    UC4 <|-- UC11

    UC5 <|-- UC12
    UC5 <|-- UC13
```

```

UC5 <|-- UC14
UC5 <|-- UC15

UC6 ..> UC2 : extend
UC6 ..> UC1 : extend
}

actor Пользователь" системы" as SystemUser
actor Пользователь" DSL" as DSLUser
actor Разработчик" DSL" as DSLDev
actor Сильный" разработчик" as AdvancedDev

DSLUser <|-- SystemUser
DSLUser <|-- DSLDev
DSLDev <|-- AdvancedDev

SystemUser -- UC1
DSLUser -- UC6
DSLDev -- UC3
DSLDev -- UC4
DSLDev -- UC2
AdvancedDev -- UC5

note right of UC6
<<extend>>
Condition: on_visualization_request
end note

note right of UC4
Варианты реализации:
1. Ручной ввод
2. ИИгенерация-
end note
@enduml

```

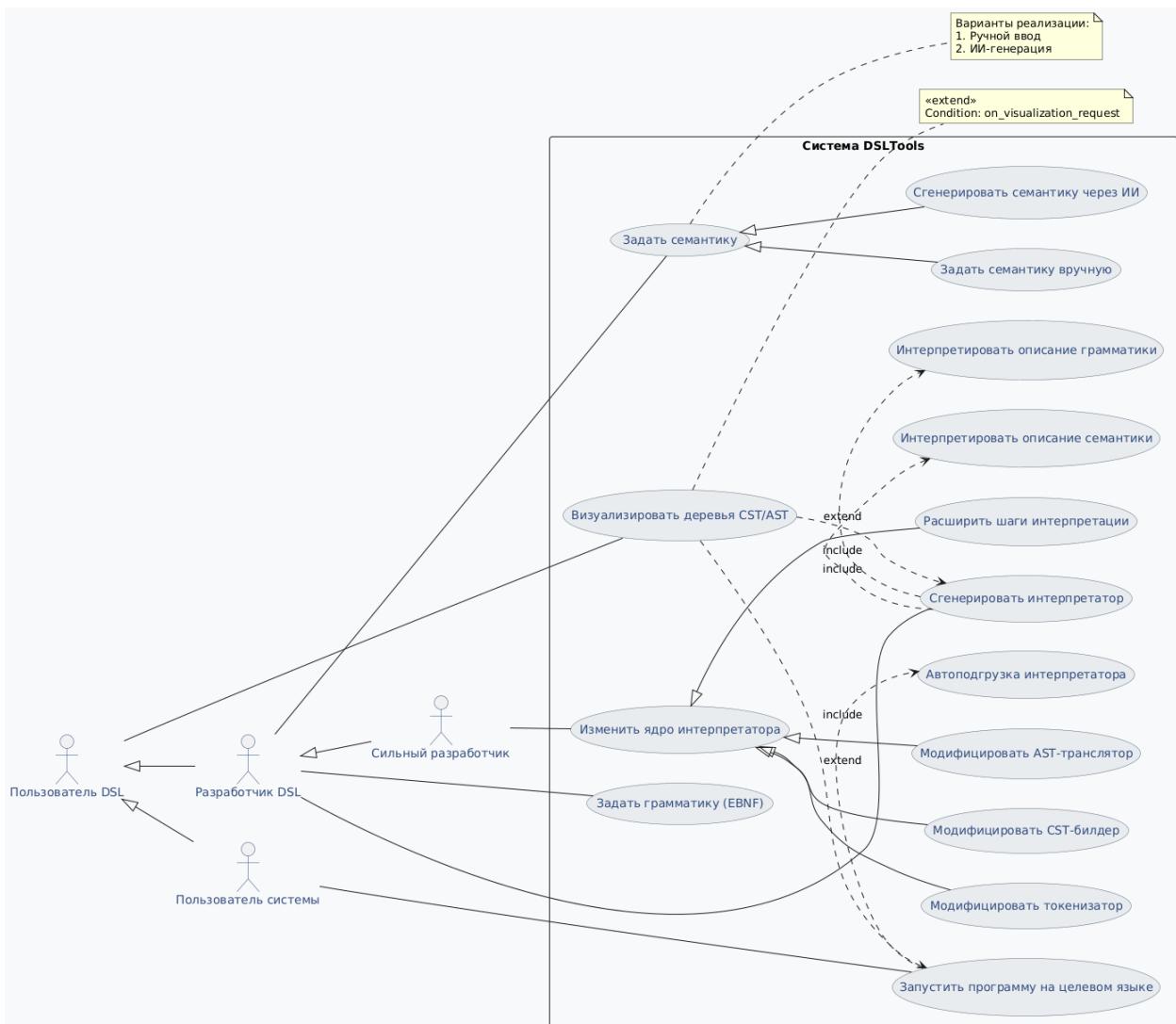


Figure 8

*Комментарий:* Модель почти повторила то, что от нее требовалось, но к сожалению есть лишняя стрелка Визуализация CST. Кроме того, верстка оставляет желать лучшего, в плане вершили из стрелочек. Хотя получилось стрелки поправить.

## 10.2 Gemini

### Результат начального промтта

Листинг

```

@startuml
skinparam actorStyle hollow
left to right direction

actor Пользователь" DSL" as DSLUser
actor Пользователь" системы" as SystemUser
    
```

```

actor Разработчик" DSL" as DSLDev
actor Сильный" разработчик" as StrongDev

' Иерархия акторов
DSLUser <|-- SystemUser
DSLUser <|-- DSLDev
DSLDev <|-- StrongDev

rectangle Система" DSLTools" {

    ' Кейсы для Пользователя системы
usecase Запустить" программу" as RunProg
usecase Подгрузить" интерпретатор" as LoadInterp
usecase Визуализировать" деревья (CST/AST)" as
VizTrees

    ' Кейсы для Разработчика DSL
usecase Задать" грамматику (EBNF)" as SetGrammar
usecase Задать" семантику (Visitor)" as SetSemantics
usecase Ручной" ввод семантики" as ManualSem
usecase ИИгенерация"- семантики" as AISem
usecase Сгенерировать" интерпретатор" as GenInterp

