Абстрактное синтаксическое дерево

Лекция №8

Пример PsKaleidoscope

Пример к лекции:

- https://sourcecraft.dev/sshambir-public/pskaleidoscope

Задача

Сейчас в PsKaleidoscope реализованы:

1. Разбор выражений
2. Разбор программы
3. Константы и переменные

Задача

Сейчас в PsKaleidoscope реализованы:

1. Разбор выражений
2. Разбор программы
3. Константы и переменные

Построчный разбор,
немедленное выполнение

Задача

Сейчас в PsKaleidoscope реализованы:

1. Разбор выражений
2. Разбор программы
3. Константы и переменные

} Построчный разбор, немедленное выполнение

Нужны:

1. Ветвления
2. Циклы
3. Функции

Задача

Сейчас в PsKaleidoscope реализованы:

1. Разбор выражений
2. Разбор программы
3. Константы и переменные

Построчный разбор, немедленное выполнение

Нужны:

1. Ветвления
2. Циклы
3. Функции

Построчное выполнение потребует повторного разбора

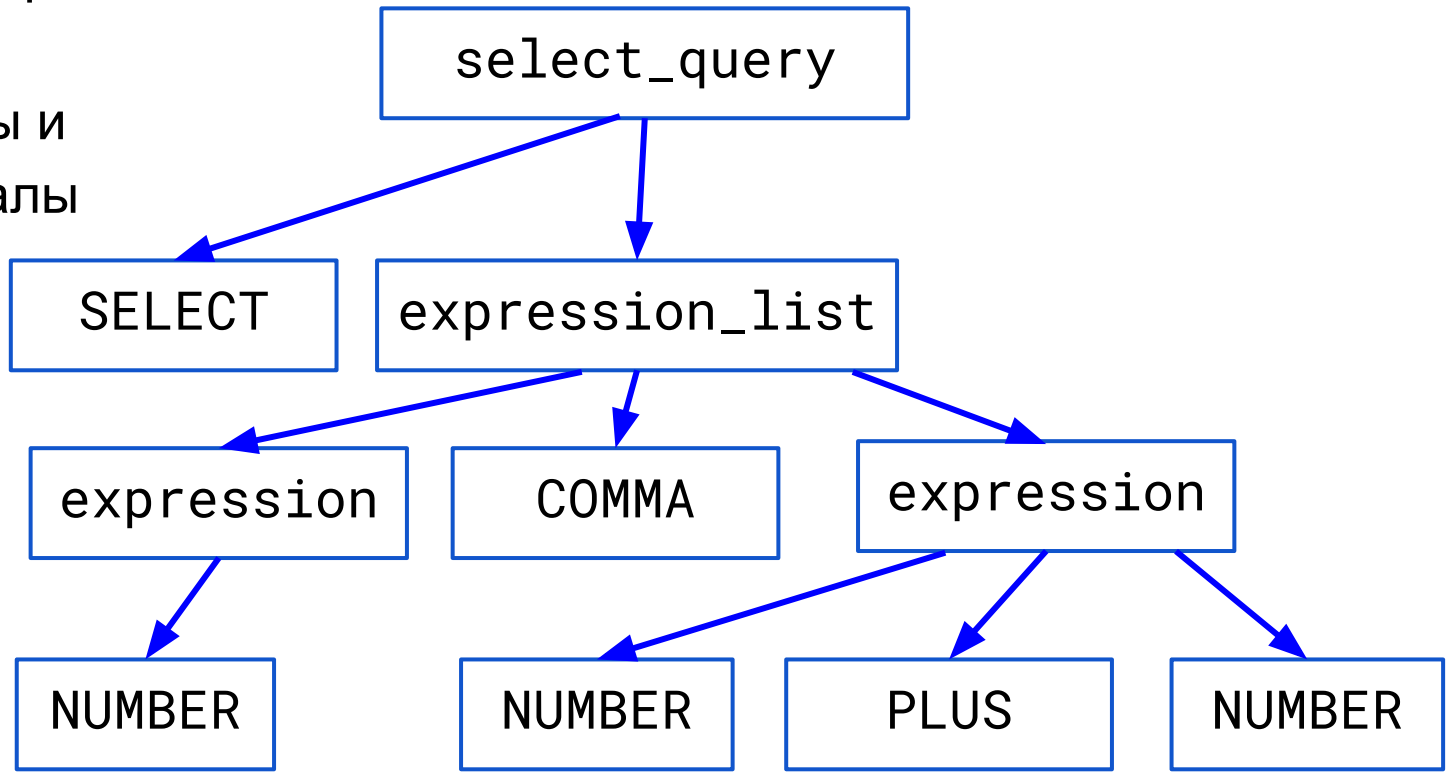
Проблемы / решения

Проблема	Решение
Циклы и функции требуют повторного выполнения	???

Abstract Syntax Tree

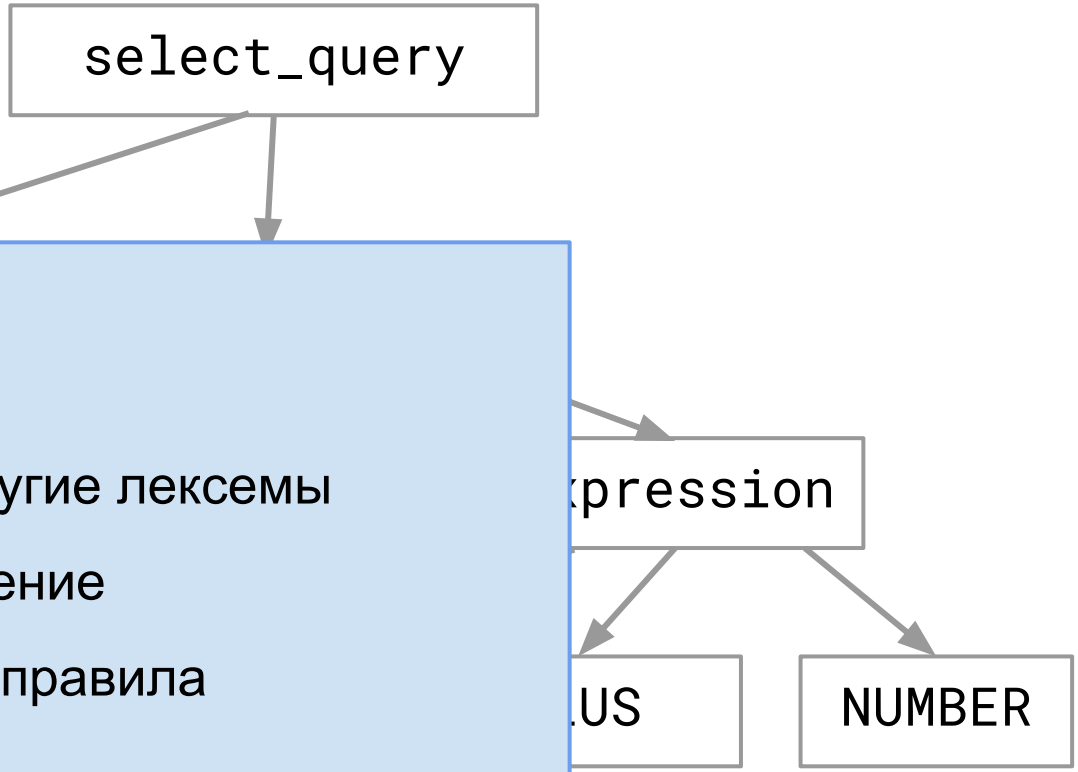
Дерево разбора (Parse Tree)

