arcadia
software country

Быстрый расчет формул из Excel на C#

Мулюкин А.А.

Senior Software Developer, Arcadia

Знакомство

Алексей

- Senior Software Engineer
- Опыт разработки более 9 лет
- Сфера интересов:
  - Распределенные вычисления
  - Проектирование
  - C#, Node.js, PHP, Python

- Email: Alexey.Mulyukin@arcadia.spb.ru
- GitHub: https://github.com/alexprey

План доклада

- Excel и для чего его готовят
- Постановка задачи
- Поиск библиотек
- Прокачиваем производительность

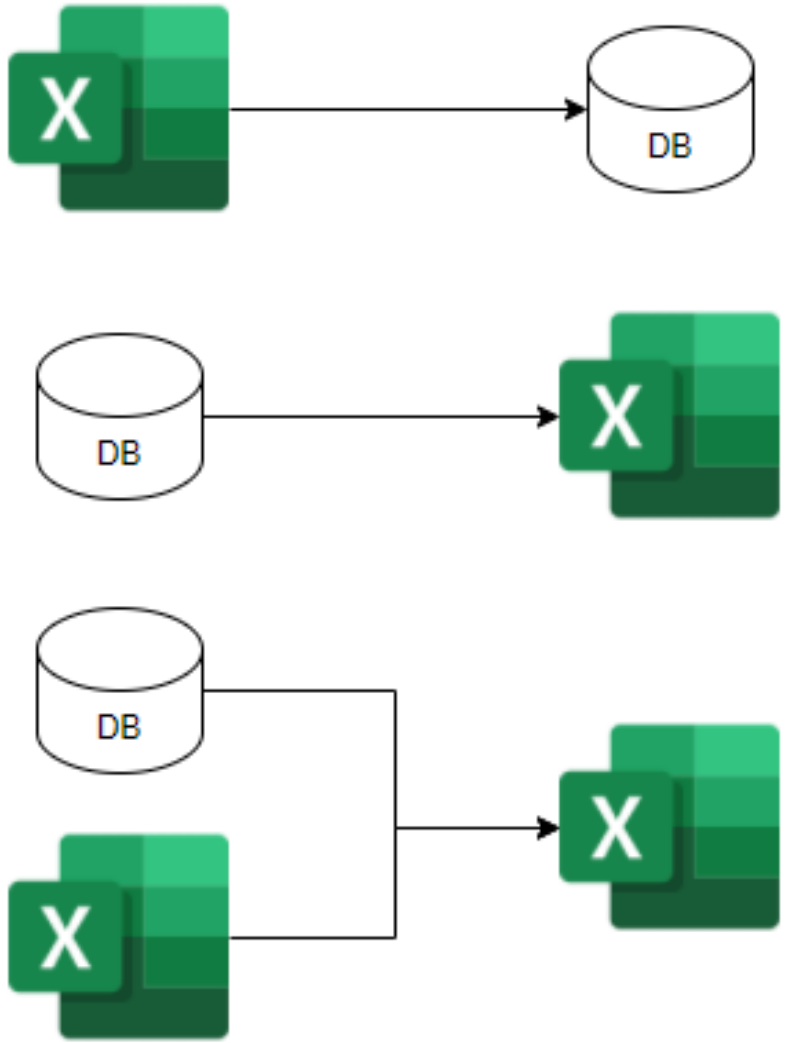
Работа с Excel

- Хранение данных
- Реализация бизнес-процессов
- Комплексные приложения
- Математическое моделирование / Прогнозирование данных

Работа с Excel

Классические задачи интеграции

- Перенос данных из Excel
- Выгрузка данных в Excel
- Построение шаблонных отчетов



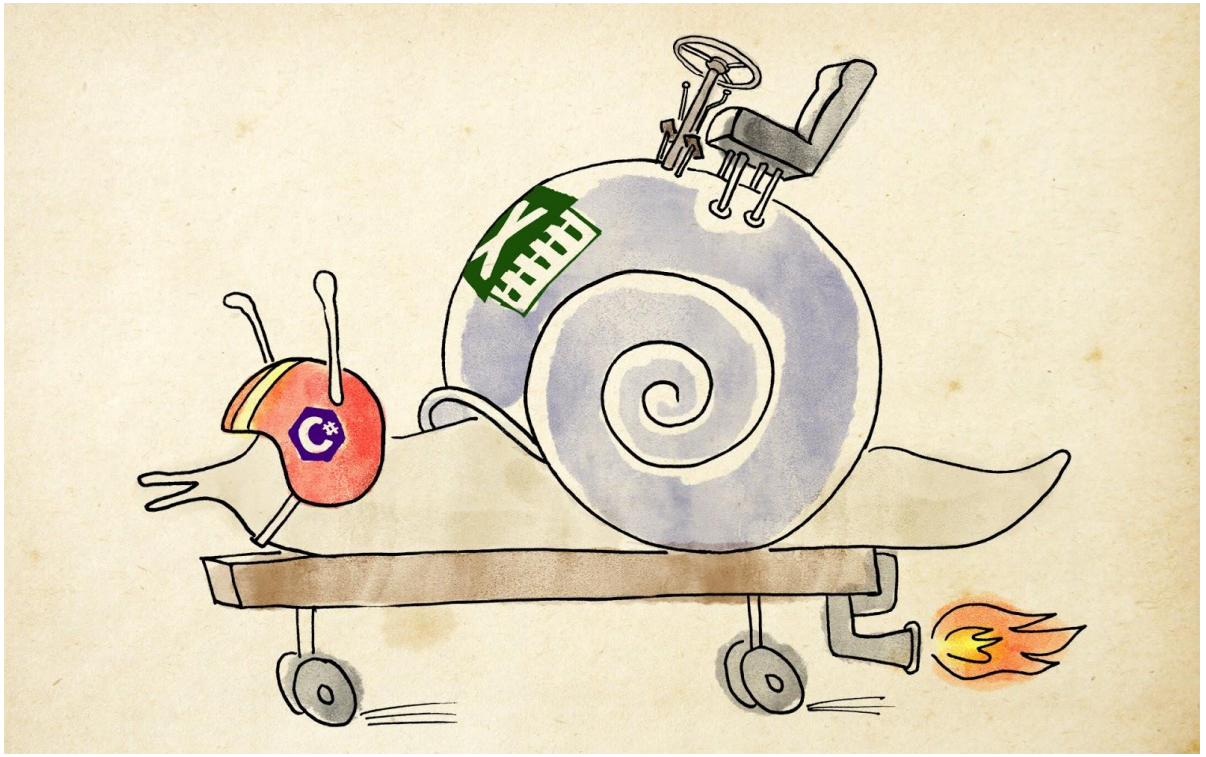
Работа с Excel

Интересные задачи интеграции

- Excel как основной интерфейс для пользователя
- Excel как центр хранения данных

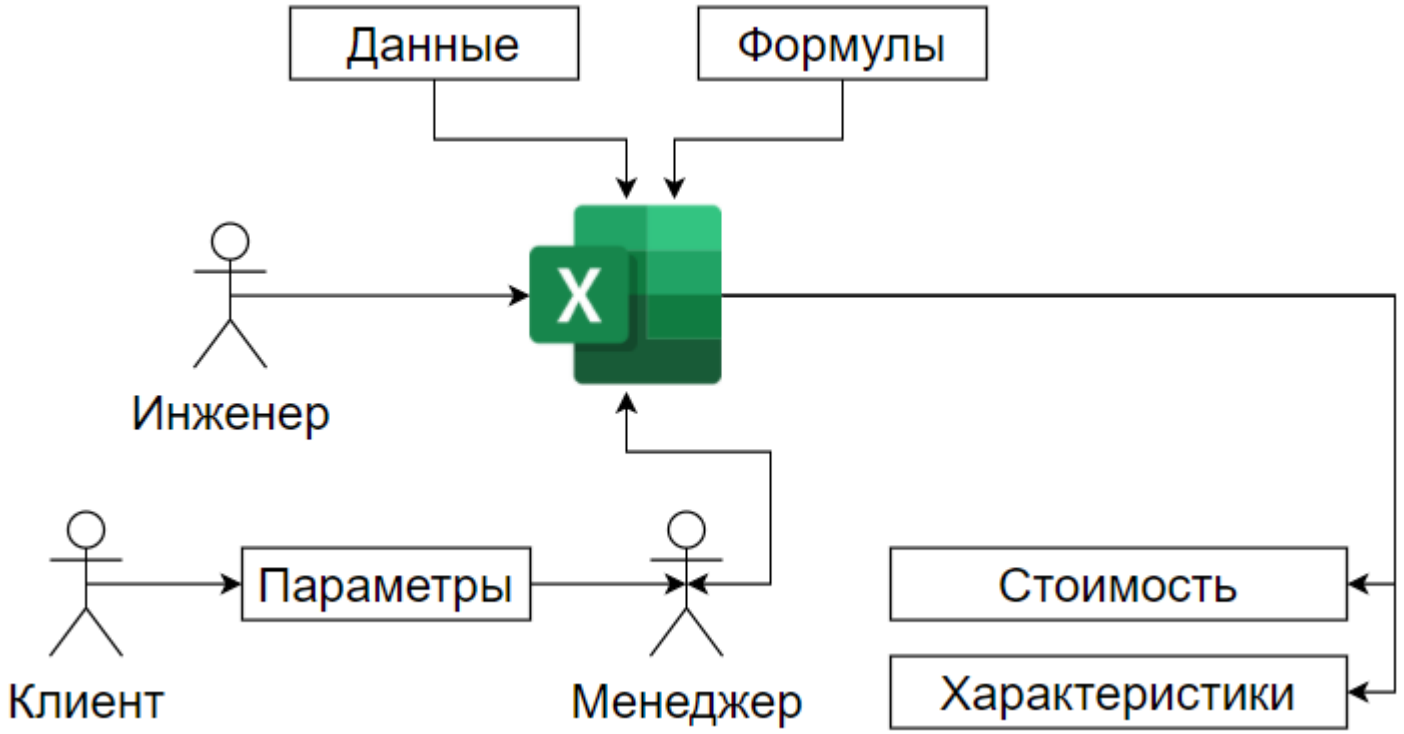
Проблемы интеграции с Excel

- Отсутствие нормального API из коробки
- Производительность



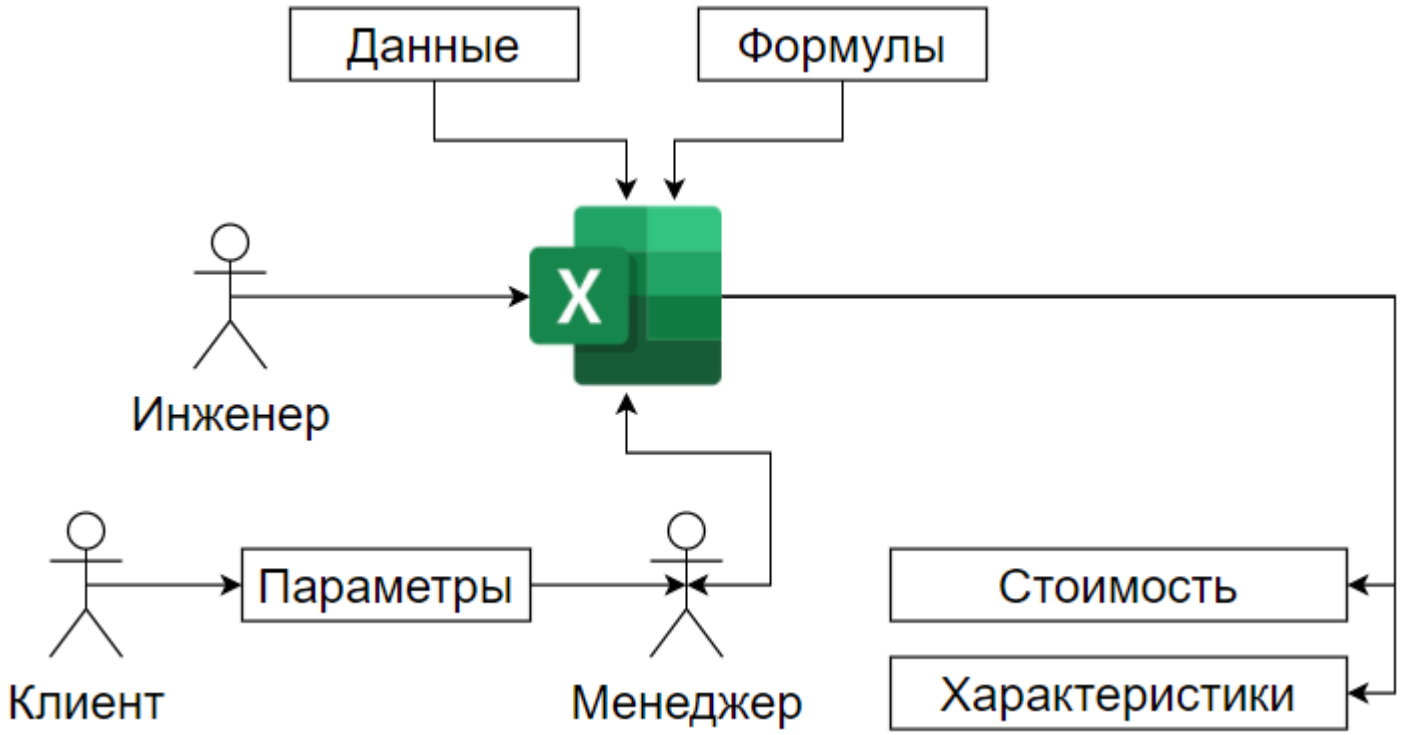
Исходные

- Расчет стоимости производства
- Расчет технических характеристик продукта
- Все данные в Excel
- Исходный файл живой



Проблемы

- Низкая надежность
- Высокая сложность
- Длительный цикл обратной связи

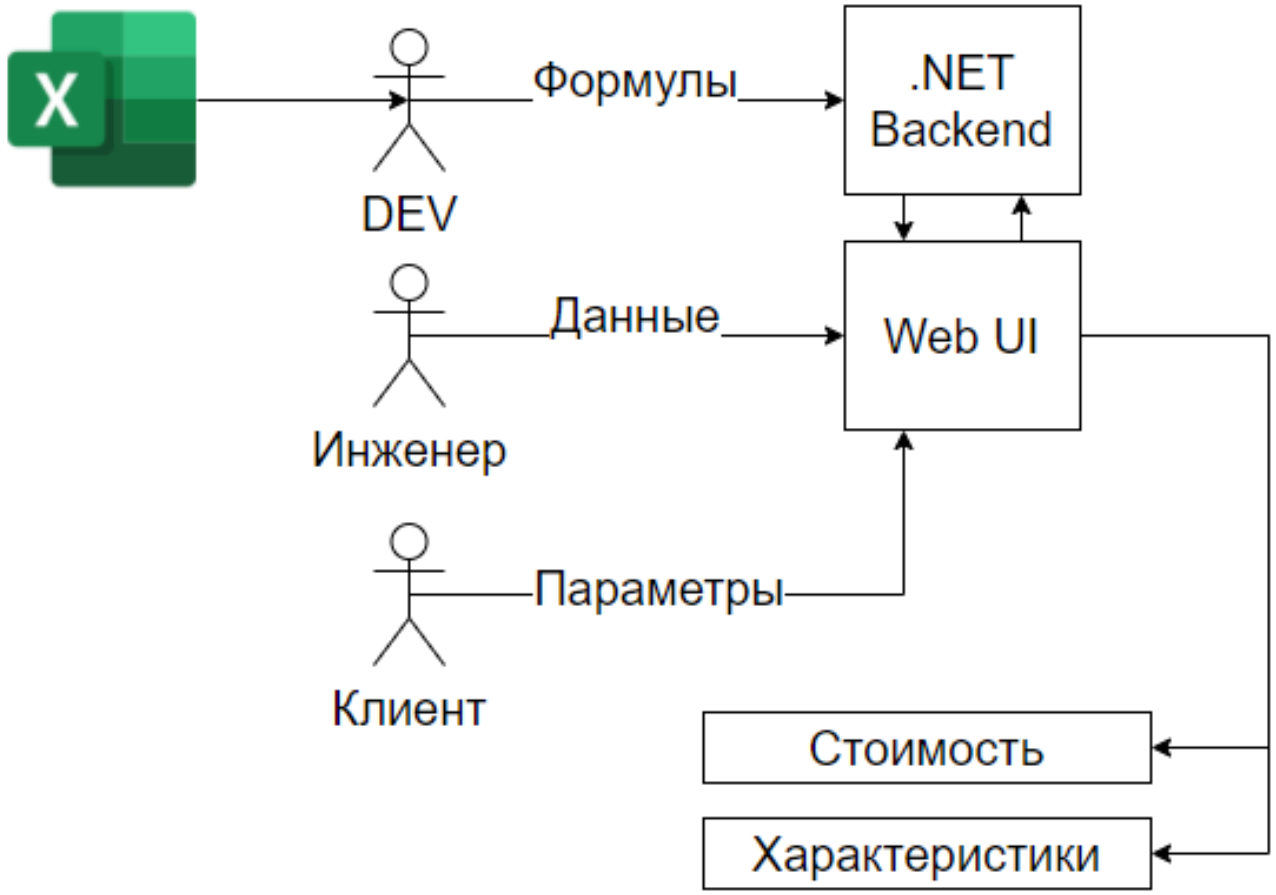


Требования

- ASP.NET
- Быстрая обратная связь взаимодействия
- Задача оптимизации параметров
- Точность
- Надежность
- Изменяемый Excel на лету
- Сжатые сроки