    ' Внутренние процессы генерации
usecase Проинтерпретировать" описание грамматики" as
InterpGrammar
usecase Проинтерпретировать" правила семантики" as
InterpSemantics

    ' Кейсы для Сильного разработчика
usecase Изменить" алгоритмы ядра" as ModifyCore
usecase Изменить" токенизатор" as ModToken
usecase Изменить" CSTбилдер-" as ModCST
usecase Изменить" ASTтранслятор-" as ModAST
usecase Расширить" шаги интерпретации" as ExtendSteps

    ' Отношения для Пользователя системы
SystemUser --> RunProg
RunProg ..> LoadInterp : <<include>>
VizTrees ..> RunProg : <<extend>>

    ' Отношения для Разработчика DSL
}

```

```

DSLDev --> SetGrammar
DSLDev --> SetSemantics
DSLDev --> GenInterp

SetSemantics <|-- ManualSem
SetSemantics <|-- AISem

GenInterp ..> InterpGrammar : <<include>>
GenInterp ..> InterpSemantics : <<include>>
VizTrees ..> GenInterp : <<extend>>

' Отношения для Сильного разработчика
StrongDev --> ModifyCore
ModifyCore <|-- ModToken
ModifyCore <|-- ModCST
ModifyCore <|-- ModAST
ModifyCore <|-- ExtendSteps
}

note right of LoadInterp
Автоматическая подгрузка
при запуске
end note

@enduml

```

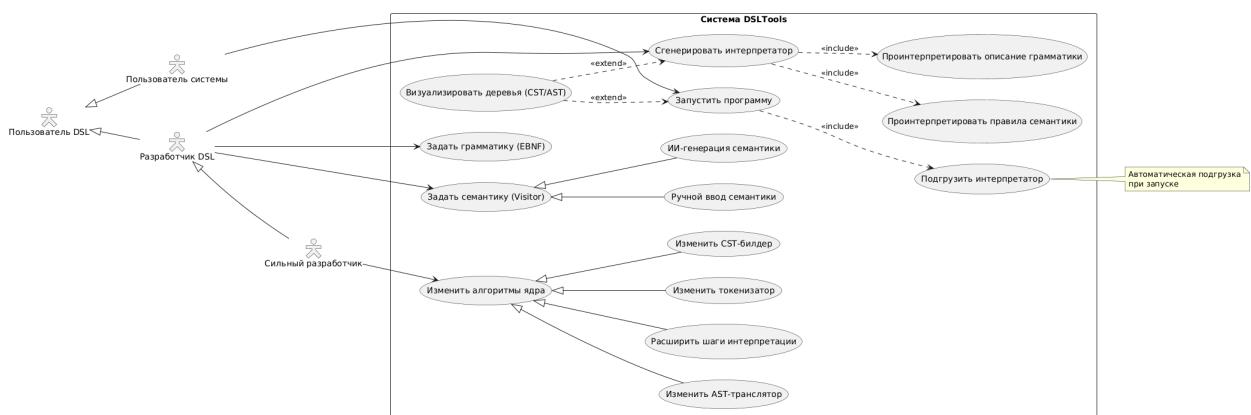


Figure 9

## Сделал корректировки

Текст исправления:

## Листинг

Неправильно нарисовал акторов – человечки должны быть из UML + к тому неправильно нарисовал стрелку ассоциации + у extend согласно стандарту должно быть разделение в овальчике на с перечислением extend points

## Листинг

```
@startuml
left to right direction

' Акторыстандартные ( UML человечки)
actor Пользователь" DSL" as DSLUser
actor Пользователь" системы" as SystemUser
actor Разработчик" DSL" as DSLDev
actor Сильный" разработчик" as StrongDev

' Иерархияакторов
DSLUser <|-- SystemUser
DSLUser <|-- DSLDev
DSLDev <|-- StrongDev

rectangle Система" DSLTools" {

    ' === КейсыПользователясистемы ===

    ' Определение Use Case сточкойрасширения
usecase RunProg as Запустить" программу
-- 
**Extension Points:**
    Точкавизуализации выполнения
    ""

usecase Подгрузить" интерпретатор" as
LoadInterp

' Общийкейсрасширения
usecase Визуализировать" деревья (CST/AST)"
as VizTrees
```

```

' === КейсыРазработчика DSL ===
usecase Задать" грамматику (EBNF)" as
    SetGrammar
usecase Задать" семантику (Visitor)" as
    SetSemantics

' Вариантызаданиясемантики (Generalization)
)
usecase Ручной" вводсемантики " as
    ManualSem
usecase ИИгенерация"- семантики" as AISem

' Определение Use Case сточкойрасширения
usecase GenInterp as Сгенерировать"
    интерпретатор
-- 
**Extension Points:**Точкивизуализации сборки
"

' Внутренниепроцессыгенерации ((
    связанысакторами напрямую )
usecase Проинтерпретировать"
    описаниеграмматики " as InterpGrammar
usecase Проинтерпретировать" правиласемантиki
    " as InterpSemantics

' === КейсыСильногоразработчика ===
usecase Изменить" алгоритмиядра " as
    ModifyCore

' Вариантыизмененияядра (Generalization)
usecase Изменить" токенизатор" as ModToken
usecase Изменить" CSTбилдер- " as ModCST
usecase Изменить" ASTтранслятор- " as ModAST
usecase Расширить" шагиинтерпретации " as
    ExtendSteps

' === Отношения ===
,
Ассоциациикоторовсосновными кейсами стандартные
( линиибезстрелок )

```

```

SystemUser -- RunProg
DSLDev -- SetGrammar
DSLDev -- SetSemantics
DSLDev -- GenInterp
StrongDev -- ModifyCore

' Отношения Include
RunProg ..> LoadInterp : <<include>>
GenInterp ..> InterpGrammar : <<include
>>
GenInterp ..> InterpSemantics : <<
include>>

' Отношения Extend с указанием точек расширения
VizTrees .> RunProg : <<extend>>\nТочка(
визуализации выполнения )
VizTrees .> GenInterp : <<extend>>\n
Точка( визуализации сборки )

' Отношения Generalization для Use Cases
SetSemantics <|-- ManualSem
SetSemantics <|-- AISem