- N-арное дерево
- Содержит терминалы и нетерминалы



Дерево разбора (Parse Tree)

- N-арное дерево
- Содержит терминалы и нетерминалы

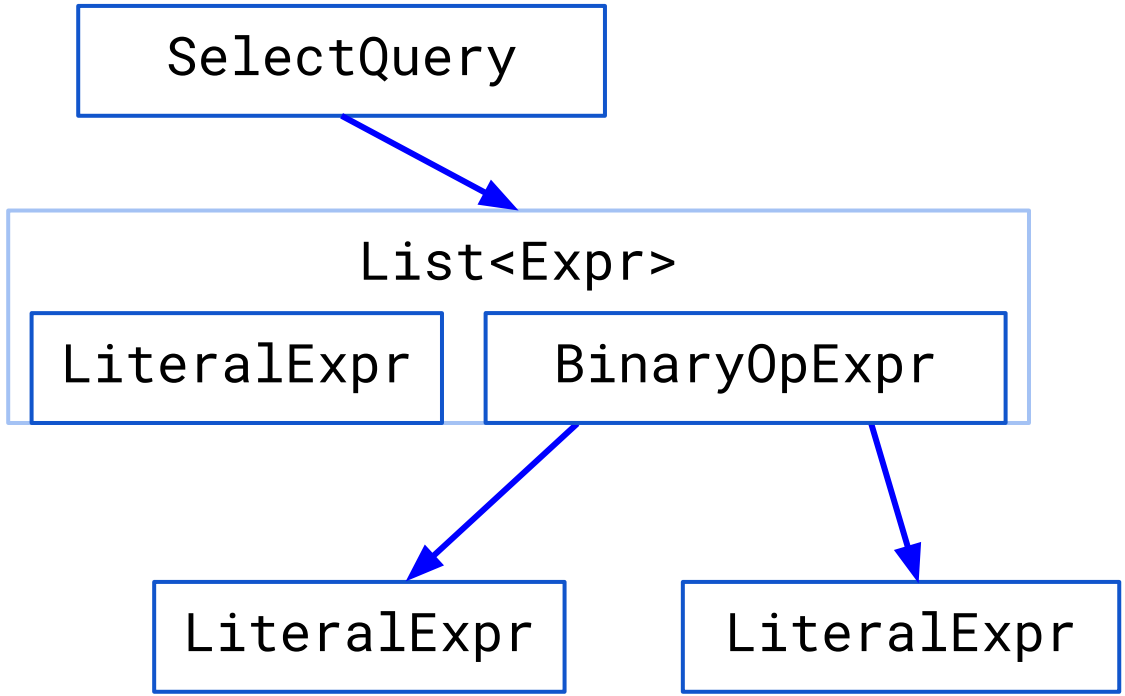


Проблемы дерева разбора:

1. Избыточная информация
   - Ключевые слова и другие лексемы
   - Не влияет на выполнение
2. Разные варианты одного правила

Абстрактное синтаксическое дерево (AST)

- N-арное дерево
- Содержит логические узлы
- Все узлы — подклассы одной иерархии