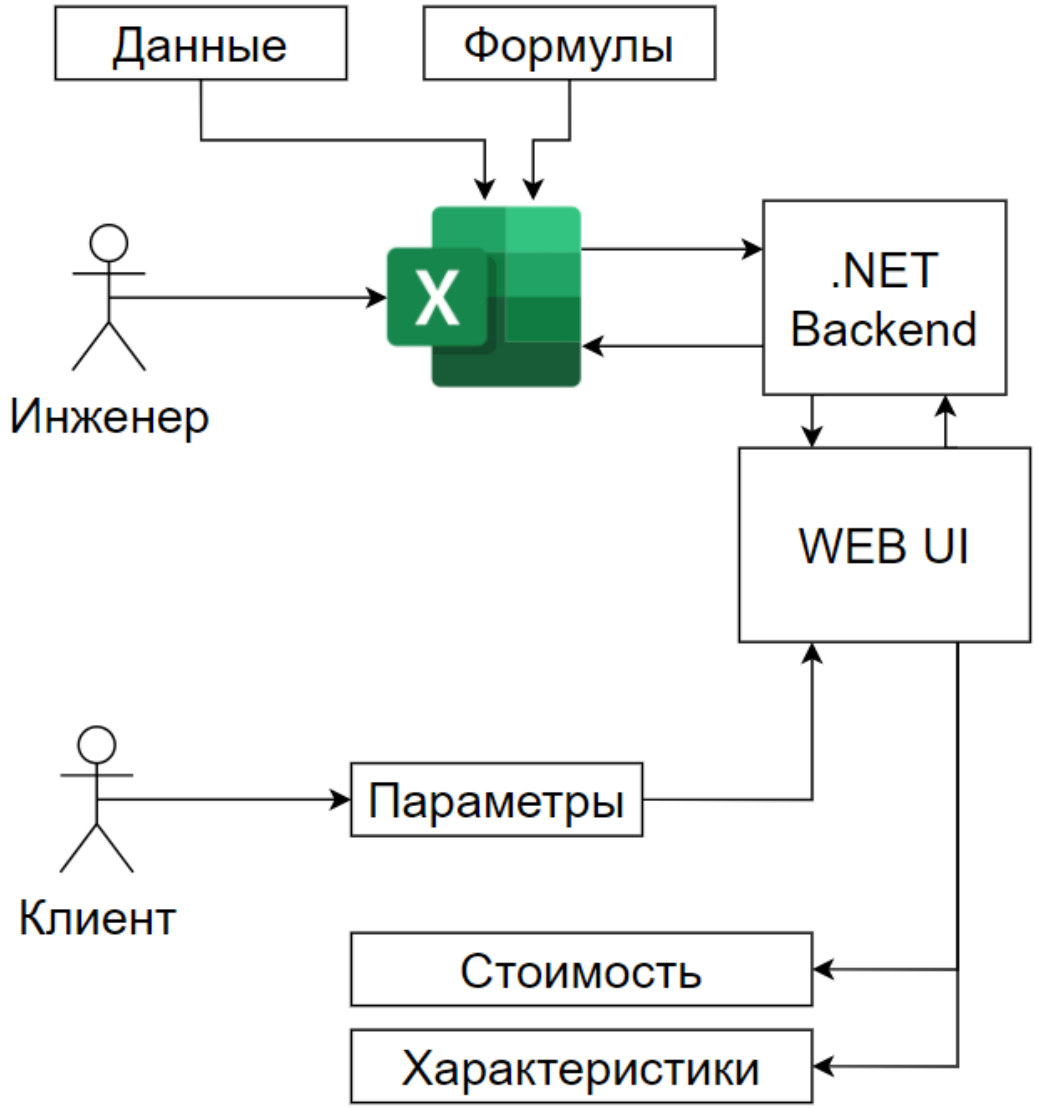
Вариант 1

- Свой интерфейс
- Ручной перенос расчетов



Вариант 2

- Интерфейс Excel
- API Backend
- Excel как центр хранения экономической модели и данных



Поиск решения

Шаг 1. Изучаем существующие решения

Требования к библиотеке

- Производительность
  - API ответ < 500 мс
- Поддержка формул Excel
- Поддержка расчетов
- Поддержка новых форматов
- Точность результатов в сравнении с Excel

Старый формат (xls) VS Новый (xlsx)

	XLS	XLSX
Формат хранения	Собственный закрытый бинарный формат	На основе открытого стандарта XML
Ограничение на размер	65536 строк 256 столбцов	1048576 строк 16384 столбцов
Поддержка макросов	Да	Нет
Производительность	Работает быстрее на сложных формулах	
Совместимость		Поддерживается начиная с Office 2007

Первый взгляд

Библиотека	Ссылка	Документация	Лицензия
EPPlus 4	github.com/JanKallman/EPPlus	github.com/JanKallman/EPPlus/wiki	GNU Library General Public License
EPPlus 5	github.com/EPPlusSoftware/EPPlus	github.com/EPPlusSoftware/EPPlus/wiki	Polyform Noncommercial License 1.0.0 (www.epplussoftware.com/LicenseOverview)
NPOI	github.com/tonyqus/npoi	github.com/tonyqus/npoi/wiki	Apache License 2.0
Spire	www.e-iceblue.com/Introduce/excel-for-net-introduce.html	www.e-iceblue.com/Tutorials/Spire.XLS/Spire.XLS-Program-Guide/Spire.XLS-Program-Guide-Content.html	www.e-iceblue.com/Tutorials/Licensing/License-Agreement.html
Excel Interop *(Microsoft.Office.Interop.Excel)*	n/a	docs.microsoft.com/ru-ru/dotnet/api/microsoft.office.interop.excel._workbook?view=excel-pia	*Требует предустановленного Excel со всеми вытекающими последствиями лицензирования*

Excel Interop

```
18      public void SetUp()
19      {
20          _application = new Application
21          {
22              Visible = false,
23              SheetsInNewWorkbook = 1,
24              DisplayAlerts = false
25          };
26
27          _workbook = _application.Workbooks.Open(_excelFilePath);
28          _worksheet = (Worksheet) _workbook.Sheets.Item[1];
29      }
30
31      public double Execute(double[] p)
32      {
33          for (int rowIndex = 0; rowIndex < 10; rowIndex++)
34          {
35              _worksheet.Cells[rowIndex + 1, 2] = p[rowIndex];
36          }
37
38          _worksheet.Calculate();
39
40          return (double)((Range)_worksheet.Cells[11, 2]).Value;
41      }
```

- Требует предустановленный Excel на хост машине
- Запускает полный инстанс Excel.exe
- Иногда случаются ошибки соединения

EPPlus

```
25      public double Execute(double[] p)
26      {
27          for (int rowIndex = 0; rowIndex < 10; rowIndex++)
28          {
29              _worksheet.Cells[rowIndex + 1, 2].Value = p[rowIndex];
30          }
31
32          _worksheet.Calculate();
33
34          return _worksheet.Cells[11, 2].GetValue<double>();
35      }
```

- Работает напрямую с Excel файлом
- Не поддерживает старые форматы
- Удобный API
- .NET Core версия распространяется под платной лицензией

NPOI

```
22      public void SetUp()
23      {
24          _workbook = new XSSFWorkbook(_fileName);
25          _worksheet = _workbook.GetSheetAt(0);
26
27          evaluator = WorkbookFactory.CreateFormulaEvaluator(_workbook);
28
29          _parametersCell = new ICell[10];
30          for (int rowIndex = 0; rowIndex < 10; rowIndex++)
31          {
32              _parametersCell[rowIndex] = _worksheet.GetRow(rowIndex).GetCell(1);
33          }
34
35          _resultCell = _worksheet.GetRow(10).GetCell(1);
36      }
37
38      public double Execute(double[] p)
39      {
40          _worksheet.ForceFormulaRecalculation = true;
41          for (int rowIndex = 0; rowIndex < 10; rowIndex++)
42          {
43              _parametersCell[rowIndex].SetCellValue(p[rowIndex]);
44          }
45
46          evaluator.EvaluateAll();
47
48          return _resultCell.NumericCellValue;
49      }
```

- .NET Standard 2.0!
- Поддержка старых версий Excel файлов (xls)
- Неочевидные проблемы производительности

Spire

```
38      public double Execute(double[] p)
39      {
40          for (int rowIndex = 0; rowIndex < 10; rowIndex++)
41          {
42              _inputsCell[rowIndex].NumberValue = p[rowIndex];
43          }
44
45          _workbook.CalculateAllValue();
46
47          try
48          {
49              return (double)_resultCell.FormulaValue;
50          }
51          catch (InvalidCastException)
52          {
53              TearDown();
54              SetUp();
55
56              throw;
57          }
58      }
```

- .NET Support (Требует PRO лицензию)
- Поддержка старых и новых форматов файлов
- Платное многоуровневое лицензирование
- Отсутствие механизма самовосстановления при ошибках

Поиск решения

Шаг 2. Анализ и тестирование

Подготовка данных

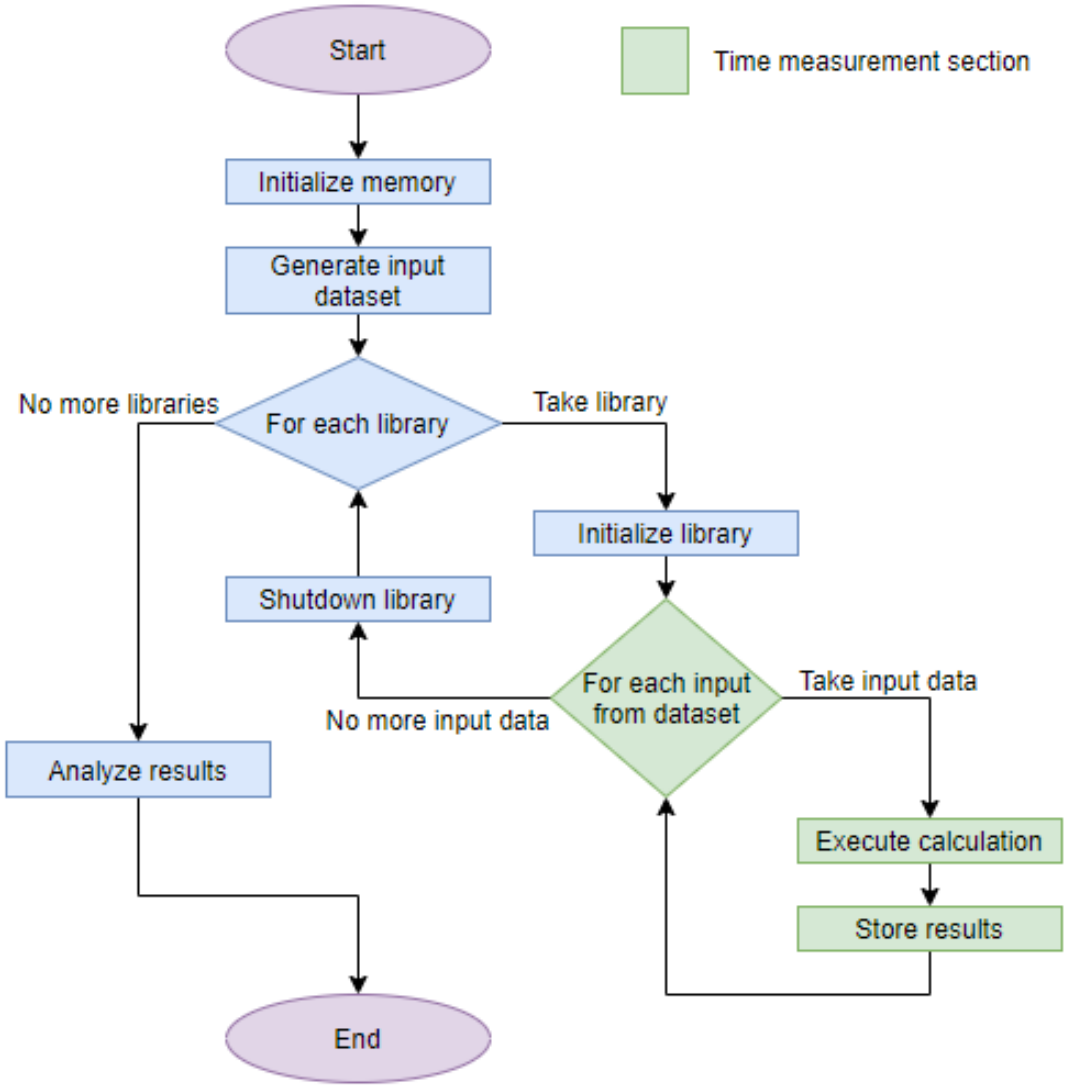
B11 | =POWER(B1*B9/B5*SIN(B6)*COS(B3)+ABS(B2-SUM(B3:B9))*SQRT(B1*B1+B2*B2)/2*PI(), B10)

	A	B
1	P1	1
2	P2	2
3	P3	3
4	P4	4
5	P5	5
6	P6	6
7	P7	7
8	P8	8
9	P9	9
10	P10	10
11	Total	3.10465E+21
12		

```
public double Execute(double[] p)
{
    return Math.Pow(p[0] * p[8] / p[4] * Math.Sin(p[5]) * Math.Cos(p[2]) +
                    Math.Abs(p[1] - (p[2] + p[3] + p[4] + p[5] + p[6] + p[7] + p[8]))
                    * Math.Sqrt(p[0] * p[0] + p[1] * p[1]) / 2.0 * Math.PI, p[9]);
```

Формируем тест

- Время инициализации
- Время расчета на 1 проход
- Ср. кв. отклонение
- Точность
- % Ошибок (exceptions rate per execution)



Генерация данных

```
32      var rnd = new Random((int)DateTime.Now.Ticks);
33
34      var parametersAtlas = new double[_iterationsCount][];
35      var resultsAtlas = new double[_iterationsCount][];
36
37      for (int iterationIndex = 0; iterationIndex < _iterationsCount; iterationIndex++)
38      {
39          double[] parameters = new double[10];
40          for (int parameterIndex = 0; parameterIndex < parameters.Length; parameterIndex++)
41          {
42              parameters[parameterIndex] = (double)rnd.Next(-5000, 5000) / 1000.0;
43          }
44
45          parametersAtlas[iterationIndex] = parameters;
46          resultsAtlas[iterationIndex] = new double[executors.Length];
```

Время операции и данные

```
56      var initializationStopwatch = new Stopwatch();
57      initializationStopwatch.Start();
58      executor.SetUp();
59      initializationStopwatch.Stop();
60
61      Console.WriteLine($"{executorName}: Initialization time = {initializationStopwatch.ElapsedMilliseconds} ms");
62
63      var executionStopwatch = new Stopwatch();
64      executionStopwatch.Start();
65      for (int iterationIndex = 0; iterationIndex < _iterationsCount; iterationIndex++)
66      {
67          try
68          {
69              resultsAtlas[iterationIndex][executorIndex] = executor.Execute(parametersAtlas[iterationIndex]);
70          }
71          catch (Exception e)
72          {
73              errorsCount[executorIndex]++;
74              resultsAtlas[iterationIndex][executorIndex] = double.NaN;
75          }
76      }
77
78      executionStopwatch.Stop();
79      var averageExecutionTime = (double) executionStopwatch.ElapsedMilliseconds / _iterationsCount;
```

Ср.кв. отклонение

```
106     var idealResult = resultsPerExecutor[0];
107     var result = resultsPerExecutor[executorIndex];
108     if (!double.IsNaN(result) && !double.IsNaN(idealResult))
109     {
110         var delta = idealResult - result;
111         deviation[executorIndex] += delta * delta;
112         validValuesCount[executorIndex]++;
113 
114         if (Math.Abs(delta) < _epsilon)
115         {
116             accuracySuccessCount[executorIndex]++;
117         }
118     }
```