ModifyCore <|-- ModToken
ModifyCore <|-- ModCST
ModifyCore <|-- ModAST
ModifyCore <|-- ExtendSteps
}

note bottom of LoadInterp
Обязательная автоматическая подгрузка при запуске

end note
@enduml

```

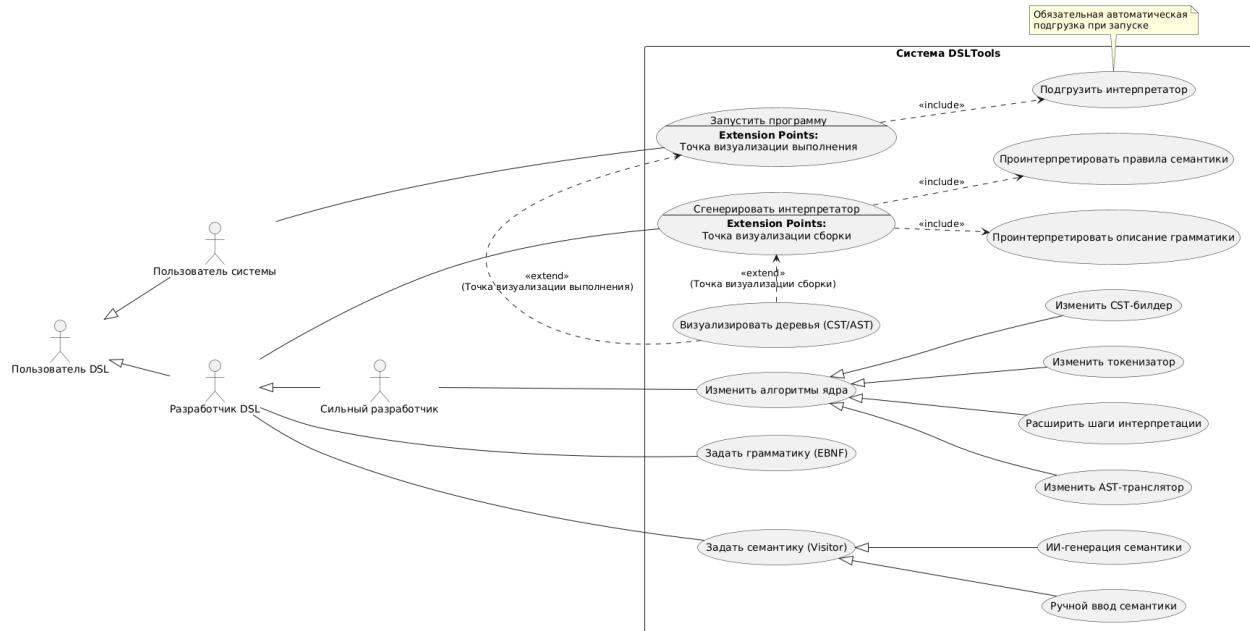


Figure 10

**Комментарий:** На диаграмме присутствую ошибки вроде подписи чего-то похожего на condition рядом с extend. Однако в целом картинка почти красиво отверстана и все стрелки угаданы верно. Ну и конечно какой-то лишний комментарий: "обязательная подгрузка при запуске".

# 11 Промт 4

## 11.1 DeepSeek

### Аналитическая сводка проектирования архитектуры

#### 1. Цепочка мыслей (Reasoning Chain)

Этап анализа	Логическое обоснование и принятые архитектурные решения
<b>Роли</b>	Иерархия акторов разделяет ответственность по принципу специализации. <i>BaseUser</i> является абстрактной ролью, пользователем системы. <i>Creator</i> — конкретный пользователь, создающий интерпретатор. <i>StrongCreator</i> — специализированный <i>Creator</i> с расширенными правами (например, доступ к модификации ядра). <i>TargetUser</i> — конечный потребитель, исполняющий программы, но не участвующий в создании DSL. Это обеспечивает четкое разделение интерфейсов для разных категорий пользователей.
<b>Инкапсуляция</b>	Ключевой принцип — скрыть детали системных процессов от акторов. Акторы инициируют только высокоуровневые цели ( <i>Сгенерировать интерпретатор</i> , <i>Выполнить программу</i> ). Внутренние шаги ( <i>Проинтерпретировать грамматику</i> , <i>Подгрузить интерпретатор</i> ) инкапсулированы внутри этих целей через отношения <i>include</i> . Это защищает диаграмму от перегруженности и отражает архитектурный контракт: актору доступен только публичный API системы.
<b>Связи</b>	Для алгоритмических компонентов ( <i>Модифицировать модуль ядра</i> ) применено <b>Generalization</b> , что точно отражает "is-a" отношение: каждый конкретный алгоритм является вариацией модификации ядра. Для опциональной функциональности ( <i>Визуализировать дерево</i> ) использовано <b>extend</b> с указанием точек расширения и условий. Это четко отделяет основную функциональность от вспомогательной, зависящей от контекста и внешних условий.

<b>Визуализация</b>	Логика направлена на максимальную читаемость. Направление связей строго регламентировано: от акторов к кейсам (инициирование) и между кейсами согласно отношениям зависимостей ( <code>include</code> , <code>extend</code> ). Эшелонирование и скрытые связи предотвращают пересечение линий и формируют логические горизонтали, соответствующие уровню абстракции. Это позволяет одновременно воспринимать и структуру системы, и поток выполнения.
---------------------	---

## 2. Принятые решения по верстке

- **Эшелонирование:** Распределение кейсов по 5 уровням для исключения пересечений. Каждый эшелон соответствует этапу в цепочке создания/исполнения: **1. Инициация** (основные цели), **2. Системные процессы** (логика системы), **3. Определения** (конфигурация), **4. Реализации** (способы исполнения), **5. Ядро** (базовые операции). Это создает вертикальную "карту" функциональности.
- **Позиционирование:** Принудительное закрепление акторов по осям (Верх, Лево, Низ) основано на их роли в потоке данных. `BaseUser` (вверху) как источник всех взаимодействий. `Creator` и `StrongCreator` (слева) как активные конфигураторы системы. `TargetUser` (внизу) как пассивный исполнитель, потребляющий результат работы `Creator`. Такое расположение интуитивно понятно и соответствует основным сценариям использования.

### Листинг