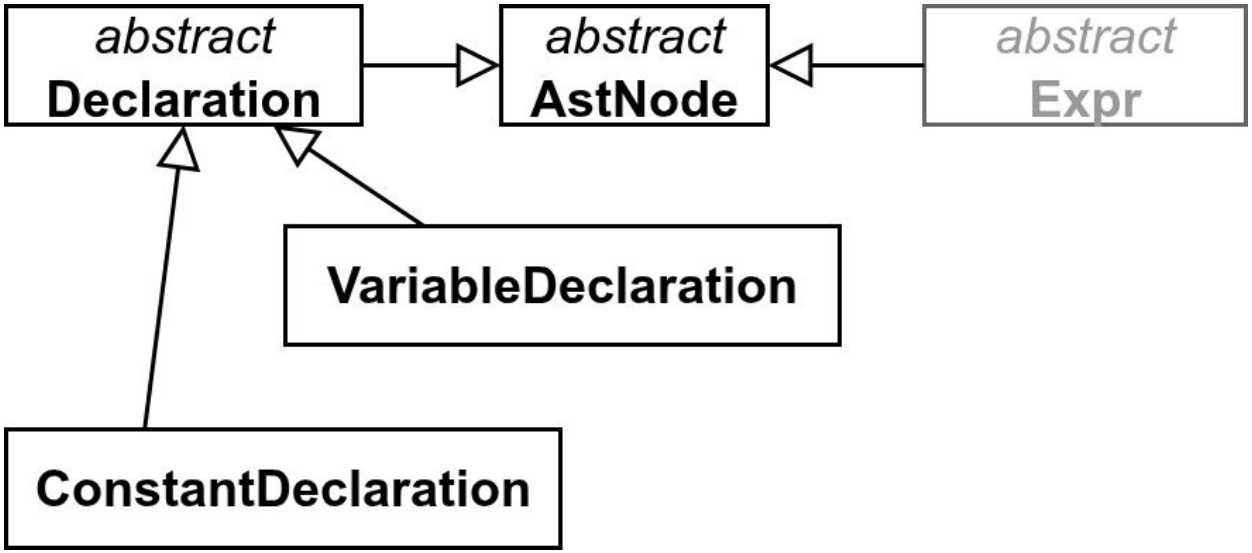


AST языка Kaleidoscope

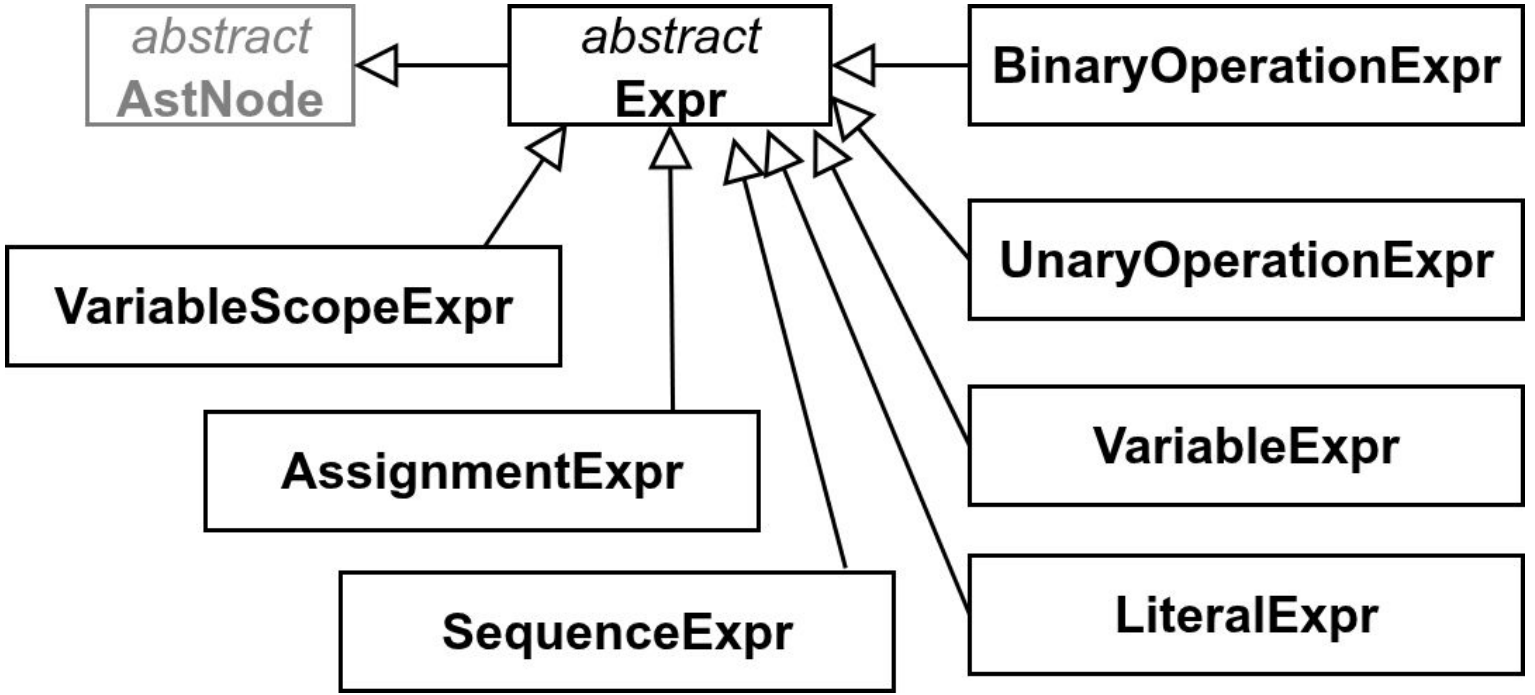
Абстрактные классы

- AstNode — надкласс всех узлов AST
- Declaration — надкласс объявлений
- Expression — надкласс выражений

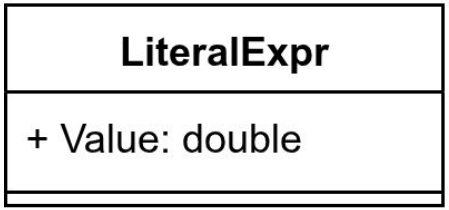
Подклассы объявлений

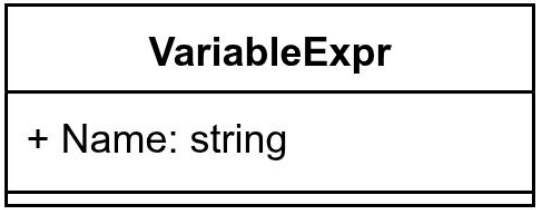


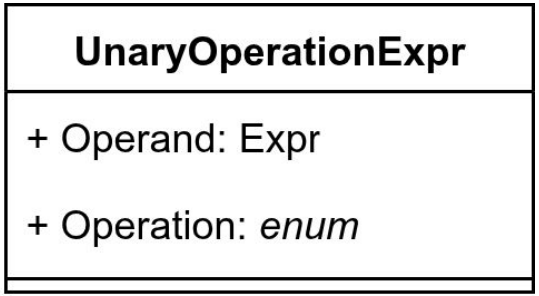
Подклассы выражений

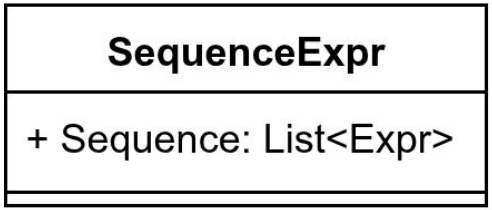


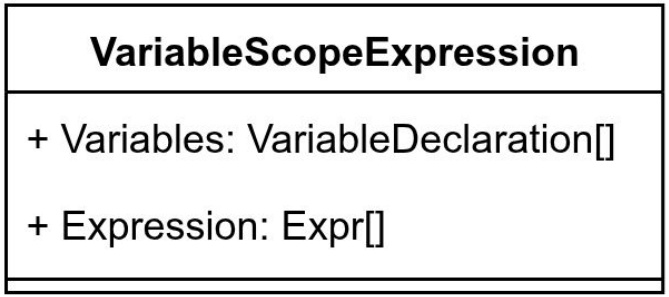
Свойства подклассов выражений

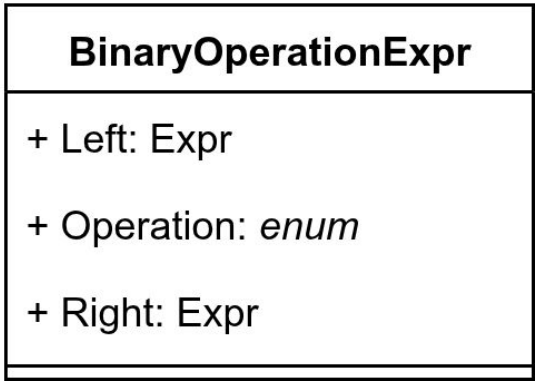












Изменения в парсере

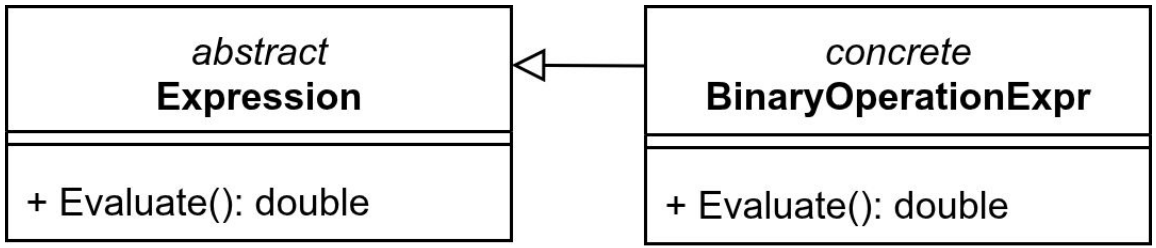
Заменяем прямые вычисления на создание узлов AST

```
case TokenType.Identifier:
    string name = Match(TokenType.Identifier)
        .Value!.ToString();
    return new VariableExpression(name);
```