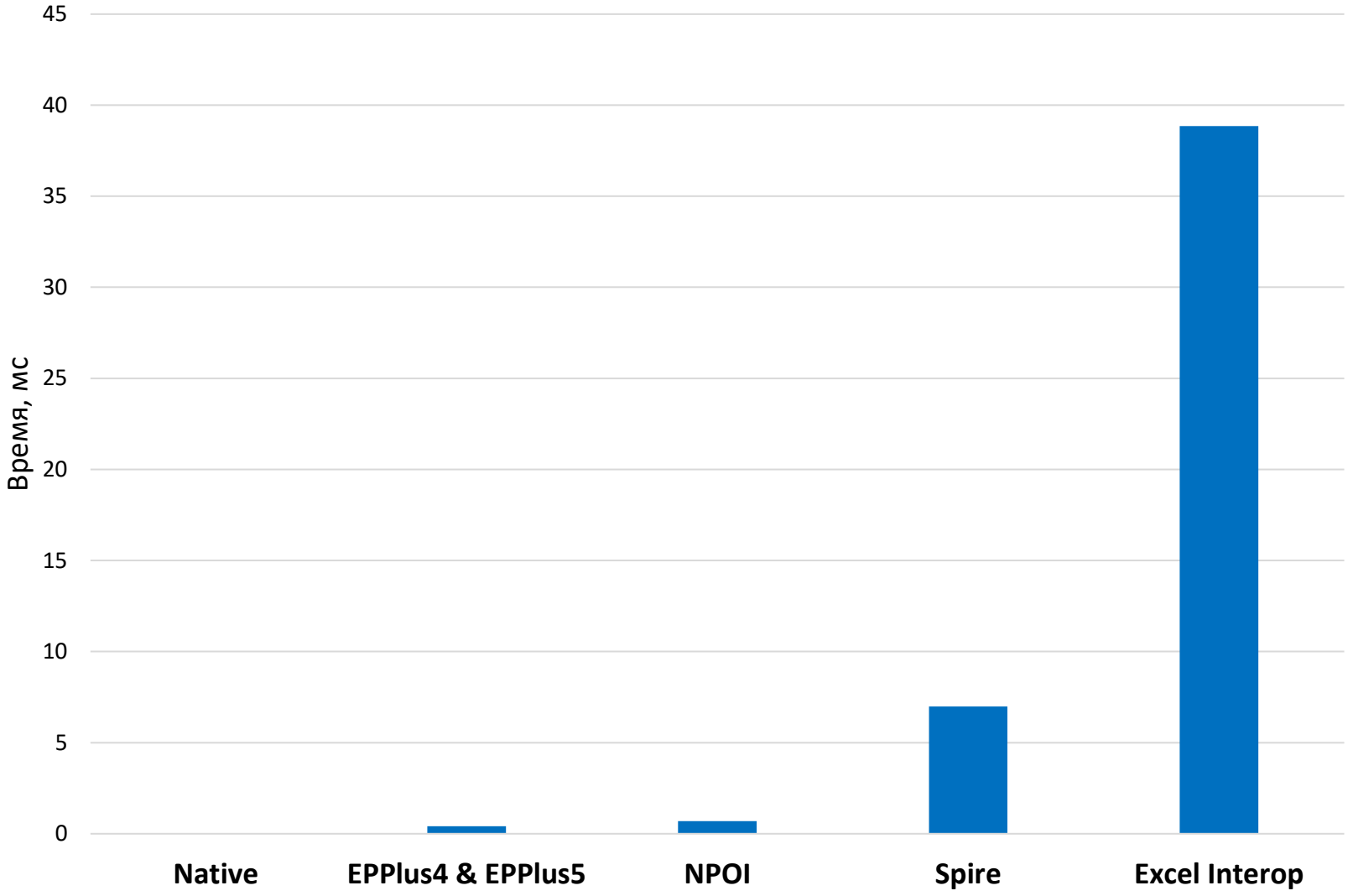
Поиск решения

Шаг 3. Результаты

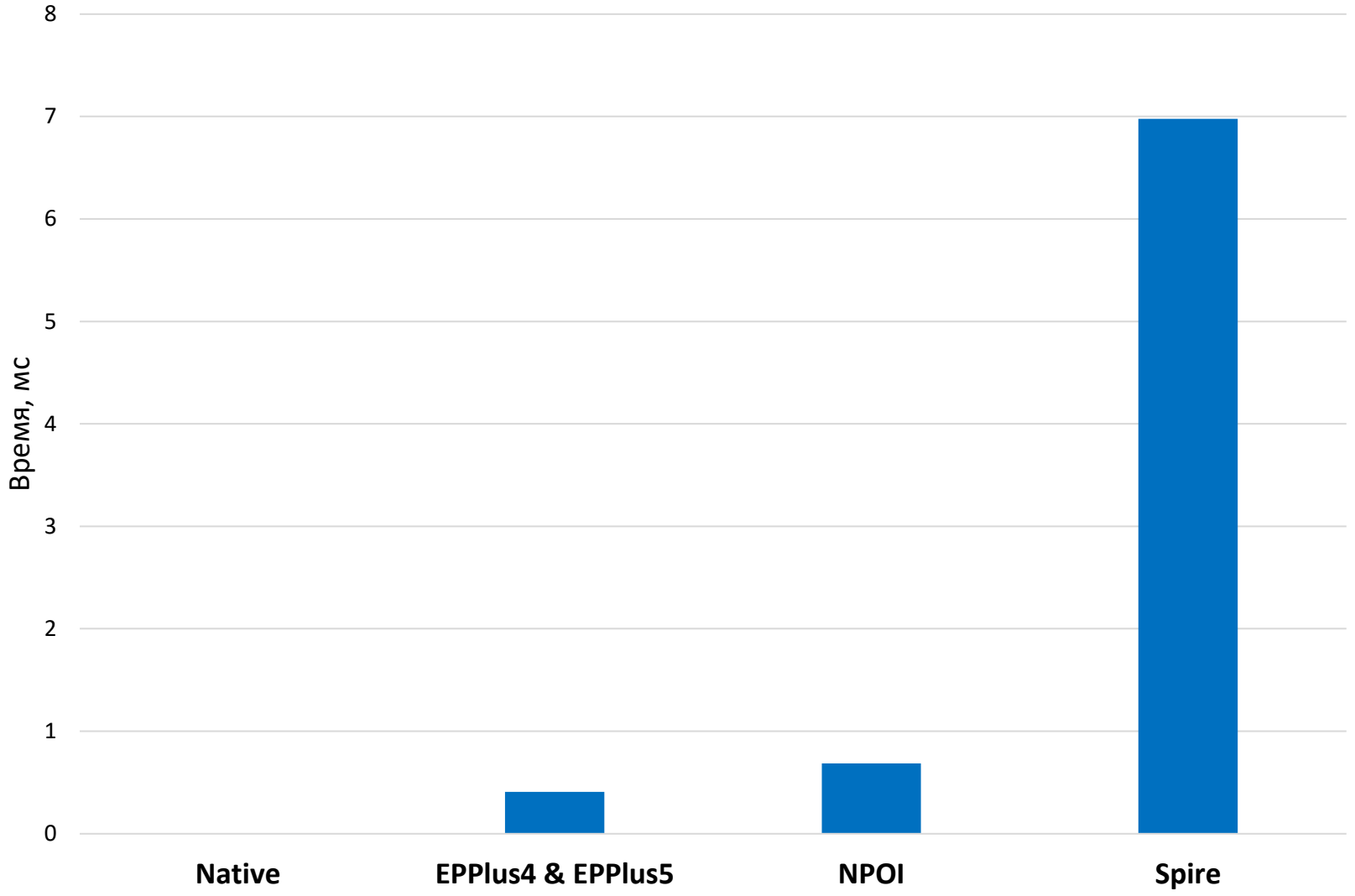
Стенд тестирования

- Intel Core i7-2600 3,4 GHz
- RAM 16 GB 1333 MHz
- SSD

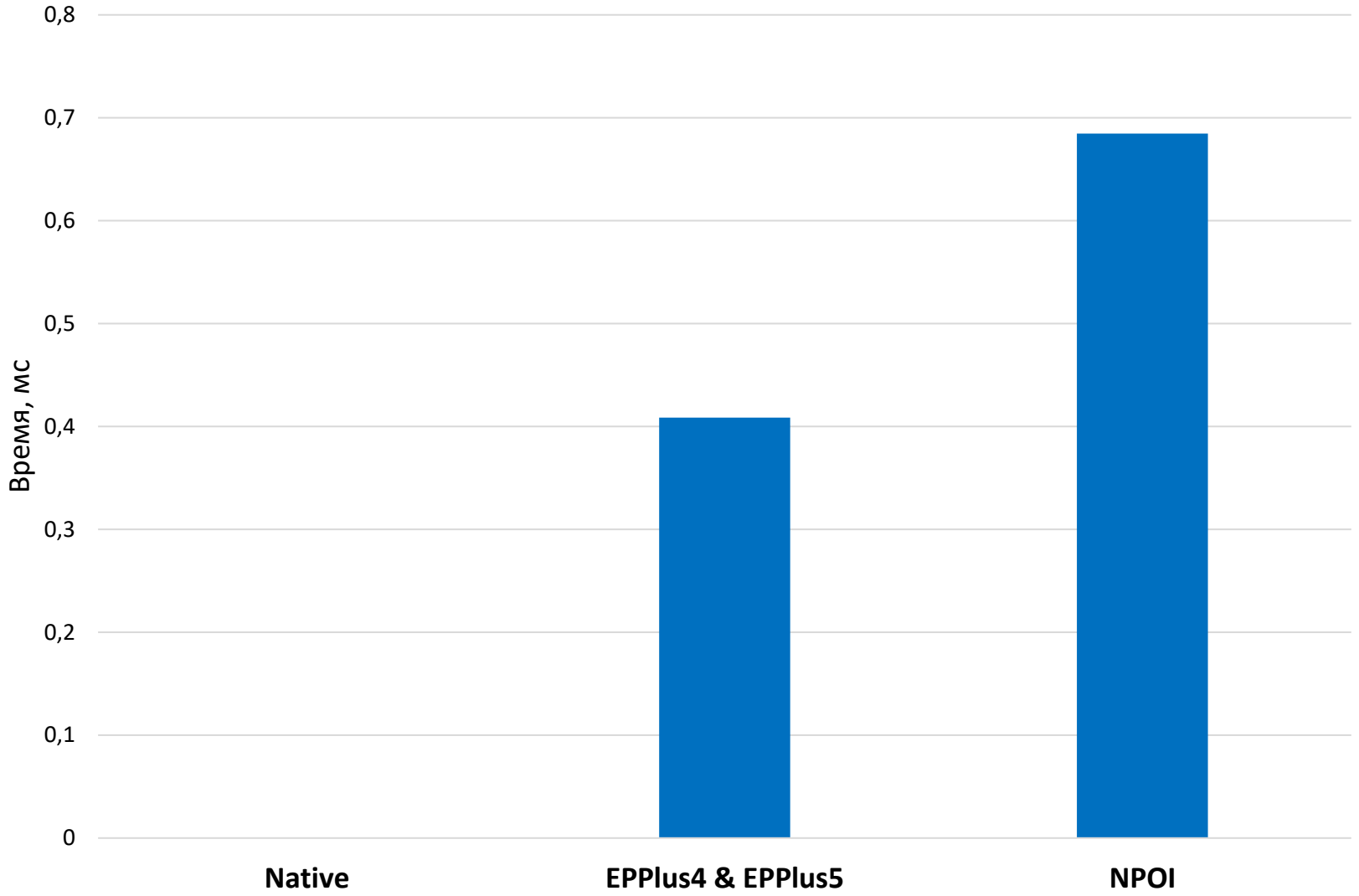
Итоги: Ср. время на 1 проход



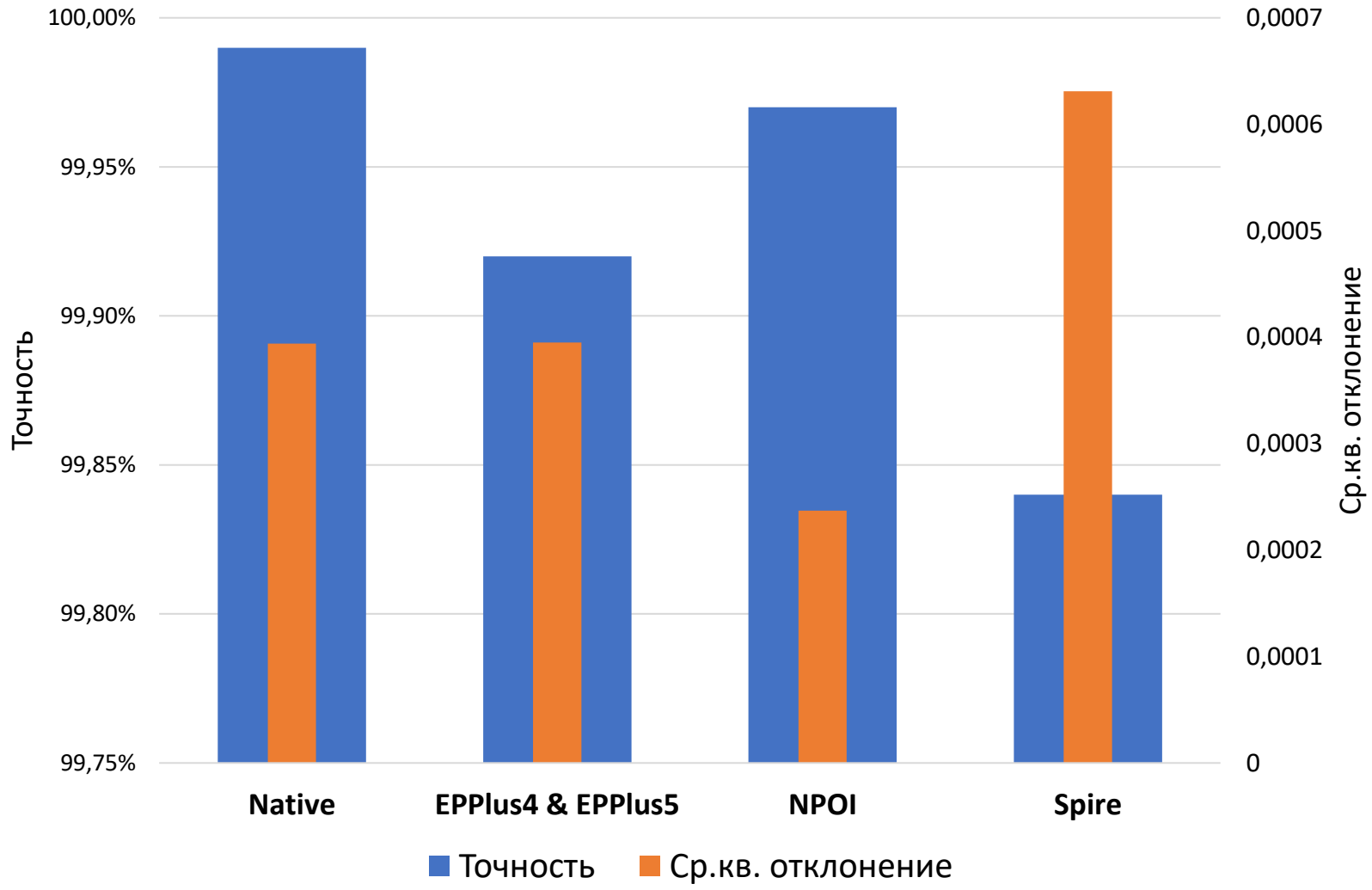
Итоги: Ср. время на 1 проход



Итоги: Ср. время на 1 проход



Итоги: Точность

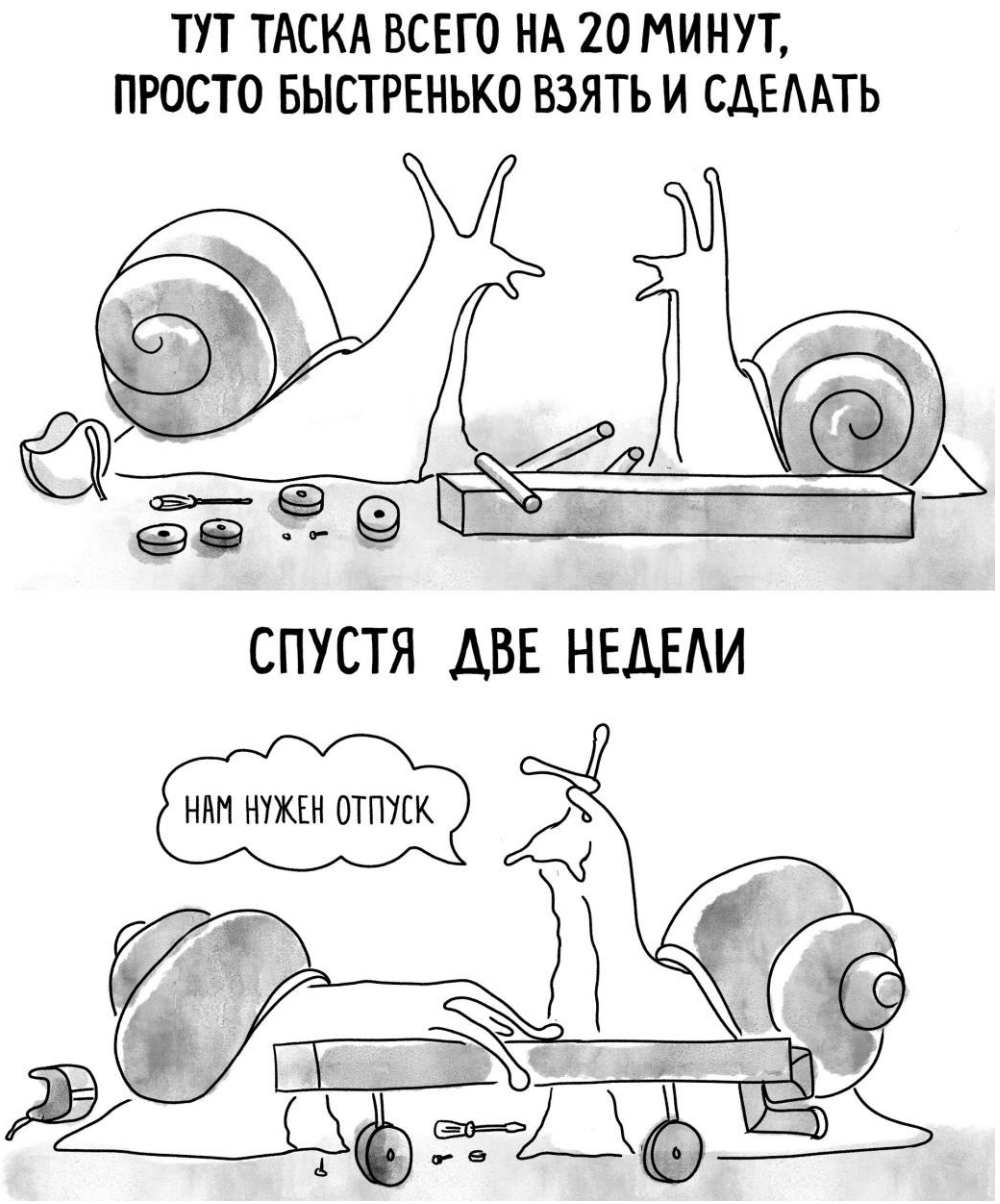


Итоги: Задача оптимизации параметров

Показатель	Native	EPPlus4 & EPPlus5	NPOI	Spire	Excel Interop
Кол-во параметров	2 475 000	1 211	722	70	12