```
@startuml DSLTools Use Case Diagram
' =====
' Настройки направления и оформления
' =====
top to bottom direction
skinparam actorStyle awesome
skinparam usecase {
    BackgroundColor #F9F7F5
    BorderColor #2E4053
    ArrowColor #2E4053
}
skinparam note {
    BackgroundColor #FFFDE7
    BorderColor #FFC107
}
title Use Case Diagram: Система DSLTools
```

```

    , =====
    ' 1. ОПРЕДЕЛЕНИЕ АКТОРОВ И ИХ ПОЗИЦИОНИРОВАНИЕ
    , =====
    actor BaseUser as BU <<abstract>> #gold
    actor Creator as CR <<extends BU>> #lightblue
    actor StrongCreator as SCR <<extends CR>> #orange
    actor TargetUser as TU <<extends BU>> #lightgreen

    ' Расположение акторов строго по требованиям: Верх, Лево, Низ
    BU - [hidden]up- 0
    CR - [hidden]left- 0
    SCR - [hidden]left- 0
    TU - [hidden]down- 0

    , =====
    ' 2. ЭШЕЛОНIRОВАНИЕ: ЭШЕЛОН 1 - ВХОДЫ ИНИЦИАЦИЯ()
    , =====
    ' Основные цели, доступные акторам.
    use case Сгенерировать" интерпретатор" as UC_GEN #palegreen
    use case Выполнить" программу на DSL" as UC_EXEC #palegreen

    ' Связи от акторов к верхнеуровневым кейсам правило(
верхушки")
    BU --> UC_GEN
    BU --> UC_EXEC
    CR --> UC_GEN
    SCR --> UC_GEN
    TU --> UC_EXEC

    , =====
    ' 3. ЭШЕЛОНIRОВАНИЕ: ЭШЕЛОН 2 - СИСТЕМНЫЕ ПРОЦЕССЫ
    , =====
    use case Проинтерпретировать" грамматику" as UC_PARSE_G #
moccasin
    use case Проинтерпретировать" семантику" as UC_PARSE_S #
moccasin
    use case Подгрузить" интерпретатор" as UC_LOAD #moccasin

    ' Инкапсуляция внутренних процессов через include
    UC_GEN ..> UC_PARSE_G : <<include>>
    UC_GEN ..> UC_PARSE_S : <<include>>

```

```

UC_EXEC ..> UC_LOAD : <<include>>

    ' Невидимые связи для выравнивания эшелона 2 под эшелоном 1
UC_PARSE_G - [hidden]down- UC_PARSE_S
UC_PARSE_S - [hidden]down- UC_LOAD

    ' =====
    ' 4. ЭШЕЛОННИРОВАНИЕ: ЭШЕЛОН 3 - ОПРЕДЕЛЕНИЯ
    ' =====
use case Задать" синтаксис + лексику" as UC_DEF_SYN #
lightskyblue
use case Задать" семантику (Visitor)" as UC_DEF_SEM <<
abstract>> #lightskyblue

    ' Связи: системные процессы требуют определений
UC_PARSE_G ..> UC_DEF_SYN : <<include>>
UC_PARSE_S ..> UC_DEF_SEM : <<include>>

    ' Выравнивание эшелона 3 под эшелоном 2
UC_DEF_SYN - [hidden]down- UC_DEF_SEM

    ' =====
    ' 5. ЭШЕЛОН 4 - РЕАЛИЗАЦИИ СЕМАНТИКИ
    ' =====
use case Ручная" регистрация" as UC_MANUAL #thistle
use case Автогенерация" через ИИ" as UC_AI #thistle

    ' Обобщение (Generalization) для способов определения
семантики
UC_DEF_SEM <|-- UC_MANUAL
UC_DEF_SEM <|-- UC_AI

    ' Выравнивание эшелона 4 под эшелоном 3
UC_MANUAL - [hidden]down- UC_AI

    ' =====
    ' 6. ЭШЕЛОН 5 - ЯДРО БАЗОВЫЕ( МОДИФИКАЦИИ)
    ' =====
use case Модифицировать" модуль ядра" as UC_CORE <<abstract>>
#salmon

    ' Связь: создание семантики через( AI) может потребовать

```

модификации ядра.

’ Ручная регистрация не требует модификации ядра от системы.

UC\_AI ..> UC\_CORE : <<extend>>\nдля( оптимизации)

’ =====

’ 7. СПЕЦИАЛИЗАЦИЯ АЛГОРИТМОВ ЯДРА (GENERALIZATION)

’ =====

use case Алгоритм" токенизатора" as UC\_TOK #wheat

use case "CSTбилдер-" as UC\_CST #wheat

use case "ASTтранслятор-" as UC\_AST #wheat

use case Шаги" интерпретации" as UC\_STEP #wheat

’ Иерархия модификаций ядра

UC\_CORE <|-- UC\_TOK

UC\_CORE <|-- UC\_CST

UC\_CORE <|-- UC\_AST

UC\_CORE <|-- UC\_STEP

’ Выравнивание специализаций ядра

UC\_TOK -[hidden]right- UC\_CST

UC\_CST -[hidden]right- UC\_AST

UC\_AST -[hidden]right- UC\_STEP

’ =====

’ 8. ОПЦИОНАЛЬНАЯ ФУНКЦИОНАЛЬНОСТЬ (EXTEND)

’ =====

use case Визуализировать" дерево" as UC\_VIS #lightcoral

’ Точки расширения определены в основных сценариях

UC\_GEN .> UC\_VIS : <<extend>> (point: after generation)\n<  
color:#2E4053>(if debug flag is ON)</color>

UC\_EXEC .> UC\_VIS : <<extend>> (point: after execution)\n<  
color:#2E4053>(if debug flag is ON)</color>

note on link

\*\*Condition\*\*: Системная( настройка 'debug' == TRUE)

AND У( пользователя есть роль 'DebugViewer')

end note

’ =====

’ 9. СПЕЦИАЛЬНЫЙ ДОСТУП StrongCreator К ЯДРУ

’ =====