Обработка AST

Диспетчеризация

1. Диспетчеризация — действие выбирается в зависимости от подтипа объекта
   - Вызываем метод Expression.Evaluate()
   - → вызывается метод BinaryOperationExpr.Evaluate()



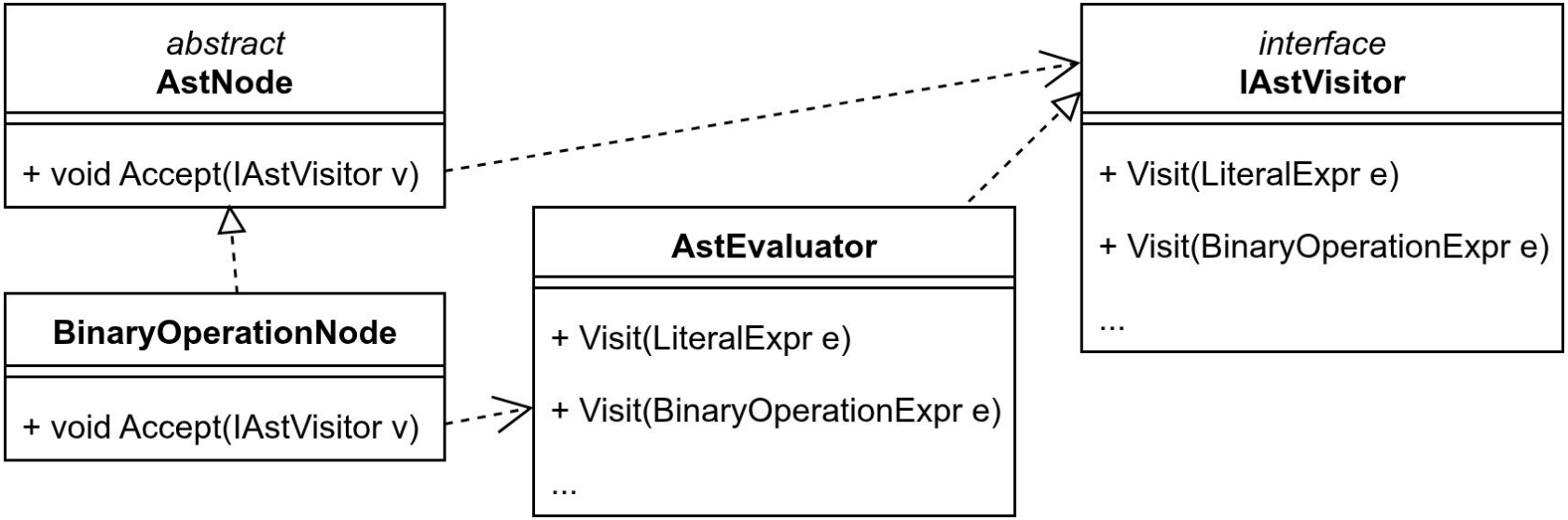
Диспетчеризация

1. Диспетчеризация — действие выбирается в зависимости от подтипа объекта
2. Двойная диспетчеризация — действие выбирается в зависимости от подтипа объекта **и аргумента**
   - Шаблон «Посетитель» (Visitor)
   - Сопоставление (Pattern Matching) по подтипам

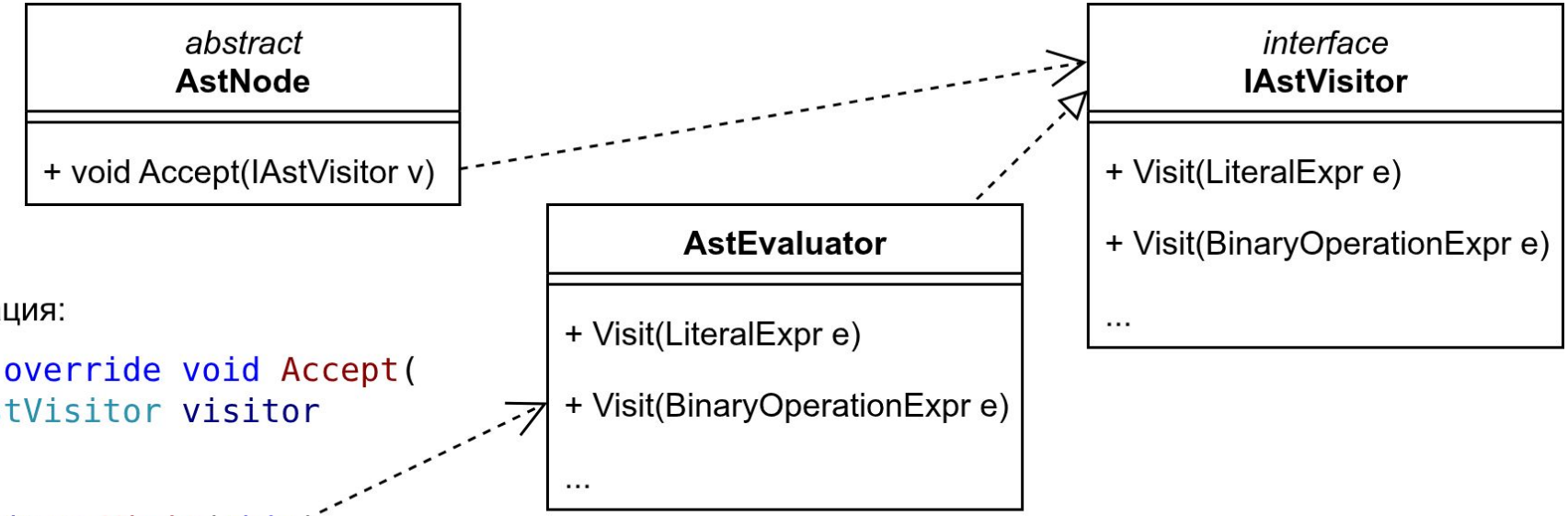
Сопоставление по типу (Pattern Matching by type)

```
return _value switch
{
    string s => s,
    double d => d.ToString(CultureInfo.InvariantCulture),
    _ => throw new NotImplementedException(),
};
```

Шаблон «Посетитель» (Visitor)



Шаблон «Посетитель» (Visitor)



Реализация:

```
public override void Accept(
    IAstVisitor visitor
)
{
    visitor.Visit(this);
}
```

Вычисление выражений по AST

Вычисление выражения по AST

1. Создаём стек
2. Обходим дерево
3. Возвращаем единственное оставшееся значение в стеке

```
public double Evaluate(AstNode node)
{
    node.Accept(this);
    // TODO: бросить исключение, если в стеке не 1 значение
    return _values.Pop();
}
```

Литерал

```
public void Visit(LiteralExpression e)
{
    _values.Push(e.Value);
}
```

Чтение переменной или константы

```
public void Visit(VariableExpression e)
{
    _values.Push(_context.GetValue(e.Name));
}
```