Ошибка в тестовых данных

- Только 1 ячейка с зависимостью от констант



Обновляем тест

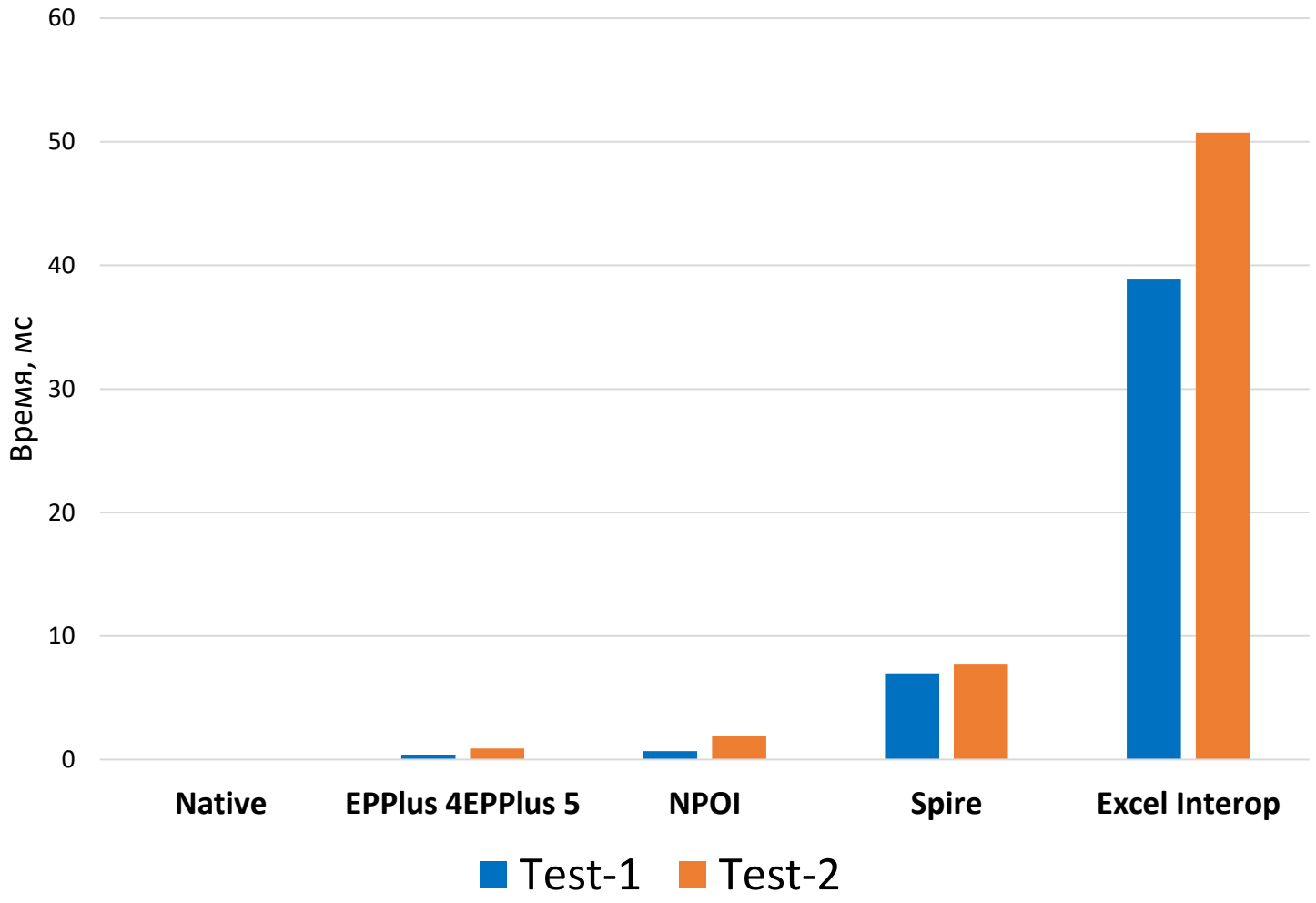
SUM

=AVERAGE(B12:B17)/MAX(1,ABS(B12+B13+B14+B15+B16+B17))

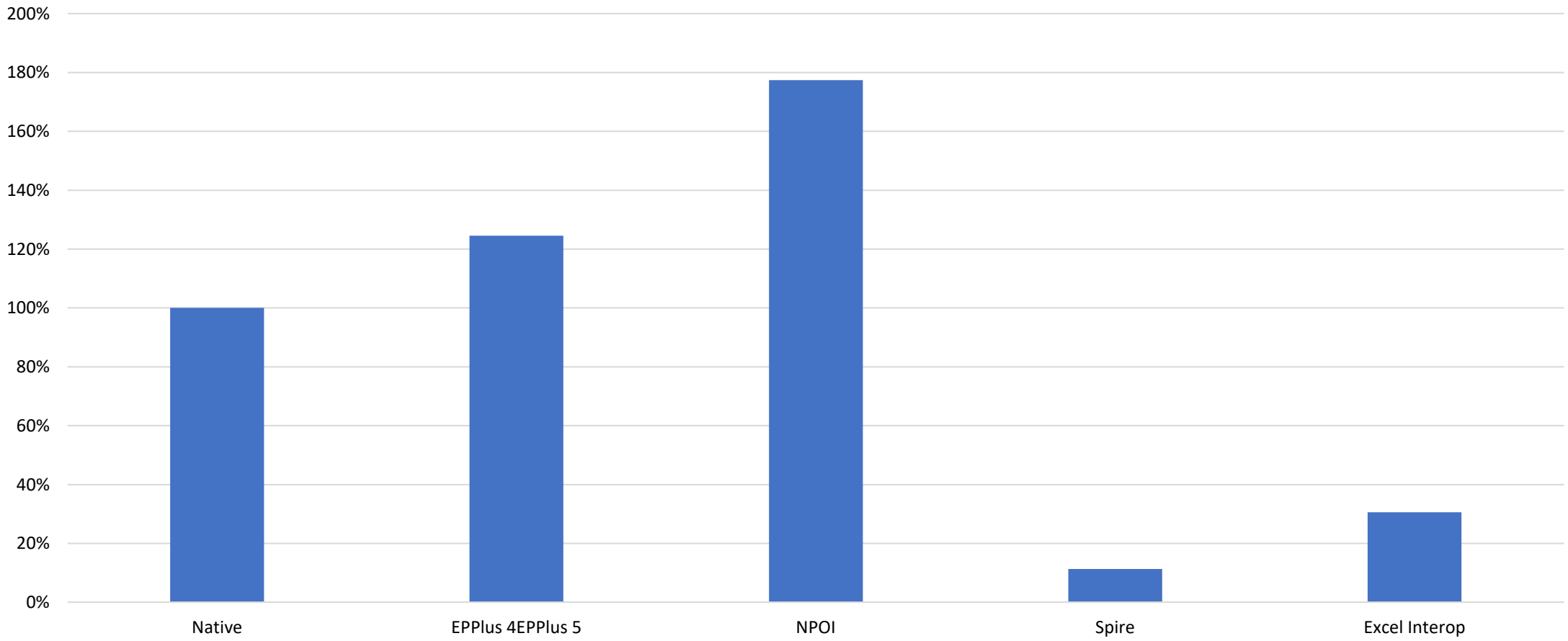
	A	B
1	P1	1
2	P2	2
3	P3	3
4	P4	4
5	P5	5
6	P6	6
7	P7	7
8	P8	8
9	P9	9
10	P10	10
11	Total	B17))
12	Pre 1	3.10465E+21
13	Pre 2	11.25
14	Pre 3	12
15	Pre 4	8.306623863
16	Pre 5	2.311745444
17	Pre 6	5.593220339
18		

```
10      public double Execute(double[] p)
11      {
12          var price1 = Math.Pow(p[0] * p[8] / p[4] * Math.Sin(p[5]) * Math.Cos(p[2]) +
13                                Math.Abs(p[1] - (p[2] + p[3] + p[4] + p[5] + p[6] + p[7] + p[8]))
14                                * Math.Sqrt(p[0] * p[0] + p[1] * p[1]) / 2.0 * Math.PI, p[9]);
15
16          var price2 = p[4] * p[5] * p[2] / Math.Max(1, Math.Abs(p[7]));
17
18          var price3 = Math.Abs(p[7] - p[3]) * p[2];
19
20          var price4 = Math.Sqrt(Math.Abs(p[1] * p[2] + p[3] * p[4] + p[5] * p[6]) + 1.0);
21
22          var price5 = p[0] * Math.Cos(p[1]) + p[2] * Math.Sin(p[1]);
23
24          var sum = p[0] + p[1] + p[2] + p[3] + p[4] + p[5] + p[6] + p[7] + p[8] + p[9];
25
26          var price6 = sum / Math.Max(1, Math.Abs((p[0] + p[1] + p[2] + p[3]) / 4.0))
27                       + sum / Math.Max(1, Math.Abs((p[4] + p[5] + p[6] + p[7] + p[8] + p[9]) / 6.0));
28
29          var pricingAverage = (price1 + price2 + price3 + price4 + price5 + price6) / 6.0;
30
31          return pricingAverage / Math.Max(1, Math.Abs(price1 + price2 + price3 + price4 + price5 + price6));
```

Итоги 2.0: Ср. время на 1 проход



Итоги 2.0: Снижение производительности



Итоги 2.0: Задача оптимизации параметров

Показатель	Native	EPPlus4 & EPPlus5	NPOI	Spire	Excel Interop
Кол-во параметров	1 237 500	539	260	63	9

Кодогенерация

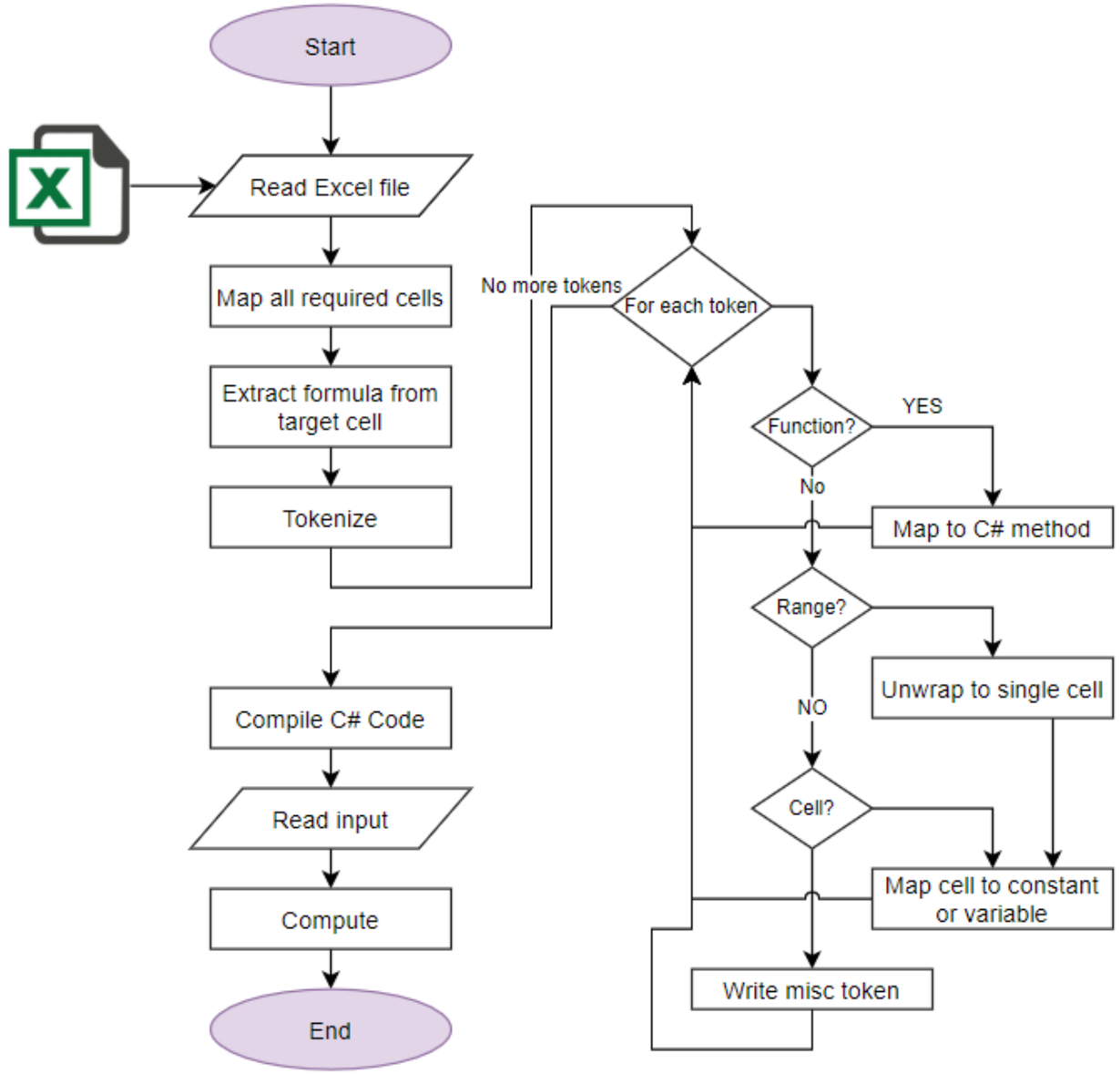
Кодогенерация?

Кодогенерация — способ решения задачи с помощью динамического формирования исходного кода и последующего его использования

- Статическая – генерация во время билд-процесса или ранее
- Динамическая – генерация и исполнение кода в рантайме

Алгоритм

- Прочитать формулу
- Выяснить зависимости
- Транслировать формулу в C# аналог
- Сгруппировать код в функцию и сборку
- Скомпилировать
- Загрузить в память и использовать новую функцию

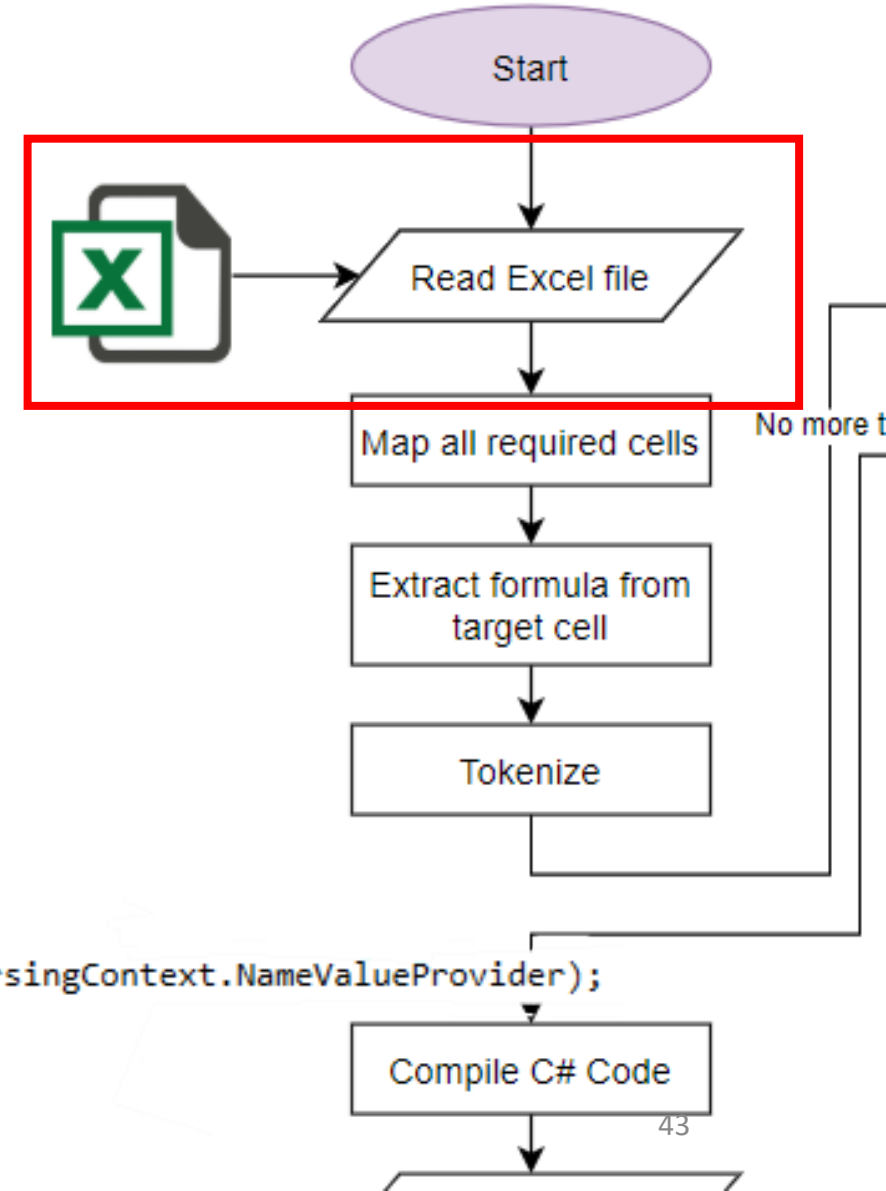


Читаем формулу

- EPPlus как основа
  - Содержит код отвечающий за формирование списка токенов и AST дерева
- EPPlus AST имеет много приватных данных
- Excel Формула – функциональная парадигма
- Выход - Чтение токенов