```

' StrongCreator имеет прямой доступ к базовым операциям ядра
SCR --> UC_CORE

',
=====
' ЛЕГЕНДА ДЛЯ ПОЯСНЕНИЯ ОПЦИОНАЛЬНО()
=====

legend right
| Цветовая схема эшелонов |
<color:palegreenЭшелон> 1: Основные цели</color> |
<color:moccasinЭшелон> 2: Системные процессы</color> |
<color:lightskyblueЭшелон> 3: Определения</color> |
<color:thistleЭшелон> 4: Реализации</color> |
<color:salmonЭшелон> 5: Ядро</color> |
---
Стрелки:
--> Ассоциация актора
..> <<include>> / <<extend>>
<!-- Наследование (Generalization)
endlegend

@enduml

```

## Не работает! Не компилируется!

### После исправления

#### Листинг

Не компилируется  
Пишет ошибку:  
Syntax error? (Assumed diagram type: component) на строчке

36

#### Листинг

```

@startuml DSLTools Use Case Diagram
',
=====
' Настройки направления оформления
',
=====
top to bottom direction

```

```

skinparam actorStyle awesome
skinparam usecase {
    backgroundColor #F9F7F5
    BorderColor #2E4053
    ArrowColor #2E4053
}
skinparam note {
    backgroundColor #FFFDE7
    BorderColor #FFC107
}
title Use Case Diagram: Система DSLTools

' =====
' 1. ОПРЕДЕЛЕНИЕ АКТОРОВ ИХ ПОЗИЦИОНИРОВАНИЕ
' =====

actor BaseUser as BU <<abstract>> #gold
actor Creator as CR <<extends BU>> #lightblue
actor StrongCreator as SCR <<extends CR>> #
    orange
actor TargetUser as TU <<extends BU>> #
    lightgreen

' Расположение акторов строго по требованиям      : Верх,
'             Лево, Низ
BU -[hidden]up-> 0
CR -[hidden]left-> 0
SCR -[hidden]left-> 0
TU -[hidden]down-> 0

' =====
' 2. ЭШЕЛОННОЕ ПОЛОЖЕНИЕ: ЭШЕЛОН 1 - ВХОДЫ ИНИЦИАЦИЯ ()
' =====

' Основные цели , доступные акторам .
usecase Сгенерировать" интерпретатор" as UC_GEN #
    palegreen
usecase Выполнить" программу на DSL" as UC_EXEC #
    palegreen

' Связи от акторов к верхнему уровню включая сам правила
' верхушки ""
BU --> UC_GEN
BU --> UC_EXEC

```

```

CR --> UC_GEN
SCR --> UC_GEN
TU --> UC_EXEC

' =====
' 3. ЭШЕЛОННОЕ ПРОЦЕССИРОВАНИЕ: ЭШЕЛОН 2 - СИСТЕМНЫЕ ПРОЦЕССЫ
' =====

usecase Проинтерпретировать" грамматику" as
    UC_PARSE_G #moccasin

usecase Проинтерпретировать" семантику" as UC_PARSE_S
    #moccasin

usecase Подгрузить" интерпретатор" as UC_LOAD #
    moccasin

' Инкапсуляция внутренних процессов через      include
UC_GEN ..> UC_PARSE_G : <<include>>
UC_GEN ..> UC_PARSE_S : <<include>>
UC_EXEC ..> UC_LOAD : <<include>>

' Невидимые связи для выравнивания эшелона      2
под эшелоном 1
UC_PARSE_G -[hidden]down-> UC_PARSE_S
UC_PARSE_S -[hidden]down-> UC_LOAD

' =====
' 4. ЭШЕЛОННОЕ ПРОЦЕССИРОВАНИЕ: ЭШЕЛОН 3 - ОПРЕДЕЛЕНИЯ
' =====

usecase Задать" синтаксис + лексику" as UC_DEF_SYN #
    lightskyblue
usecase Задать" семантику (Visitor)" as UC_DEF_SEM
    <<abstract>> #lightskyblue

' Связи: системные процессы требуют определений
UC_PARSE_G ..> UC_DEF_SYN : <<include>>
UC_PARSE_S ..> UC_DEF_SEM : <<include>>

' Выравнивание эшелона 3 под эшелоном 2
UC_DEF_SYN -[hidden]down-> UC_DEF_SEM

' =====
' 5. ЭШЕЛОН 4 - РЕАЛИЗАЦИЯ И СЕМАНТИКА
' =====

```

```

usecase Ручная" регистрация" as UC_MANUAL #thistle
usecase Автогенерация" черезИИ " as UC_AI #thistle

' Обобщение (Generalization)
дляспособовопределениясемантики
UC_DEF_SEM <|-- UC_MANUAL
UC_DEF_SEM <|-- UC_AI

' Выравниваниеэшелона 4 подэшелоном 3
UC_MANUAL -[hidden] down-> UC_AI

', =====
', 6. ЭШЕЛОН 5 - ЯДРОБАЗОВЫЕ ( МОДИФИКАЦИИ)
', =====
usecase Модифицировать" модульядра " as UC_CORE <<
abstract>> #salmon

' Связь: созданиесемантикичерез ( AI)
можетпотребоватьмодификацийдрапо
',
Ручнаярегистрациянетребуетмодификацийдрапо
системы

UC_AI ..> UC_CORE : <<extend>>\nдля( оптимизации)

', =====
', 7. СПЕЦИАЛИЗАЦИЯАЛГОРИТМОВЯДРА (GENERALIZATION)
', =====
usecase Алгоритм" токенизатора" as UC_TOK #wheat
usecase "CST билдер-" as UC_CST #wheat
usecase "AST транслятор-" as UC_AST #wheat
usecase Шаги" интерпретации" as UC_STEP #wheat

' Иерархиямодификацийдрапо
UC_CORE <|-- UC_TOK
UC_CORE <|-- UC_CST
UC_CORE <|-- UC_AST
UC_CORE <|-- UC_STEP

' Выравниваниеспециализацийдрапо
UC_TOK -[hidden] right-> UC_CST
UC_CST -[hidden] right-> UC_AST
UC_AST -[hidden] right-> UC_STEP

```

```

', =====
', 8. ОПЦИОНАЛЬНАЯФУНКЦИОНАЛЬНОСТЬ (EXTEND)
', =====
usecase Визуализировать" дерево" as UC_VIS #
    lightcoral