Изменение значения переменной

```
public void Visit(AssignmentExpression e)
{
    // NOTE: Вычисляем выражение, и затем присваиваем
    // его значение переменной, сохраняя результат в стеке.
    e.Value.Accept(this);
    double value = _values.Peek();
    _context.AssignVariable(e.Name, value);
}
```

Унарная операция

```
public void Visit(UnaryOperationExpression e) {
    e.Operand.Accept(this);
    switch (e.Operation) {
        case UnaryOperation.Minus:
            _values.Push(-_values.Pop());
            break;
        case UnaryOperation.Plus:
            break;
        default:
            throw new NotImplementedException(...);
    }
}
```

Бинарная операция

```
public void Visit(BinaryOperationExpression e) {
    e.Left.Accept(this);
    e.Right.Accept(this);
    double right = _values.Pop()
    double left = _values.Pop();
    switch (e.Operation) {
        case BinaryOperation.Plus:
            _values.Push(left + right);
            break;
        // ...
        default:
            throw new NotImplementedException(...);
    }
}
```

Проблемы / решения

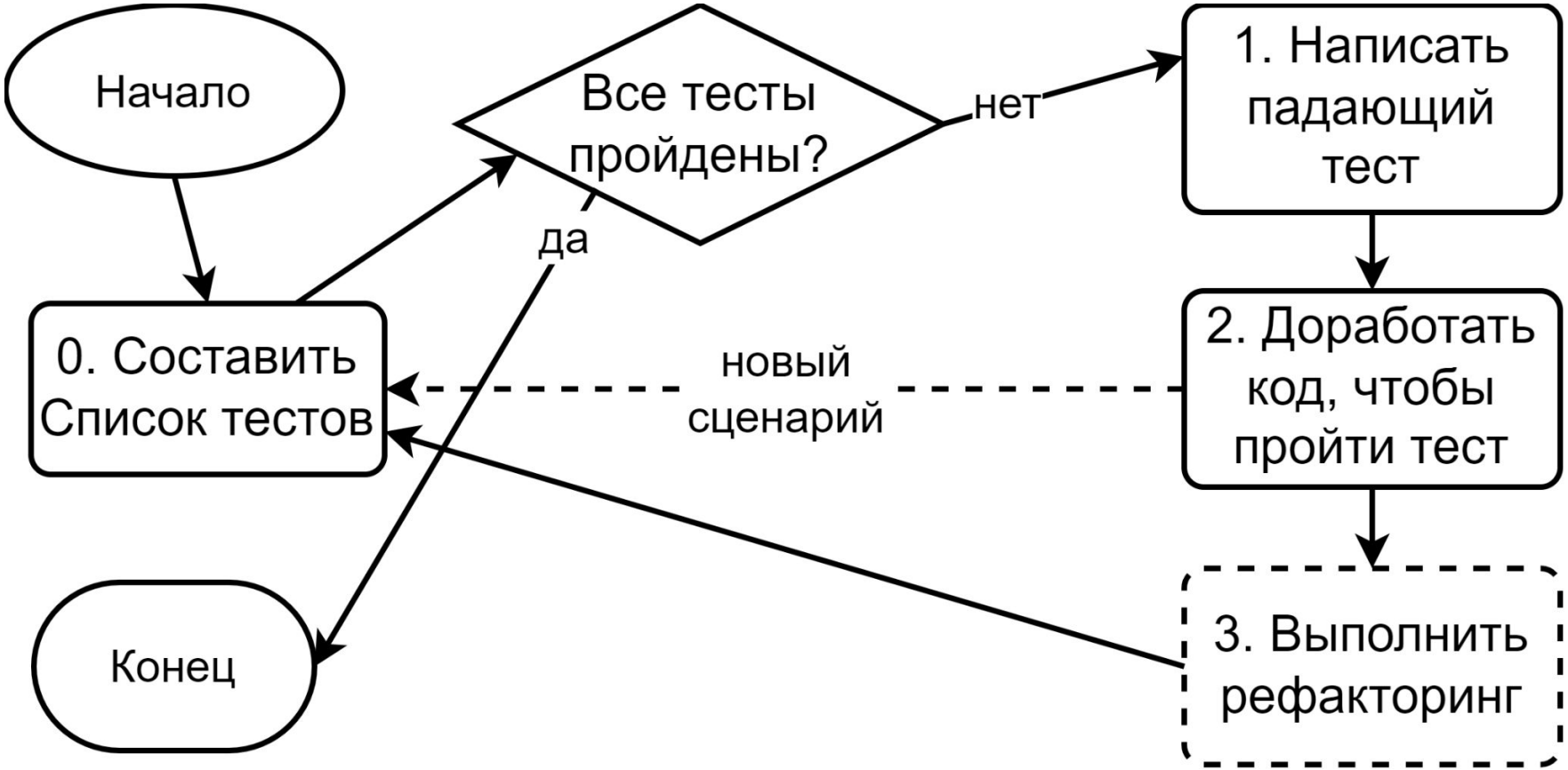
Проблема	Решение
Циклы и функции требуют повторного выполнения	Внедряем AST (Abstract Syntax Tree)

Проблемы / решения

Проблема	Решение
Циклы и функции требуют повторного выполнения	Внедряем AST (Abstract Syntax Tree)
Рефакторинг может сломать программу	???

Модульные тесты

Каноничный TDD



Цели модульного тестирования

1. Защита от регрессий
2. Тесты как документация модуля
3. Уверенность в каждом модуле
4. Тесты как способ проектирования — см. Canon TDD

Цели модульного тестирования

1. Защита от регрессий
2. Тесты как документация модуля
3. Уверенность в каждом модуле
4. Тесты как способ проектирования — см. Canon TDD

Вопрос: какая цель важнее всего?

Цели модульного тестирования

1. Защита от регрессий
2. Тесты как документация модуля
3. Уверенность в каждом модуле
4. Тесты как способ проектирования — см. Canon TDD