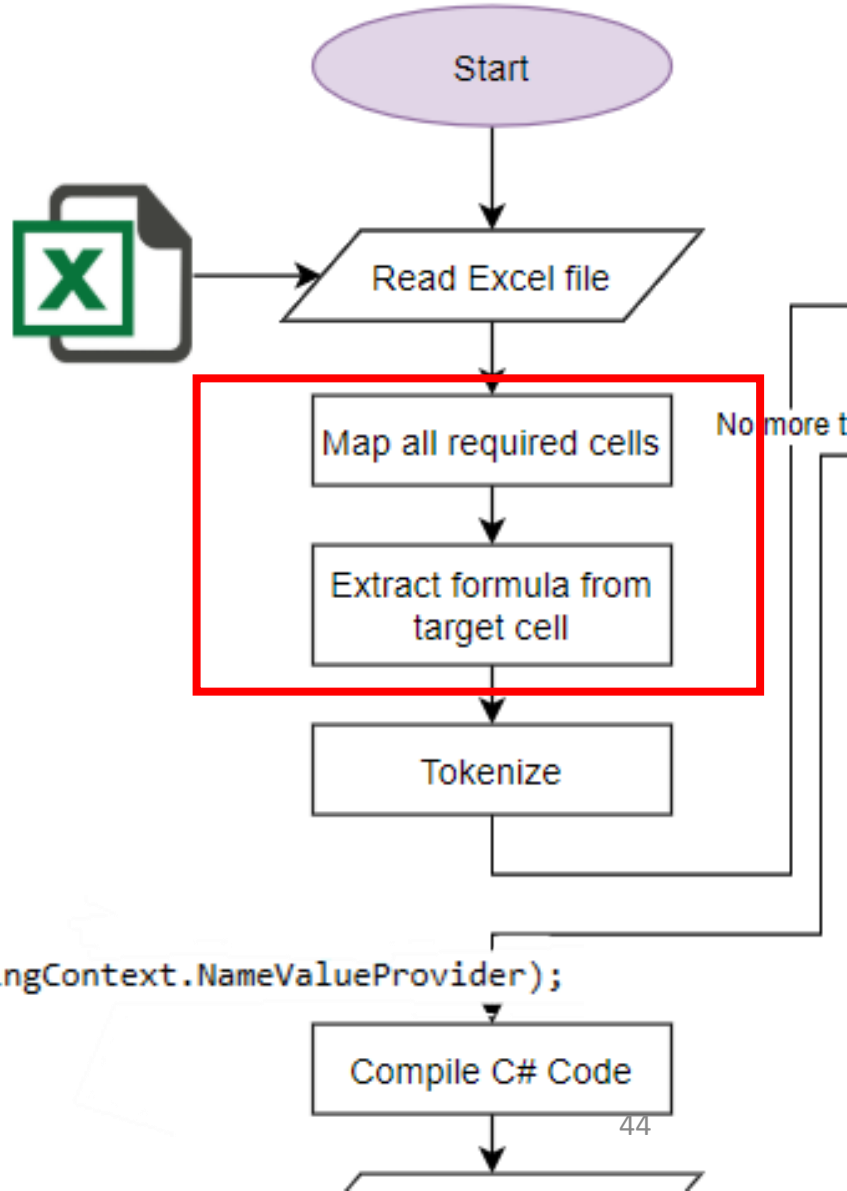
Читаем формулу

```
132     // Initialize excel package by EPPlus library
133     _package = new ExcelPackage(new FileInfo(_fileName));
134     _workbook = _package.Workbook;
135     _worksheet = _workbook.Worksheets[1];
136
137     _inputRange = new ExcelRange[10];
138     for (int rowIndex = 0; rowIndex < 10; rowIndex++)
139     {
140         _inputRange[rowIndex] = _worksheet.Cells[rowIndex + 1, 2];
141     }
142
143     // Access to result cell and extract formula string
144     _resultRange = _worksheet.Cells[11, 2];
145
146     var formula = _resultRange.Formula;
147
148     // Initialize parsing context and setups data provider
149     var parsingContext = ParsingContext.Create();
150     parsingContext.ExcelDataProvider = new EpplusExcelDataProvider(_package);

179     // Initialize basic compile components, e.g. lexer
180     var lexer = new Lexer(parsingContext.Configuration.FunctionRepository, parsingContext.NameValueProvider);
181
182     using (var scope = parsingContext.Scopes.NewScope(RangeAddress.Empty))
183     {
```



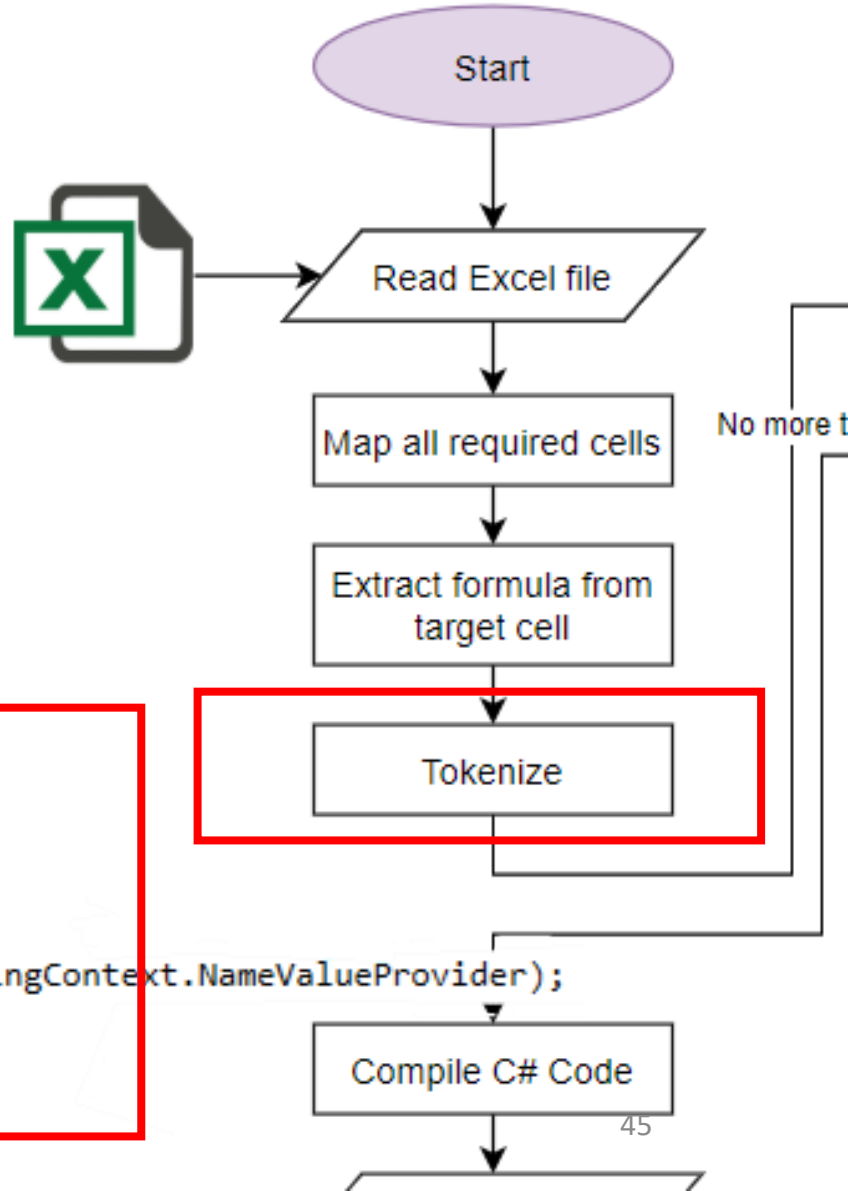
Читаем формулу

```
132     // Initialize excel package by EPPlus library
133     _package = new ExcelPackage(new FileInfo(_fileName));
134     _workbook = _package.Workbook;
135     _worksheet = _workbook.Worksheets[1];
136
137     _inputRange = new ExcelRange[10];
138     for (int rowIndex = 0; rowIndex < 10; rowIndex++)
139     {
140         _inputRange[rowIndex] = _worksheet.Cells[rowIndex + 1, 2];
141     }
142
143     // Access to result cell and extract formula string
144     _resultRange = _worksheet.Cells[11, 2];
145
146     var formula = _resultRange.Formula;
147
148     // Initialize parsing context and setups data provider
149     var parsingContext = ParsingContext.Create();
150     parsingContext.ExcelDataProvider = new EpplusExcelDataProvider(_package);

179     // Initialize basic compile components, e.g. lexer
180     var lexer = new Lexer(parsingContext.Configuration.FunctionRepository, parsingContext.NameValueProvider);
181
182     using (var scope = parsingContext.Scopes.NewScope(RangeAddress.Empty))
183     {
```

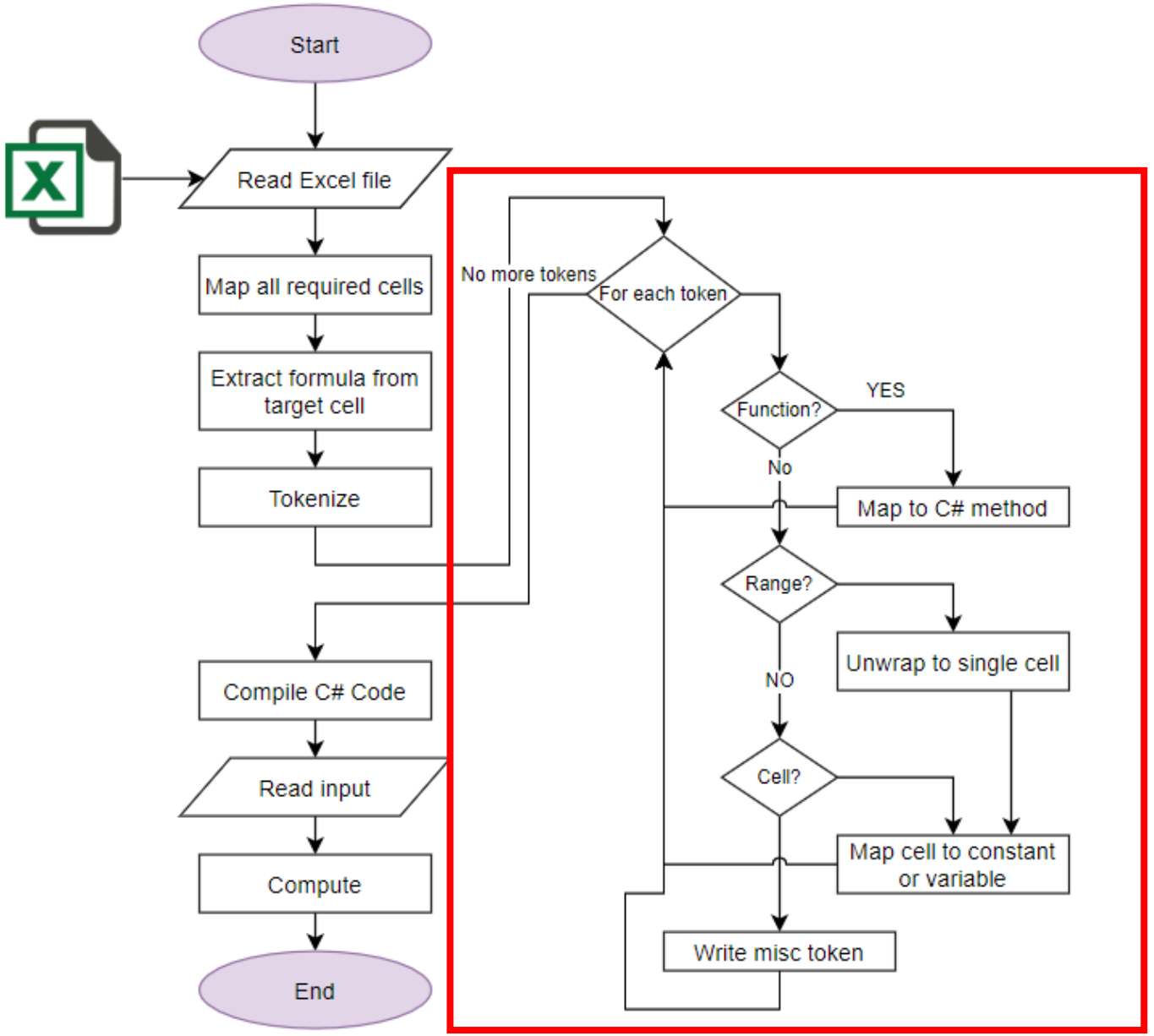


Читаем формулу

```
132     // Initialize excel package by EPPlus library
133     _package = new ExcelPackage(new FileInfo(_fileName));
134     _workbook = _package.Workbook;
135     _worksheet = _workbook.Worksheets[1];
136
137     _inputRange = new ExcelRange[10];
138     for (int rowIndex = 0; rowIndex < 10; rowIndex++)
139     {
140         _inputRange[rowIndex] = _worksheet.Cells[rowIndex + 1, 2];
141     }
142
143     // Access to result cell and extract formula string
144     _resultRange = _worksheet.Cells[11, 2];
145
146     var formula = _resultRange.Formula;
147
148     // Initialize parsing context and setups data provider
149     var parsingContext = ParsingContext.Create();
150     parsingContext.ExcelDataProvider = new EpplusExcelDataProvider(_package);

179     // Initialize basic compile components, e.g. lexer
180     var lexer = new Lexer(parsingContext.Configuration.FunctionRepository, parsingContext.NameValueProvider);
181
182     using (var scope = parsingContext.Scopes.NewScope(RangeAddress.Empty))
183     {
```

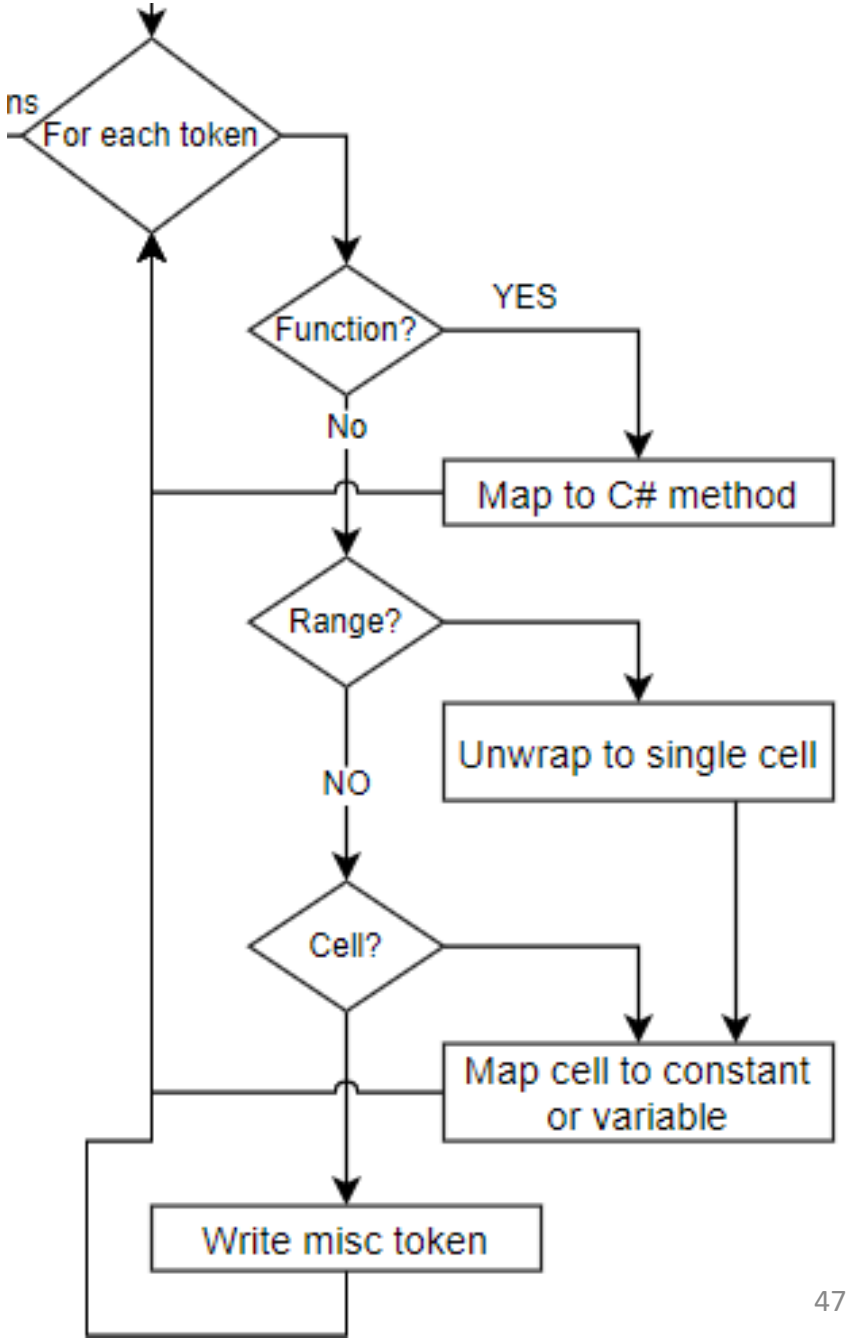


Трансляция


Start
Read Excel file
Map all required cells
Extract formula from target cell
Tokenize
Compile C# Code
Read input
Compute
End
No more tokens
For each token
Function?
YES
No
Map to C# method
Range?
Unwrap to single cell
NO
Cell?
Map cell to constant or variable
Write misc token