    ' Точкирасширенияопределенывосновныхсценариях
UC_GEN .> UC_VIS : <<extend>> (point: after
    generation)\n<color:#2E4053>(if debug flag is
    ON)</color>
UC_EXEC .> UC_VIS : <<extend>> (point: after
    execution)\n<color:#2E4053>(if debug flag is
    ON)</color>

note on link
**Condition**: Системная( настройка 'debug' == TRUE
)
AND У( пользователяестьроль      'DebugViewer')
end note

', =====
', 9. СПЕЦИАЛЬНЫЙДОСТУП StrongCreator КЯДРУ
', =====
', StrongCreator имеетпрямойдоступкбазовыемоперациямядра
SCR --> UC_CORE

', =====
', ЛЕГЕНДАДЛЯПОЯСНЕНИЯОПЦИОНАЛЬНО      ()
', =====
legend right
| Цветоваясхемаэшелонов      |
<color:palegreenЭшелон> 1: Основныецели </color> |
<color:moccasinЭшелон> 2: Системныепроцессы </color>
|
<color:lightskyblueЭшелон> 3: Определения</color> |
<color:thistleЭшелон> 4: Реализации</color> |
<color:salmonЭшелон> 5: Ядро</color> |
---Стрелки
:
--> Ассоциацияактора
... > <<include>> / <<extend>>

```

```

<!-- Наследование (Generalization)
end legend

```

```
@enduml
```

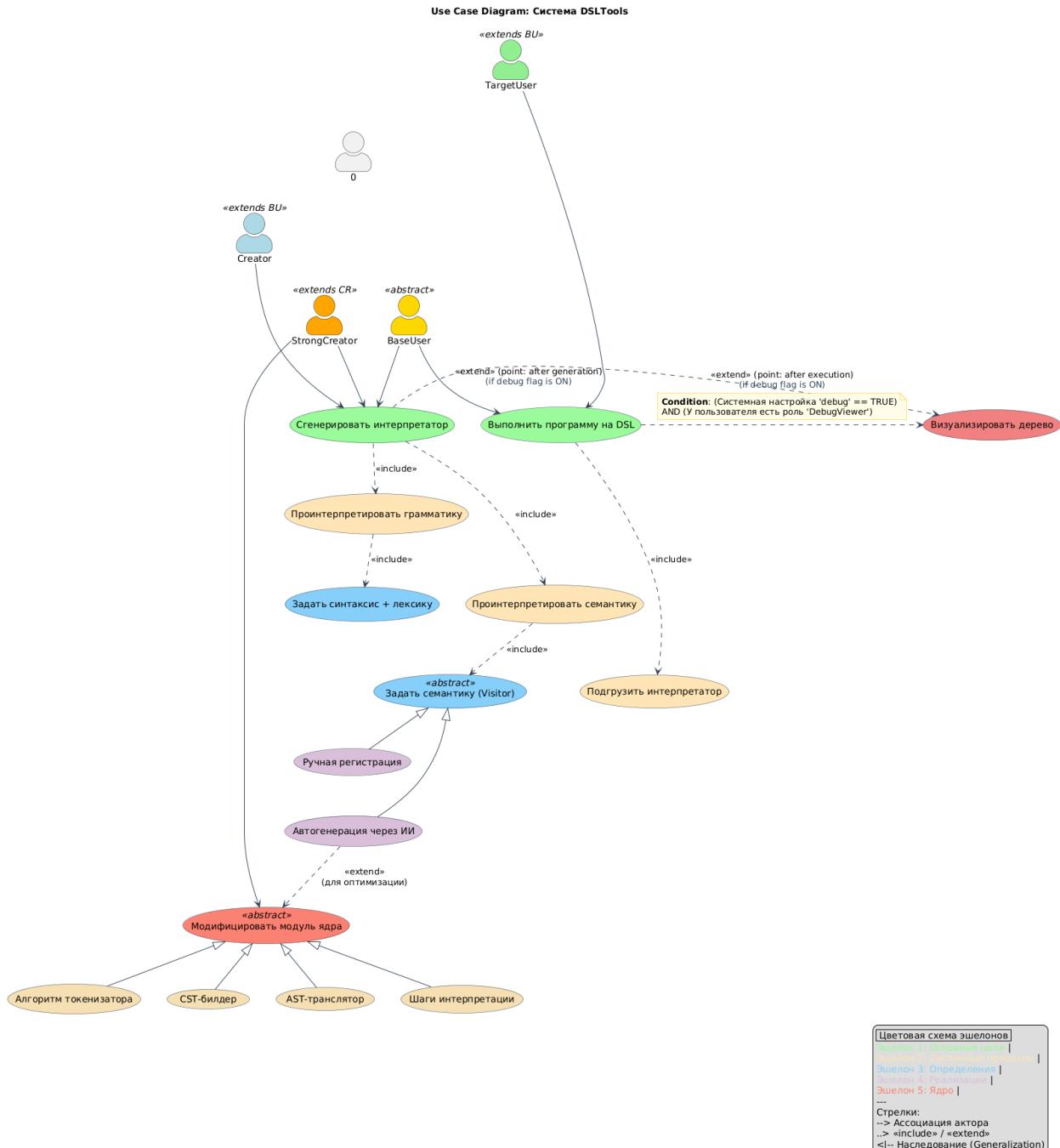


Figure 11

*Комментарий:* Тут произошло определенно что-то не то. Модель выдала цветастую схему с кучей ошибок согласно стандартам UML, начиная от моделей акторов, неправильными стрелками, и заканчивая отсутствием общего прямоугольника для системы. Я решил

не поправлять модель, потому что суммарно потраченное время на запрос + написание запросов в реальном времени уже сильно превышает аналогичный результат для 3 промта DeepSeek и 4-ого промта Gemini.

## 11.2 Gemini

### Цепочка мыслей (Reasoning Chain)

Этап анализа	Логическое обоснование и принятые архитектурные решения
Роли	Иерархия построена на принципе расширения полномочий: <i>BaseUser</i> предоставляет базовый доступ к рантайму, <i>Creator</i> расширяет его инструментами описания DSL, а <i>StrongCreator</i> получает эксклюзивный доступ к низкоуровневой модификации ядра. Это минимизирует дублирование ассоциаций.
Инкапсуляция	Архитектура разделяет «намерения» пользователя и «механику» системы. Процессы интерпретации и загрузки скрыты за отношениями <i>include</i> , так как они являются атомарными частями транзакции запуска/генерации и не инициируются пользователем напрямую.
Связи	Отношение <i>Generalization</i> применено для реализации семантики (Manual/AI), так как они являются альтернативными полиморфными реализациями одного интерфейса. <i>Extend</i> для визуализации вынесен как optionalный аспект с четкой точкой расширения (Extension Point).
Визуализация	Применена стратегия «гравитационного эшелонирования». Использование невидимых связей ( <i>hidden</i> ) гарантирует, что дерево зависимостей не превратится в «спагетти», сохраняя логический поток сверху вниз.

## 2. Принятые решения по верстке

- **Эшелонирование:** Распределение кейсов по 5 уровням (Inputs -> Processes -> Definitions -> Implementations -> Core) для исключения пересечений векторов.
- **Позиционирование:** Принудительное закрепление акторов: *BaseUser* (сверху), *TargetUser* (снизу), *Creator/StrongCreator* (слева) для создания визуального баланса «Вход-Выход-Конфигурация».

## Листинг