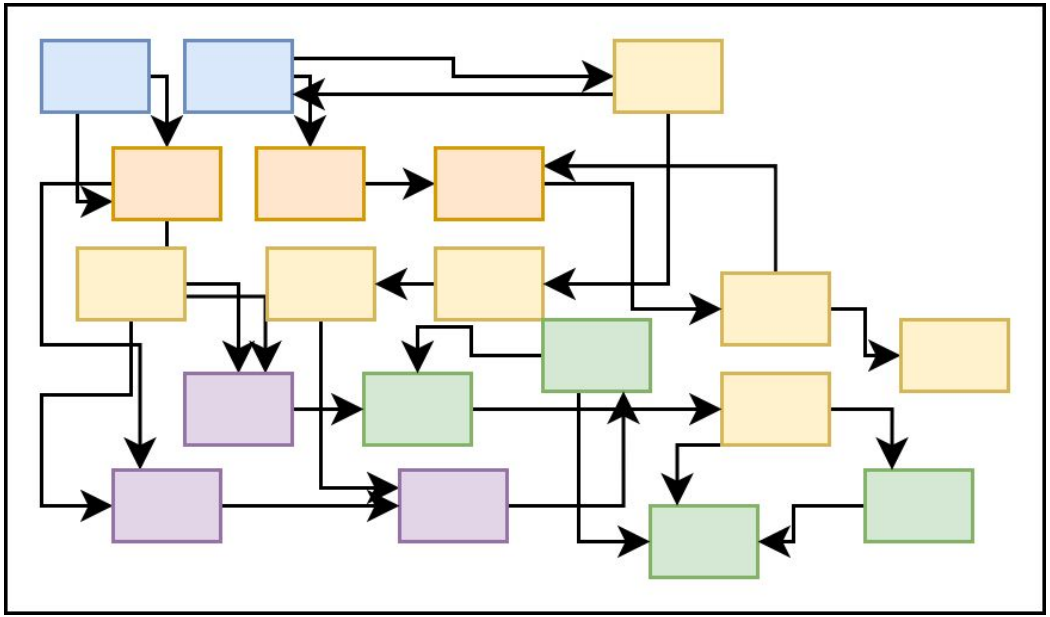
Вопрос: какая цель важнее всего?