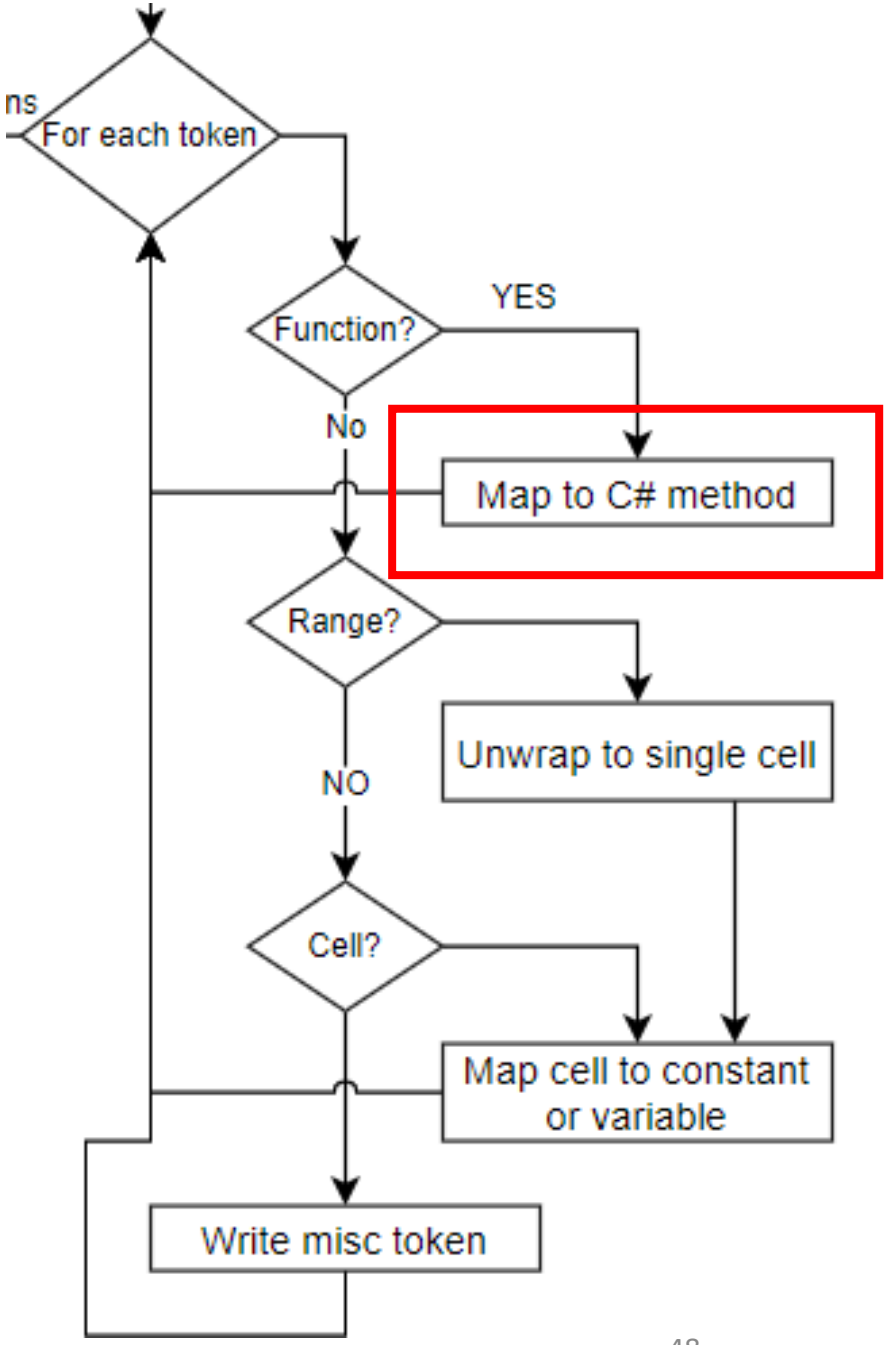
Трансляция

```
245     foreach (Token token in tokens)
246     {
247         switch (token.TokenType)
248         {
249             case TokenType.Function:
250                 output.Append(BuildFunctionName(token.Value));
251                 break;
252             case TokenType.OpeningParenthesis:
253                 output.Append("(");
254                 break;
255             case TokenType.ClosingParenthesis:
256                 output.Append(")");
257                 break;
258             case TokenType.Comma:
259                 output.Append(", ");
260                 break;
261             case TokenType.ExcelAddress:
262                 var address = token.Value;
263                 output.Append(TransformAddressToSharpCode(address
264                 break;
265             case TokenType.Decimal:
266             case TokenType.Integer:
267             case TokenType.Boolean:
268                 output.Append(token.Value);
269                 break;
270             case TokenType.Operator:
271                 output.Append(token.Value);
272                 break;
273         }
274     }
```



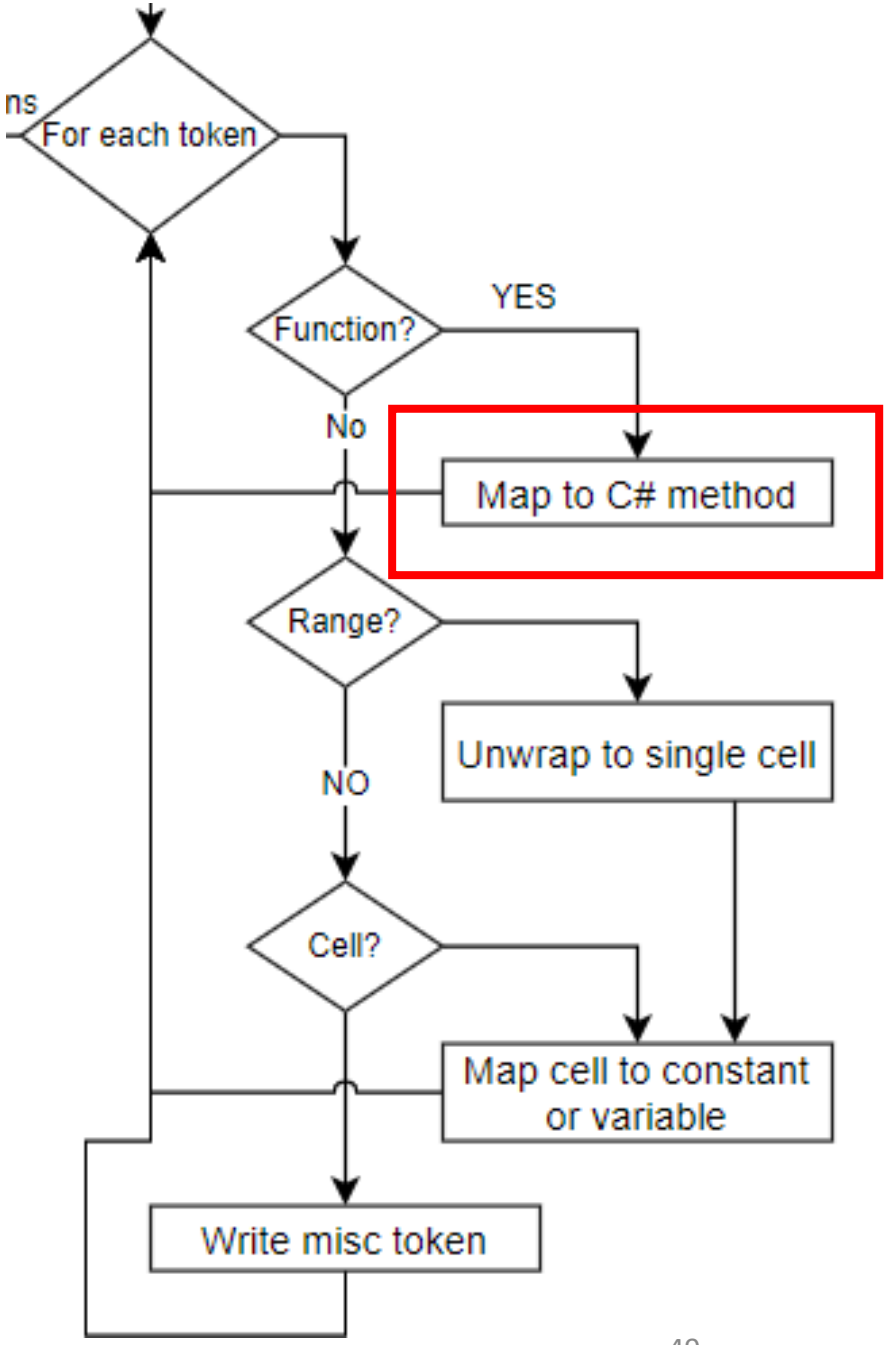
Трансляция функций

```
210     private string BuildFunctionName(string functionName)
211     {
212         if (functionName == "ABS")
213         {
214             return "Math.Abs";
215         }
216
217         if (functionName == "COS")
218         {
219             return "Math.Cos";
220         }
221
222         if (functionName == "SIN")
223         {
224             return "Math.Sin";
225         }
226
227         if (functionName == "SQRT")
228         {
229             return "Math.Sqrt";
230         }
231
232         if (functionName == "POWER")
233         {
234             return "Math.Pow";
235         }
236
237         return $"ExcelCompiledFunctions.{functionName}";
238     }
```



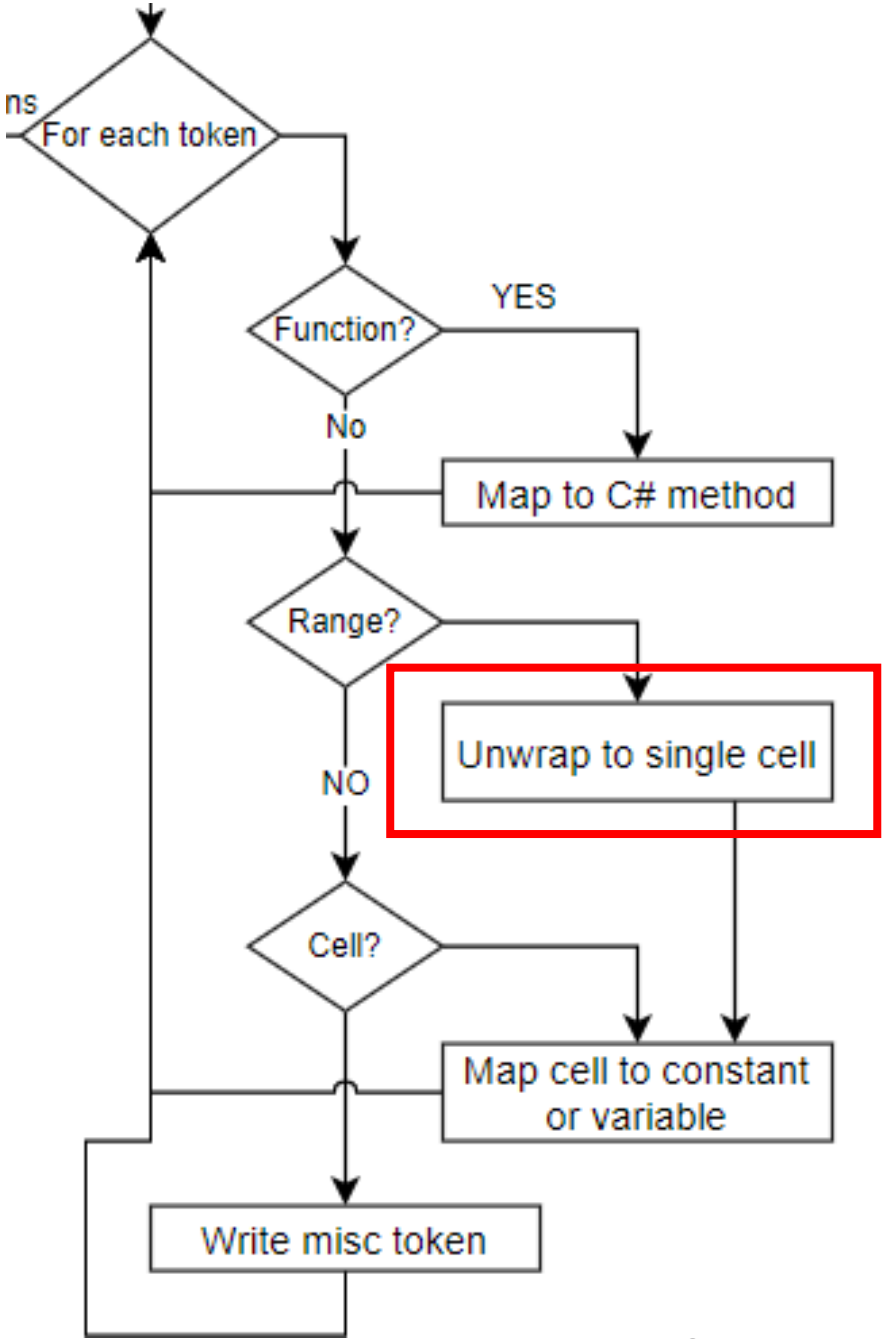
Трансляция функций

```
34      public static double MAX(params double[] args)
35      {
36          double max = double.MinValue;
37          int count = args.Length;
38          double value = 0;
39
40          for (int index = 0; index < count; index++)
41          {
42              value = args[index];
43              max = value > max ? value : max;
44          }
45
46          return max;
47      }
```