```
@startuml
' Настройки стиля для чистоты диаграммы
skinparam usecase {
    BackgroundColor White
    BorderColor Black
    ArrowColor #333333
}
skinparam actor {
    BackgroundColor White
    BorderColor Black
}

title Use Case Diagram - DSLTools System Architecture

top to bottom direction

' === Определение акторов ===
actor "BaseUser" as BU
actor "Creator" as C
actor "StrongCreator" as SC
actor "TargetUser" as TU

' Иерархия акторов
C -up-> BU
SC -up-> C
TU -up-> BU

' === Система ===
rectangle DSLTools {

    ' Эшелон 1: Входы
usecase UC_Gen as "Сгенерировать" интерпретатор
--
extension points:
RenderTree"

usecase UC_Run as "Выполнить" программу на DSL
--
extension points:
RenderTree"
```

```

        ' Эшелон 2: Системные процессы
usecase UC_IntGrammar as Проинтерпретировать"
грамматику"
usecase UC_IntSemantics as Проинтерпретировать"
семантику"
usecase UC_Load as Подгрузить" интерпретатор"

        ' Эшелон 3: Определения
usecase UC_DefSyntax as Задать" синтаксис + лексику"
usecase UC_DefSemantics as Задать" семантику (Visitor
)"

        ' Эшелон 4: Реализации
usecase UC_Manual as Ручная" регистрация"
usecase UC_AI as Автогенерация" через ИИ"

        ' Эшелон 5: Ядро
usecase UC_Core as Модифицировать" модуль ядра"
usecase UC_Tok as Алгоритм" токенизатора"
usecase UC_CST as "CSTбилдер-"
usecase UC_AST as "ASTтранслятор-"
usecase UC_Step as Шаги" интерпретации"

        ' Дополнительный кейс для Extend
usecase UC_Visualize as Визуализировать" дерево"
}

        ' === Отношения между кейсами ===

        ' Include
UC_Gen ..> UC_IntGrammar : <<include>>
UC_Gen ..> UC_IntSemantics : <<include>>
UC_Run ..> UC_Load : <<include>>

        ' Extend
UC_Visualize ..> UC_Gen : <<extend>> \n (condition: if
debug_mode)
UC_Visualize ..> UC_Run : <<extend>> \n (condition: if
trace_enabled)

        ' Наследование (Generalization)

```

```
UC_Manual -up-> UC_DefSemantics  
UC_AI -up-> UC_DefSemantics
```

```
UC_Tok -up-> UC_Core  
UC_CST -up-> UC_Core  
UC_AST -up-> UC_Core  
UC_Step -up-> UC_Core
```

' === Ассоциации с акторами согласно( строгим правилам) ===

```
BU -- UC_Run  
C -- UC_Gen  
C -- UC_DefSyntax  
C -- UC_DefSemantics  
SC -- UC_Core  
TU -- UC_Run
```

' === Выравнивание эшелонов (Hidden links) ===

```
UC_Gen -[hidden]down- UC_IntGrammar  
UC_IntGrammar -[hidden]down- UC_DefSyntax  
UC_DefSyntax -[hidden]down- UC_Manual  
UC_Manual -[hidden]down- UC_Core
```

' Позиционирование акторов