Ответ: цели упорядочены по **возрастанию** важности

Разработка системы по TDD

Стиль ООП

Программа — это набор взаимодействующих объектов

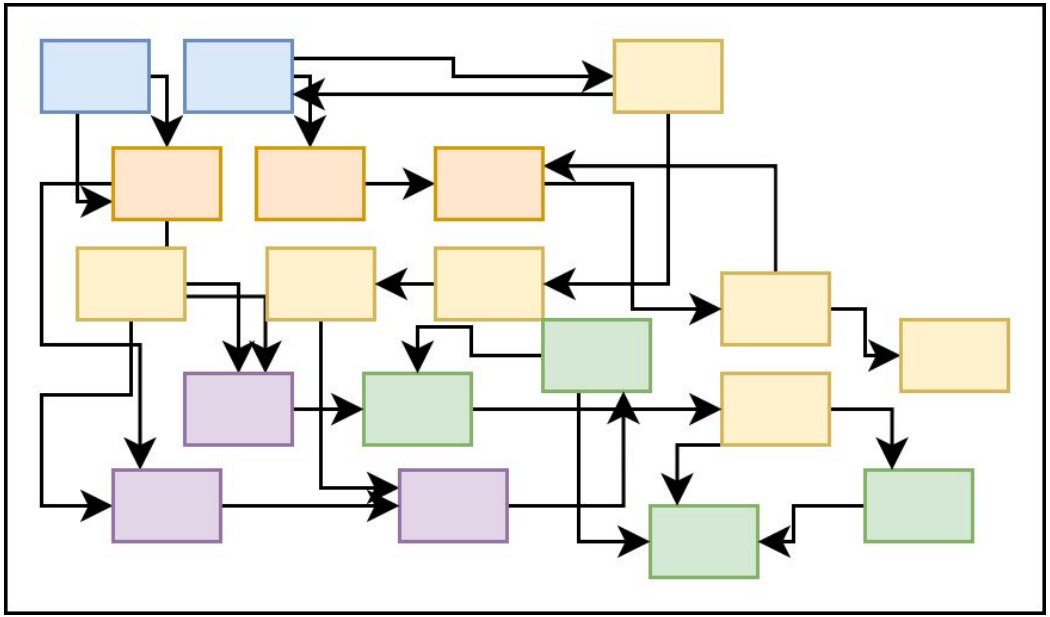


Разработка системы по TDD

1. Делим программу на модули
2. Каждый модуль пишем по TDD

Стиль ООП

Программа — это набор взаимодействующих объектов



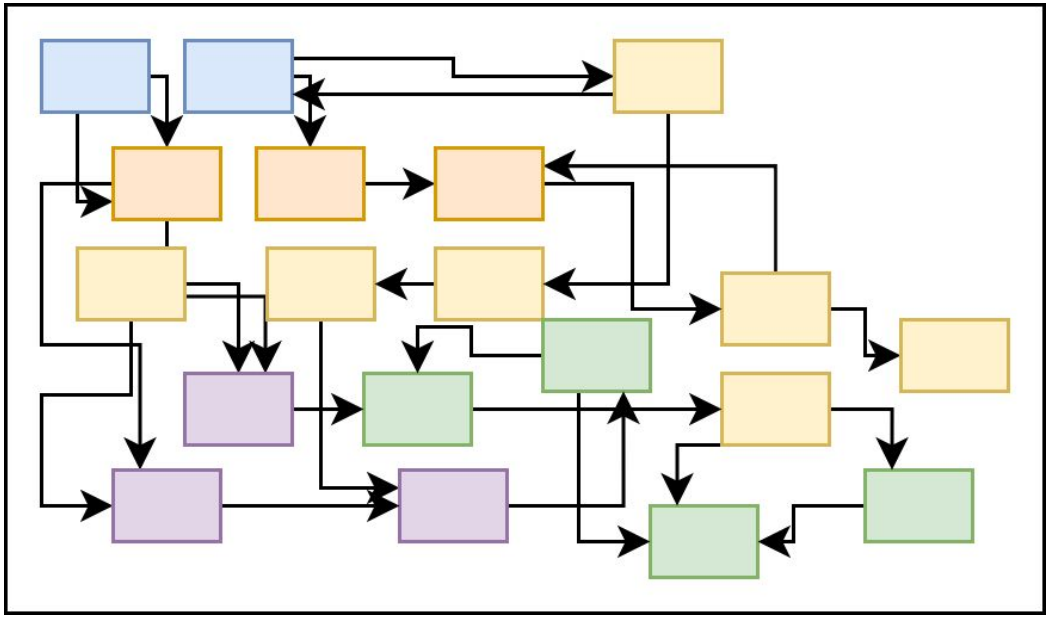
Разработка системы по TDD

1. Делим программу на модули
2. Каждый модуль пишем по TDD

1. UML Class
2. UML Sequence

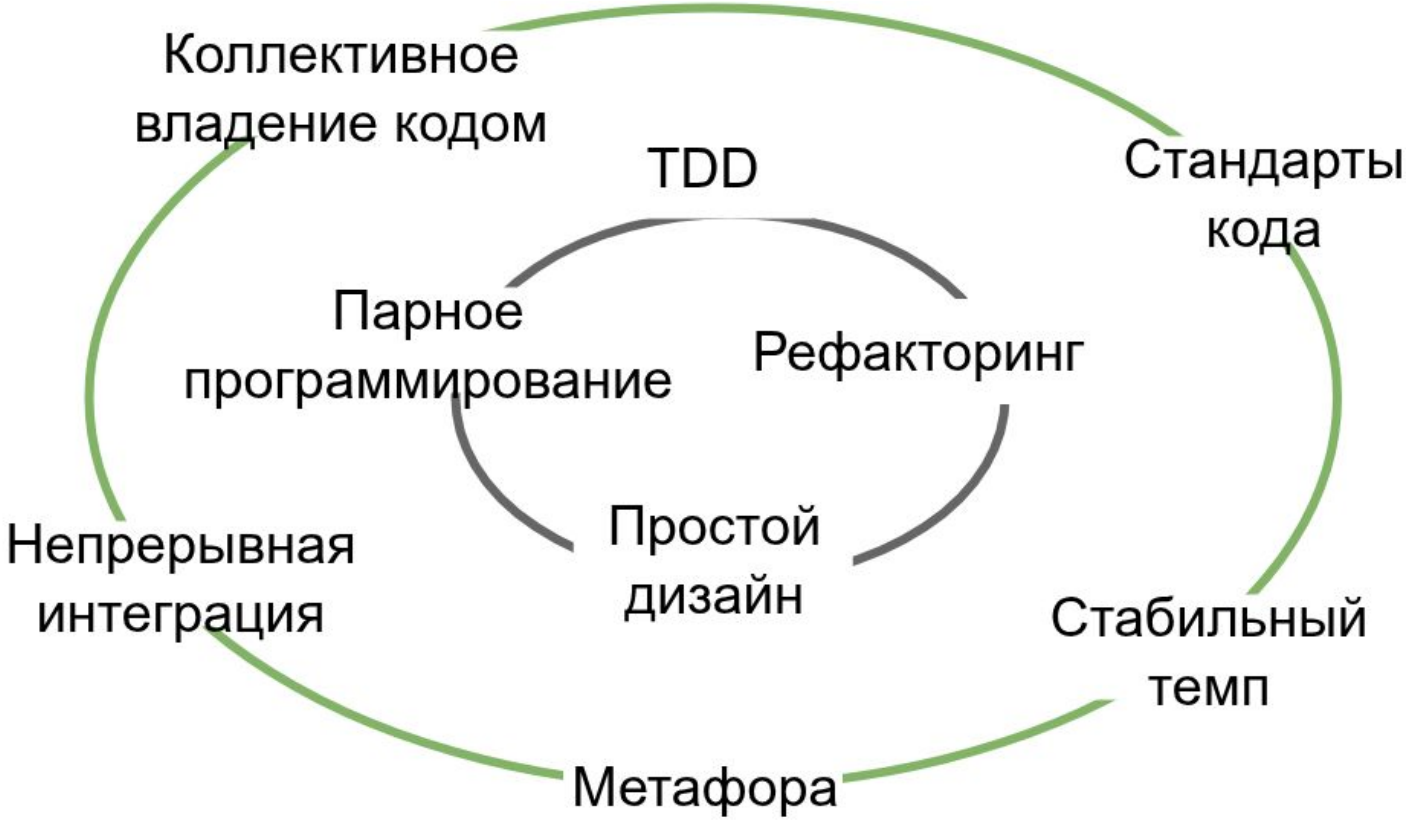
Стиль ООП

Программа — это набор взаимодействующих объектов

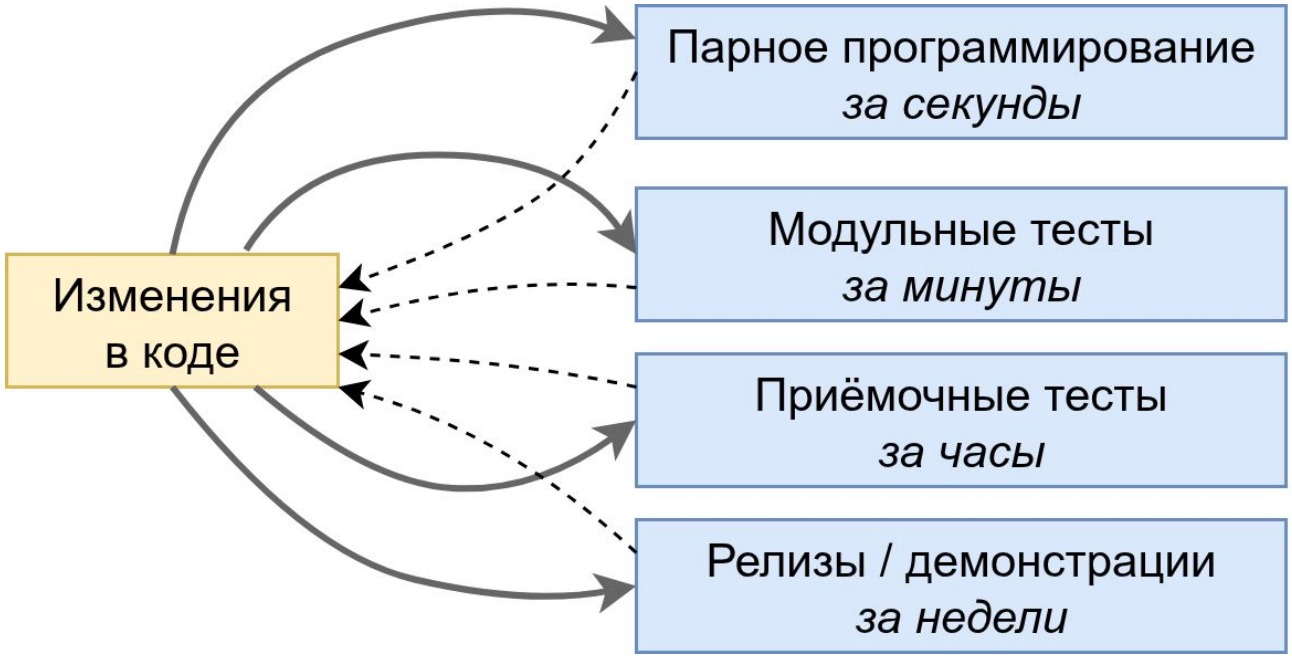


Практики eXtreme Programming

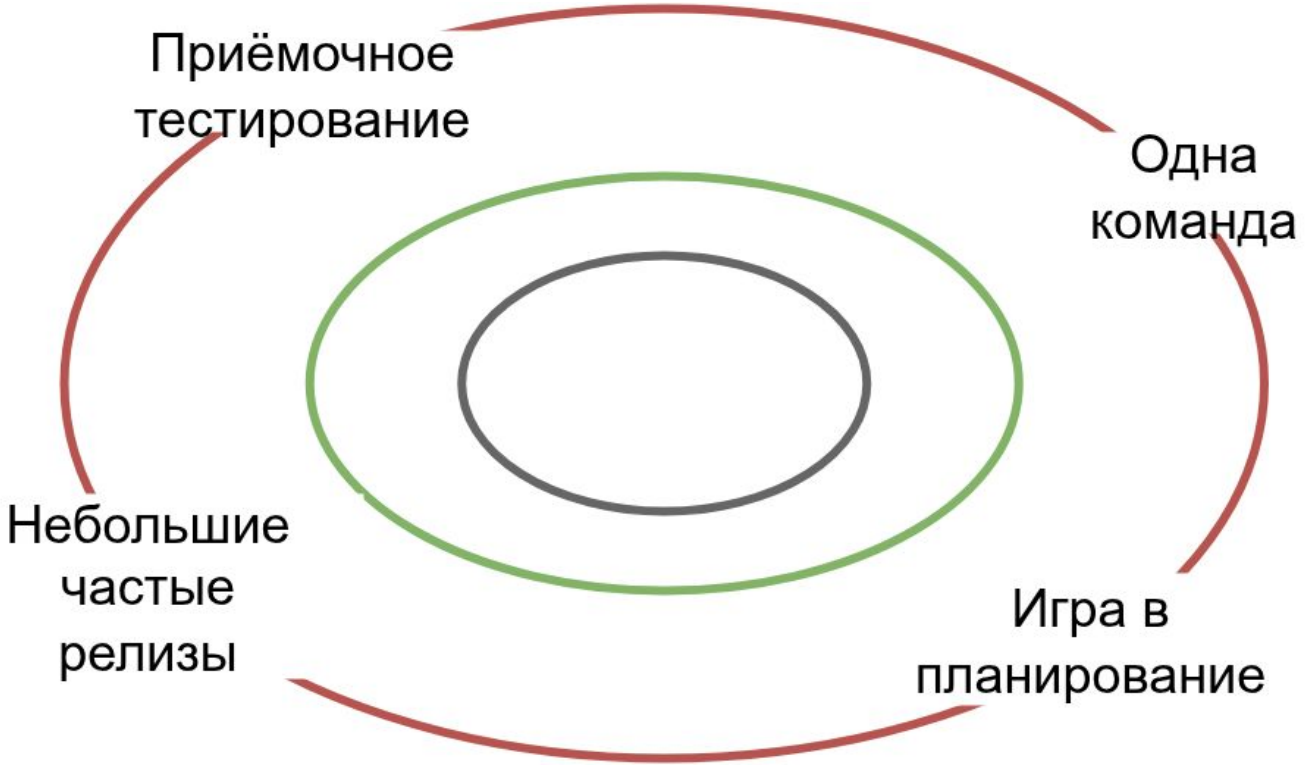
Практики eXtreme Programming — два круга из трёх



Петли обратной связи в Agile (feedback loops)



Практики eXtreme Programming — третий круг



Приёмочные тесты

Функциональное тестирование

Принципы Structural Testing (1970-е):

- Тестирование чёрного ящика
  - Тестируют наблюдаемое извне поведение
  - Пример: ввод/вывод консольной программы
  - Это **функциональные** тесты
- Тестирование прозрачного (*белого*) ящика
  - Тестируют внутренние интерфейсы программы

Приёмочные тесты

Три названия, но суть одна:

- Functional Tests — из Structural Testing, 1970-е
- Customer Tests — из eXtreme Programming, 1998
- Acceptance Tests — из Agile, 2000-е

Проблемы / решения

Проблема	Решение
Циклы и функции требуют повторного выполнения	Внедряем AST (Abstract Syntax Tree)
Рефакторинг может сломать программу	Внедряем приёмочные тесты (функциональные тесты)

Проблемы / решения

Проблема	Решение
Циклы и функции требуют повторного выполнения	Внедряем AST (Abstract Syntax Tree)
Рефакторинг может сломать программу	Внедряем приёмочные тесты (функциональные тесты)