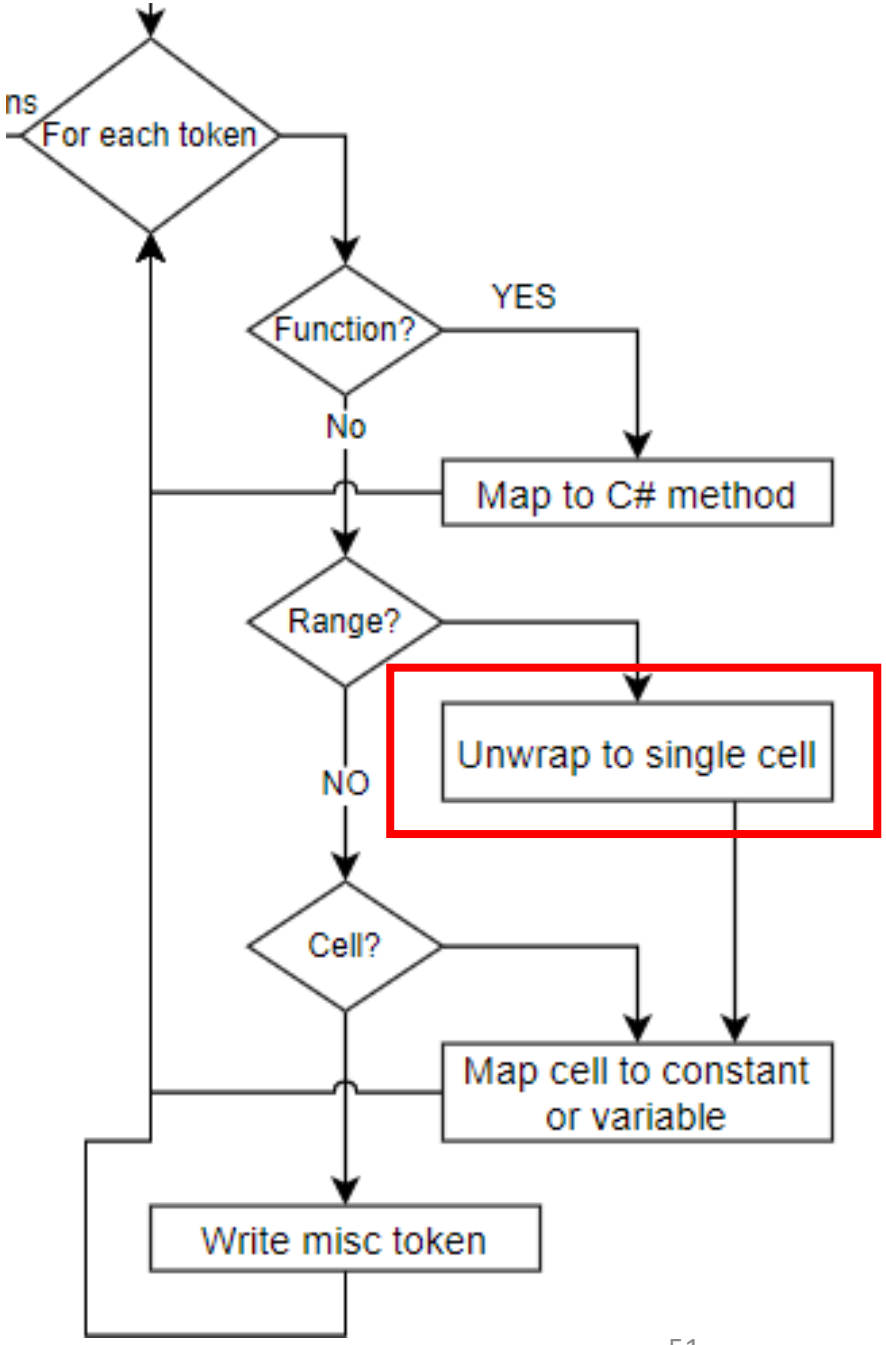
Распаковка

A1:B2 ⇨ A1, A2, B1, B2



Распаковка

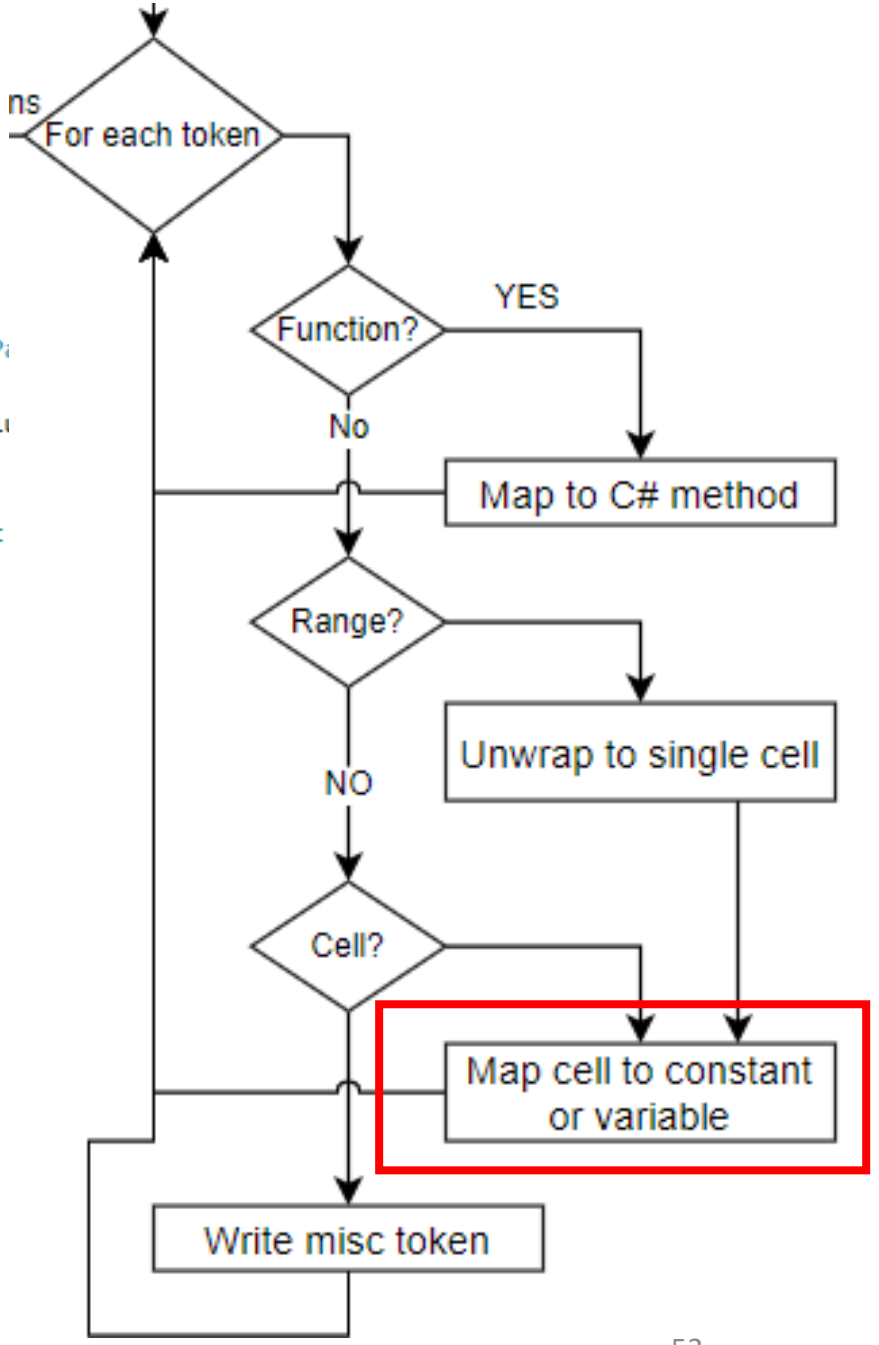
```
282     // Try to parse excel range of addresses
283     // Actually address string is not support reference to another worksheet
284
285     var rangeParts = excelAddress.Split(':');
286     if (rangeParts.Length == 1)
287     {
288         // Unpack single excel address
289         return UnpackExcelAddress(excelAddress, parsingContext, scope, lexer, variables);
290     }
291
292     // Parse excel range address
293     ParseAddressToIndexes(rangeParts[0], out int startRowIndex, out int startColumnIndex);
294     ParseAddressToIndexes(rangeParts[1], out int endRowIndex, out int endColumnIndex);
295
296     var rowDelta = endRowIndex - startRowIndex;
297     var columnDelta = endColumnIndex - startColumnIndex;
298
299     var allAccessors = new List<string>(Math.Abs(rowDelta * columnDelta));
300
301     // TODO This part of code doesn't support reverse-ordered range address
302     for (var rowIndex = startRowIndex; rowIndex <= endRowIndex; rowIndex++)
303     {
304         for (var columnIndex = startColumnIndex; columnIndex <= endColumnIndex; columnIndex+
305         {
306             // Unpack single excel address
307             allAccessors.Add(UnpackExcelAddress(rowIndex, columnIndex, parsingContext, scope,
308         }
309     }
310
311     return string.Join(", ", allAccessors);
```



Трансляция ссылок на ячейку

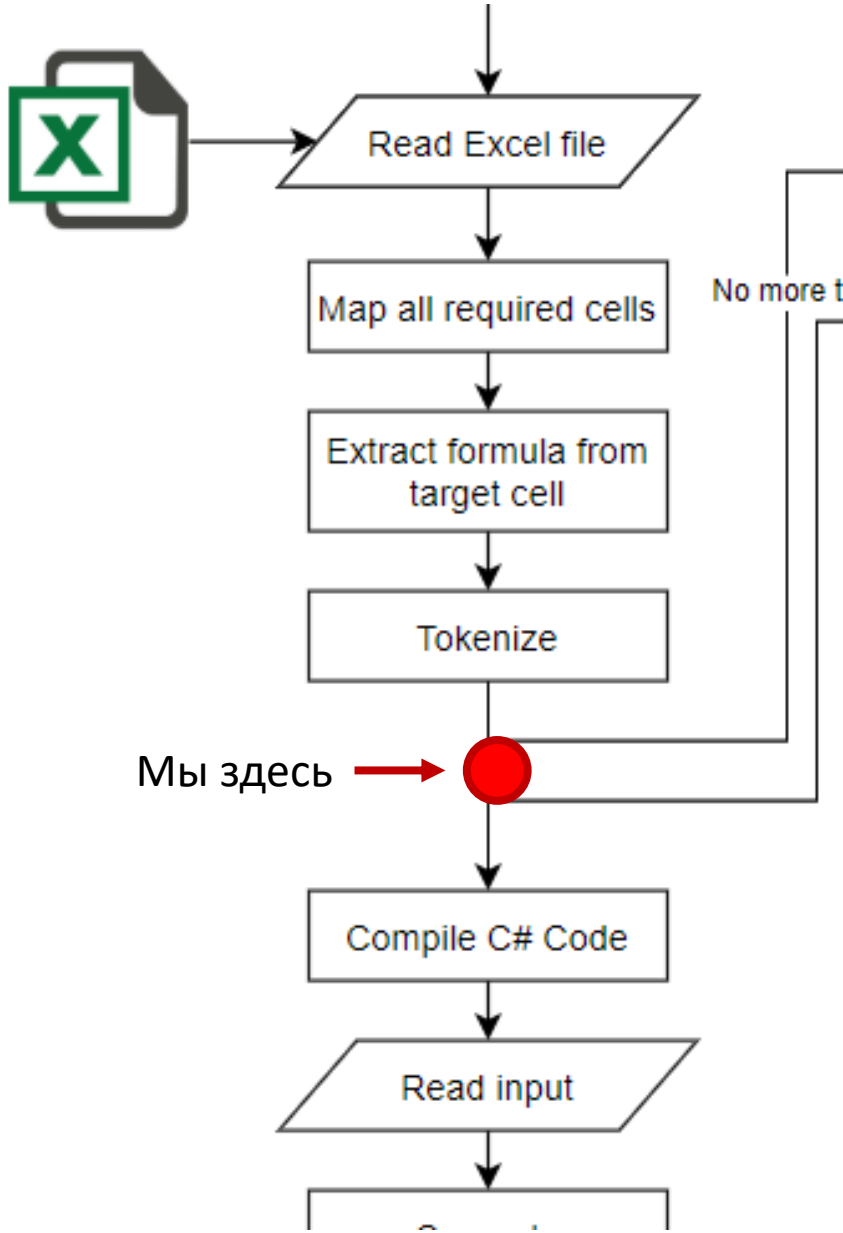
```
319     private string UnpackExcelAddress(int rowIndex, int columnIndex, ParsingContext parsingContext, Pa
320     {
321         var formula = parsingContext.ExcelDataProvider.GetRangeFormula(_worksheet.Name, rowIndex, colu
322         if (string.IsNullOrWhiteSpace(formula))
323         {
324             // When excel address doesn't contains information about any excel formula we should just
325             return $"p[{rowIndex},{columnIndex}]";
326         }
```

```
77      /// <summary>
78      /// Interface to providing values in required excel cells.
79      /// </summary>
80      public interface IExcelValueProvider
81      {
82          /// <summary>
83          /// Currently it use index property by excel row and colum
84          /// I propose change it to function and pass additional pa
85          /// </summary>
86          /// <param name="rowIndex"></param>
87          /// <param name="column"></param>
88          /// <returns></returns>
89          double this[int rowIndex, int column] { get; }
90      }
```



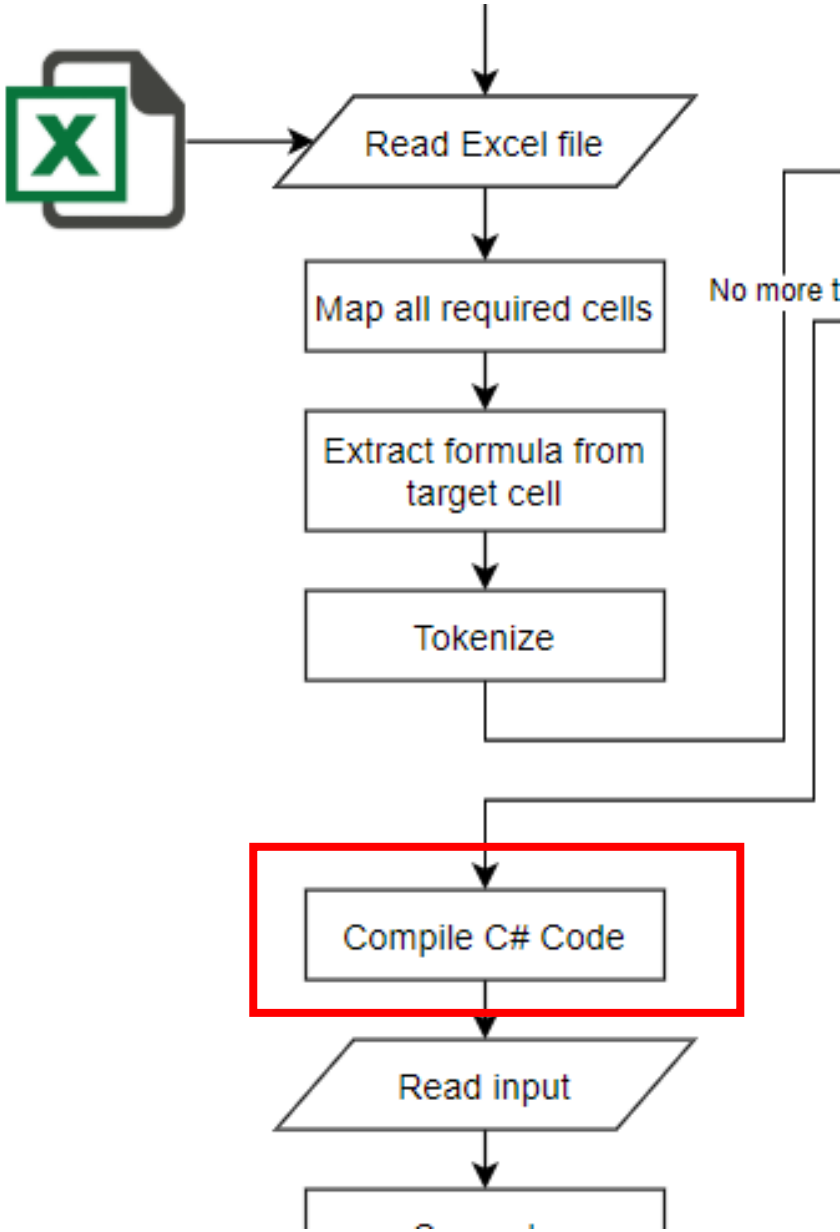
Результат трансляции

```
return Math.Pow(
    p[1, 2] * p[9, 2] / p[5, 2]
    * Math.Sin(p[6, 2])
    * Math.Cos(p[3, 2]) +
        Math.Abs(
            p[2, 2]
             - ExcelCompiledFunctions.SUM(
                p[3, 2], p[4, 2], p[5, 2],
                p[6, 2], p[7, 2], p[8, 2],
                p[9, 2]
        ))
        * Math.Sqrt(
            p[1, 2] * p[1, 2]
            + p[2, 2] * p[2, 2]
        ) / 2
        * ExcelCompiledFunctions.PI(),
    p[10, 2]
);
```



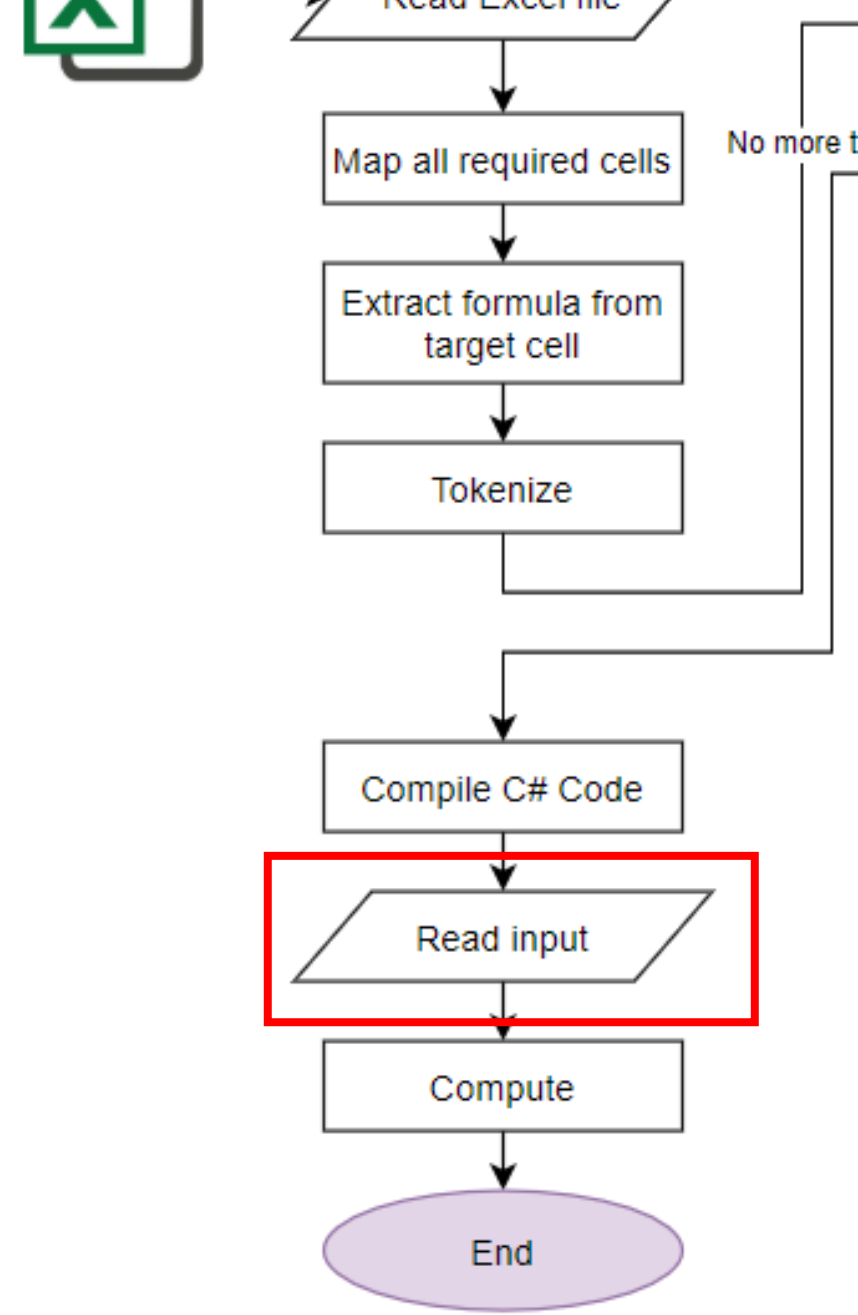
Компиляция

```
150     // Compile CSharp code to IL dynamic assembly via helper wrappers
151     _code = CodeGenerator.CreateCode<double>(
152         sourceCode,
153         new []
154         {
155             // Required for Math functions
156             "System",
157             // Required for Excel function wrappers stored at ExcelCompiledFunctions class
158             "ExcelCalculations.PerformanceTests"
159         },
160         new []
161         {
162             // Add reference to current compiled assembly,
163             // that is required to use Excel function wrappers located
164             // at ExcelCompiledFunctions static class
165             "ExcelCalculations.PerformanceTests.exe"
166         },
167         // Notify that this source code should use parameter;
168         // Use abstract p parameter - interface for values accessing.
169         new CodeParameter("p", typeof(IExcelValueProvider))
170     );
```