```
BU -[hidden]right- UC_Gen  
TU -[hidden]right- UC_Core
```

```
@enduml
```

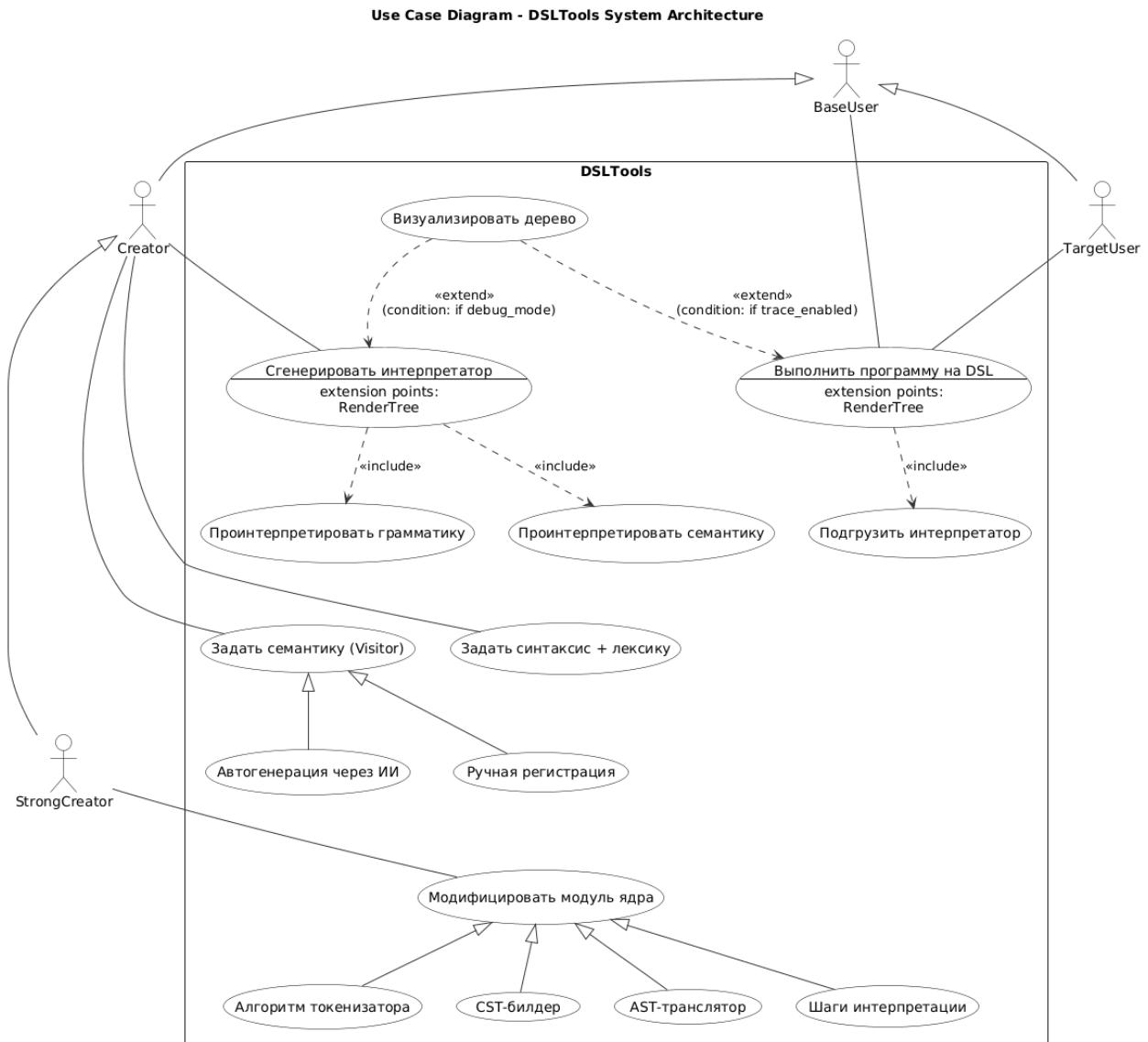


Figure 12

*Комментарий:* Считаю идеальный результат. Плюс он наконец-то мне правильно и красиво все отверстал.

## 12. Анализ результатов исследования

В ходе выполнения лабораторной работы было проведено сравнительное исследование двух моделей ИИ (DeepSeek и Gemini) на предмет их способности генерировать корректные и визуально структурированные диаграммы вариантов использования (Use Case) на языке PlantUML. Оценка проводилась на основе четырех итераций промптов: от базового до архитектурного уровня.

### 12.1 Сводная таблица оценок

Оценка выставлялась по 10-балльной шкале на основе следующих критериев: (а) корректность синтаксиса PlantUML, (б) соответствие стандартам UML, (в) полнота отображения

требований, (d) качество визуализации (верстка), (e) логическая целостность, (f) качество комментариев и самоанализа модели.

Модель / Промпт	(a)	(b)	(c)	(d)	(e)	(f)	Среднее
DeepSeek (П1)	10	8	8	7	8	8	8.2
Gemini (П1)	10	9	8	8	9	8	8.7
DeepSeek (П2)	10	7	9	7	8	8	8.2
Gemini (П2)	10	10	9	9	9	9	9.3
DeepSeek (П3)	7	7	10	6	8	9	7.8
Gemini (П3)	10	9	10	9	10	10	9.7
DeepSeek (П4)	6	6	10	5	7	10	7.3
Gemini (П4)	10	10	10	10	10	10	10.0

Table 3: Сравнительная оценка моделей DeepSeek и Gemini

## 12.2 Обоснование снижения оценок

Снижение баллов обусловлено следующими систематическими ошибками:

- **DeepSeek:** Модель продемонстрировала нестабильность синтаксиса при усложнении задачи. В промптах №3 и №4 код часто не компилировался (ошибки "assumed diagram type: component"). Наблюдались грубые нарушения стандартов UML: использование стрелок в ассоциациях, где они недопустимы, и отсутствие разделения для *extension points* в овалах кейсов. Визуализация превращалась в «вермишель» из пересекающихся линий.
- **Gemini:** Основные замечания касались ранних итераций, где модель упрощала связи *extend* и *include*, не прописывая их без явного указания. В промпте №4 (Senior Architect) модель выдала идеальный результат, полностью соблюдая верстку и семантику UML.

## 12.3 Сравнительная характеристика и эволюция результатов

Эволюция качества диаграмм напрямую коррелирует со сложностью промпта. Если для простейших задач обе модели справляются удовлетворительно, то с ростом иерархии акторов и кейсов DeepSeek начинает «галлюцинировать» синтаксисом.

**Выводы по моделям:**

1. **DeepSeek:** Применим только для создания небольших, примитивных диаграмм. Систематически допускает критические ошибки в коде и визуализации при попытке построения многоуровневых систем. Требует постоянного ручного исправления.
2. **Gemini:** Проявила себя как мощный инструмент проектирования. В отличие от конкурента, Gemini адекватно реагирует на архитектурные инструкции (эшелонирование, инкапсуляция) и выдает чистый, стандартизованный код.

## **Особенности реализации PlantUML:**

- *DeepSeek* часто использует громоздкие конструкции и внешние стили (`skinparam actorStyle awesome`), которые перегружают диаграмму, не улучшая её смысл.
- *Gemini* эффективно использует скрытые связи (`[hidden]`) для управления компоновкой элементов (например, `UC_Gen -[hidden] down- UC_IntGrammar`), что позволяет добиться строгой иерархии эшелонов .

**Анализ цепочек рассуждений (Reasoning Chain):** *Gemini* демонстрирует глубокое понимание архитектурных паттернов. В её цепочке рассуждений прослеживается логика разделения «намерений пользователя» и «механики системы», что отражается в правильном использовании связей `include`. *DeepSeek* также формирует подробную аналитическую сводку, однако не способен транслировать принятые архитектурные решения в работоспособный код PlantUML без синтаксических ошибок.