Как отделить тесты от реализации?	???

Язык Gherkin

Язык Gherkin

1. *Почти* естественный язык описания тестов

Язык Gherkin

1. *Почти* естественный язык описания тестов
2. Реализует принцип Specification by Example

Язык Gherkin

1. *Почти* естественный язык описания тестов
2. Реализует принцип Specification by Example
3. Транслируется в вызовы функций при запуске тестов
    - Reqnroll (C#) — компилирует Gherkin в C#
    - cucumber/godog (Go) — интерпретирует Gherkin, все шаги на Go
    - cucumber/js — интерпретирует Gherkin, все шаги на JS / TS

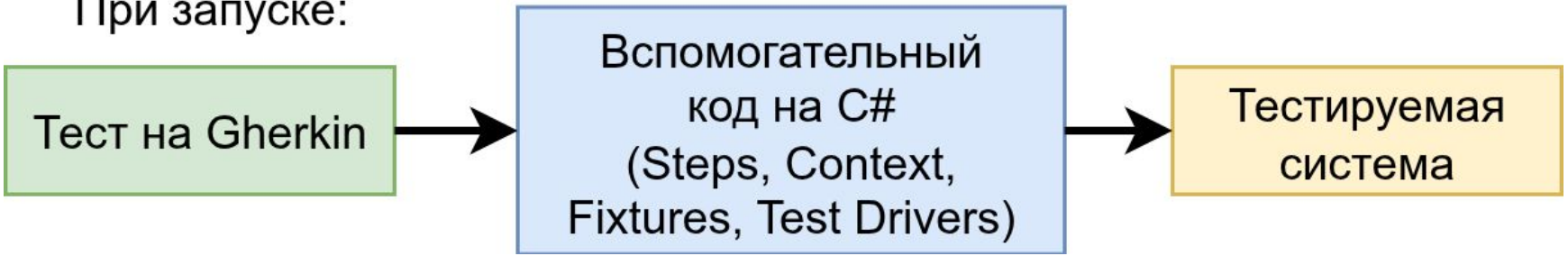
Пример тестов на Gherkin

```
Функциональность: Последовательное выполнение

    Сценарий: вычисление площади круга

        Когда я выполняю программу:
        """

        ... код …
        """

        Тогда я получаю результаты:
          | Value    |
          | 3.14159  |
```

Поддержка Gherkin в Reqnroll (C#)

При сборке:

При запуске:



Проблемы / решения

Проблема	Решение
Циклы и функции требуют повторного выполнения	Внедряем AST (Abstract Syntax Tree)
Рефакторинг может сломать программу	Внедряем приёмочные тесты (функциональные тесты)
Как отделить тесты от реализации?	Используем Gherkin и Cucumber / Reqnroll

Трёхэтапный процесс рефакторинга

1. Покрываем код приёмочными тестами
    - Они устойчивы к изменению архитектуры
2. ???
3. ???

Трёхэтапный процесс рефакторинга

1. Покрываем код приёмочными тестами
    - Они устойчивы к изменению архитектуры
2. Проводим рефакторинг
    - Добавляем AST
3. ???

Трёхэтапный процесс рефакторинга

1. Покрываем код приёмочными тестами
    - Они устойчивы к изменению архитектуры
2. Проводим рефакторинг
    - Добавляем AST
3. Реализуем новую функциональность
    - Ветвления, циклы, функции

Спасибо за внимание!
Вопросы?