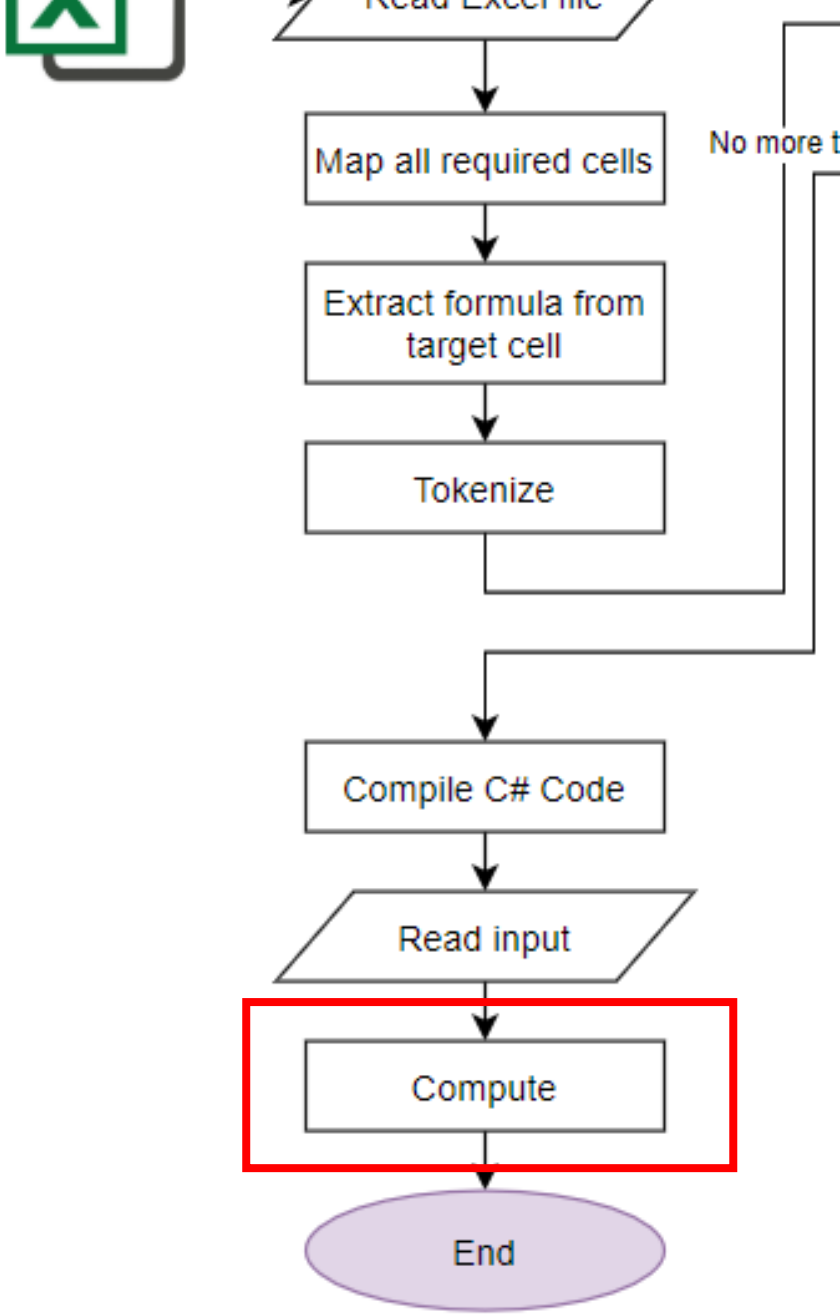
Передача параметров

```
387     // Use another implementation of value provider
388     // Also nested from MarshalByRefObject class
389     // to correct object passing to generated assembly.
390
391     // This instance is required to prevent new instance creation
392     // per each execution time, just replacing reference to input parameters.
393     private class PlainExcelValueProvider : MarshalByRefObject, IExcelValueProvider
394     {
395         public double[] p;
396
397         public PlainExcelValueProvider()
398         {
399
400         }
401
402         public double this[int rowIndex, int columnIndex]
403         {
404             get => p[rowIndex - 1];
405         }
406     }
```



Расчет

```
411     public double Execute(double[] p)
412     {
413         // Just pass a new reference to input parameters collection
414         _valueProvider.p = p;
415
416         // Execute IL generated code with new value provider
417         return _code.Execute(_valueProvider);
418     }
```



Mark-I

Библиотека	Ср. время (мс)	Коэф. замедления
Native	0,00004	1
EPPlusCompiled, Mark-I	0,0061	16
EPPlus	1,2089	3023

Mark-I: Проблема

Исходник:

A1 := B1 * B2 + 5 * B1
B1 := F(B2) + 10
B2 := 15

Результат:

A1 := (F(INPUT[‘b2’]) + 10) * INPUT[‘b2’]
+ 5 * (F(INPUT[‘b2’]) + 10)

B1 := F(INPUT[‘b2’]) + 10
B2 := INPUT[‘b2’]

Надо:

A1 := B1 * B2 + 5 * B1
B1 := F(B2) + 10
B2 := INPUT[‘b2’]

Трансляция

- Добавим использование переменных

```
325 {
326     var formula = parsingContext.ExcelDataProvider.GetRangeFormula(_worksheet.Name, rowIndex, columnIndex);
327     if (string.IsNullOrWhiteSpace(formula))
328     {
329         // When excel address doesn't contains information about any excel formula we should just use external input data parameter provider.
330         return $"p[{rowIndex},{columnIndex}]";
331     }
332
333     // When formula is provided try to identify that variable is not defined yet
334     // TODO Worksheet name is not included in variable name, potentially that can cause conflicts
335     // Extracting and reusing calculations via local variables improves performance for 0.0045ms
336     var cellVariableId = $"C{rowIndex}R{columnIndex}";
337     if (variables.ContainsKey(cellVariableId))
338     {
339         return cellVariableId;
340     }
341
342     // When variable is not exists, transform new formula and register that to variable scope
343     variables.Add(cellVariableId, TransformToSharpCode(formula, parsingContext, scope, lexer, variables));
344
345     return cellVariableId;
346 }
```

Трансляция

- Добавим использование переменных

```
325 {
326     var formula = parsingContext.ExcelDataProvider.GetRangeFormula(_worksheet.Name, rowIndex, columnIndex);
327     if (string.IsNullOrWhiteSpace(formula))
328     {
329         // When excel address doesn't contains information about any excel formula we should just use external input data parameter provider.
330         return $"p[{rowIndex},{columnIndex}]";
331     }
332
333     // When formula is provided try to identify that variable is not defined yet
334     // TODO Worksheet name is not included in variable name, potentially that can cause conflicts
335     // Extracting and reusing calculations via local variables improves performance for 0.0045ms
336     var cellVariableId = $"C{rowIndex}R{columnIndex}";
337     if (variables.ContainsKey(cellVariableId))
338     {
339         return cellVariableId;
340     }
341
342     // When variable is not exists, transform new formula and register that to variable scope
343     variables.Add(cellVariableId, TransformToSharpCode(formula, parsingContext, scope, lexer, variables));
344
345     return cellVariableId;
346 }
```

Результат трансляции

```
var C17R2 = ExcelCompiledFunctions.SUM(p[1, 2], p[2, 2], p[3, 2], p[4, 2], p[5, 2], p[6, 2], p[7, 2], p[8, 2], p[9, 2], p[10, 2])
          / ExcelCompiledFunctions.MAX(1, Math.Abs(ExcelCompiledFunctions.AVERAGE(p[1, 2], p[2, 2], p[3, 2], p[4, 2]))
                                        + ExcelCompiledFunctions.SUM(p[1, 2], p[2, 2], p[3, 2], p[4, 2], p[5, 2], p[6, 2], p[7, 2], p[8, 2], p[9, 2], p[10, 2])
                                        / ExcelCompiledFunctions.MAX(1, ExcelCompiledFunctions.AVERAGE(p[5, 2], p[6, 2], p[7, 2], p[8, 2], p[9, 2], p[10, 2])));
var C16R2 = p[1, 2] * Math.Cos(p[2, 2]) + p[3, 2] * Math.Sin(p[2, 2]);
var C15R2 = Math.Sqrt(Math.Abs(p[2, 2] * p[3, 2] + p[4, 2] * p[5, 2] + p[6, 2] * p[7, 2]) + 1);
var C14R2 = Math.Abs(p[8, 2] - p[4, 2]) * p[3, 2];
var C13R2 = p[5, 2] * p[6, 2] * p[3, 2] / ExcelCompiledFunctions.MAX(1, Math.Abs(p[8, 2]));
var C12R2 = Math.Pow(p[1, 2] * p[9, 2] / p[5, 2]
                     * Math.Sin(p[6, 2])
                     * Math.Cos(p[3, 2])
                     + Math.Abs(p[2, 2] - ExcelCompiledFunctions.SUM(p[3, 2], p[4, 2], p[5, 2], p[6, 2], p[7, 2], p[8, 2], p[9, 2]))
                     * Math.Sqrt(p[1, 2] * p[1, 2] + p[2, 2] * p[2, 2]) / 2 * ExcelCompiledFunctions.PI(), p[10, 2]);
return ExcelCompiledFunctions.AVERAGE(C12R2, C13R2, C14R2, C15R2, C16R2, C17R2)
       / ExcelCompiledFunctions.MAX(1, Math.Abs(C12R2 + C13R2 + C14R2 + C15R2 + C16R2 + C17R2));
```

Mark-II

Библиотека	Ср. время (мс)	Коэф. замедления
Native	0,0004	1
EPPlusCompiled, Mark-II	0,003	8
EPPlusCompiled, Mark-I	0,0061	16
EPPlus	1,2089	3023

Mark-III?

- Замена `public static double PI()` на прямую ссылку на константу дает незначительный прирост производительности

```
public static double SUM(params double[] args)
{
    double sum = 0;
    int count = args.Length;
    for (int index = 0; index < count; index++)
    {
        sum += args[index];
    }

    return sum;
}
```

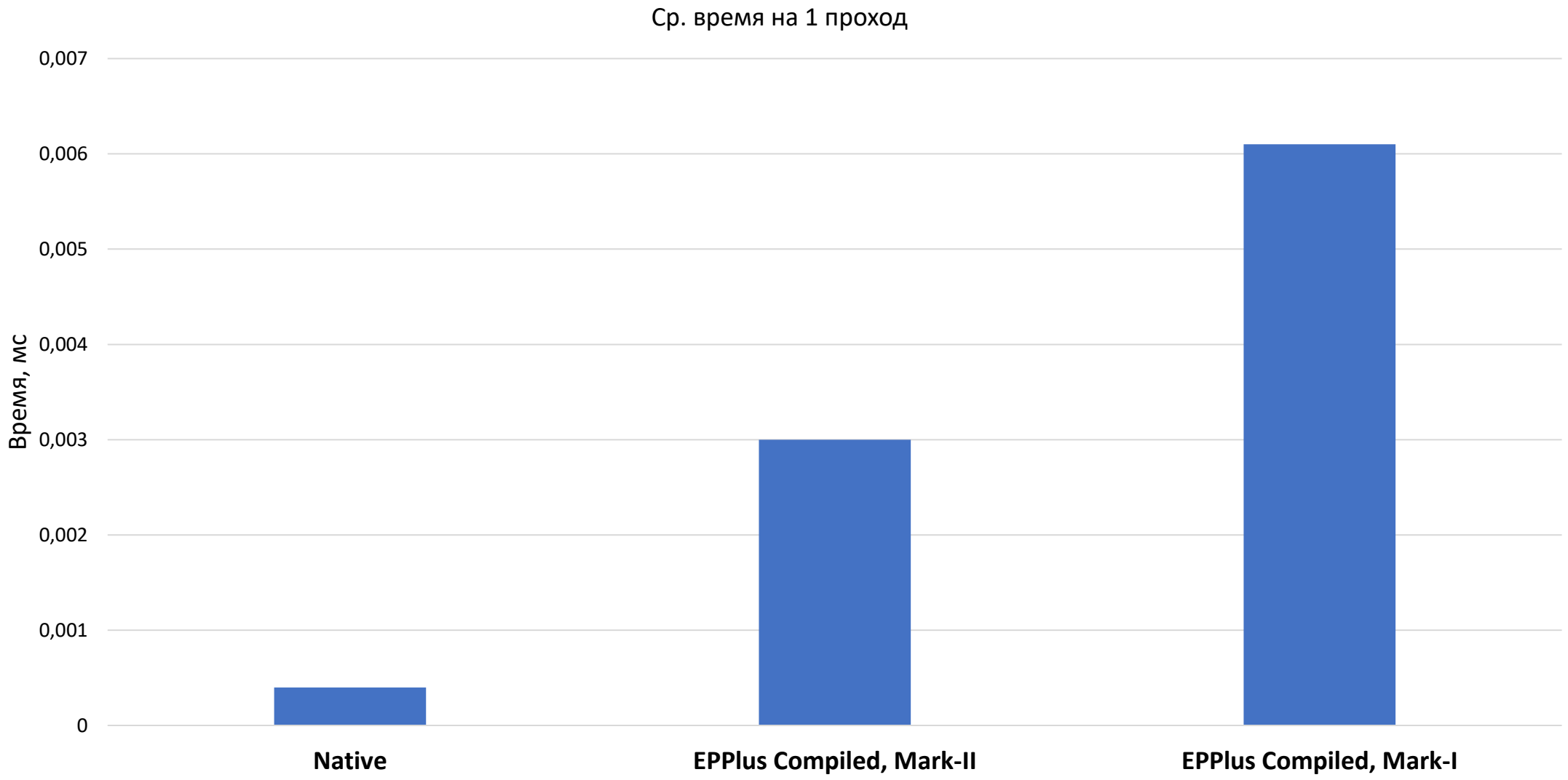
```
p[0, 0] + p[1, 0] + p[2, 0];
```

- Замена функций суммирования на inline методы без циклов дает незначительный прирост производительности
- Оптимизация дерева вычислений дает значительное улучшение производительности

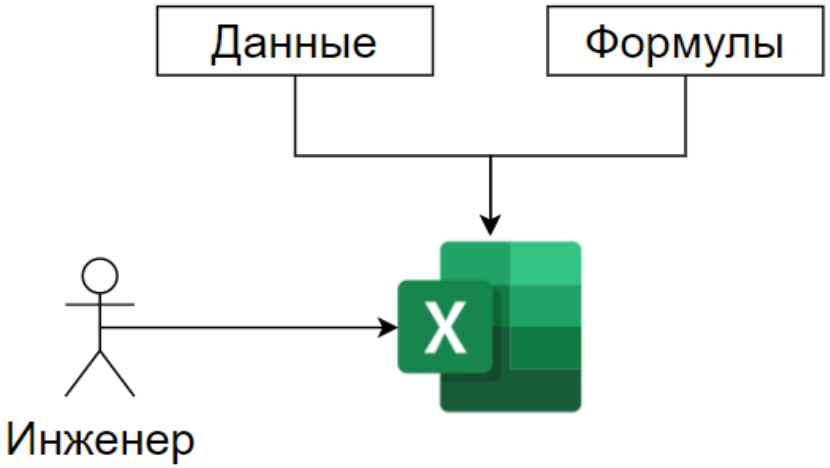
Результат


Ср. время на 1 проход
Время, мс
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
Native
EPPlus Compiled, Mark-II
EPPlus Compiled, Mark-I
EPPlus 4EPPlus 5

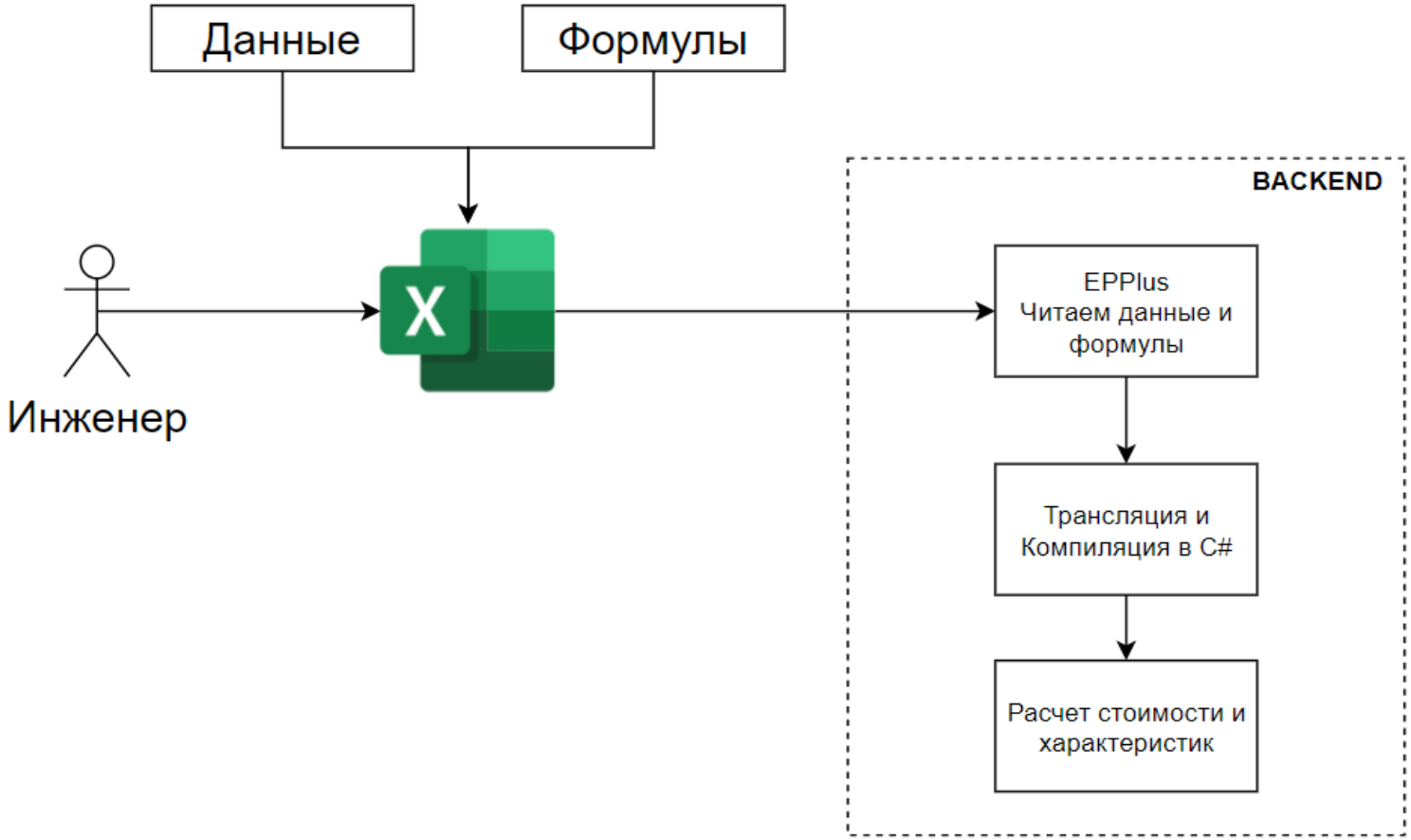
Результат


Ср. время на 1 проход
Время, мс
0,007
0,006
0,005
0,004
0,003
0,002
0,001
0
Native
EPPlus Compiled, Mark-II
EPPlus Compiled, Mark-I

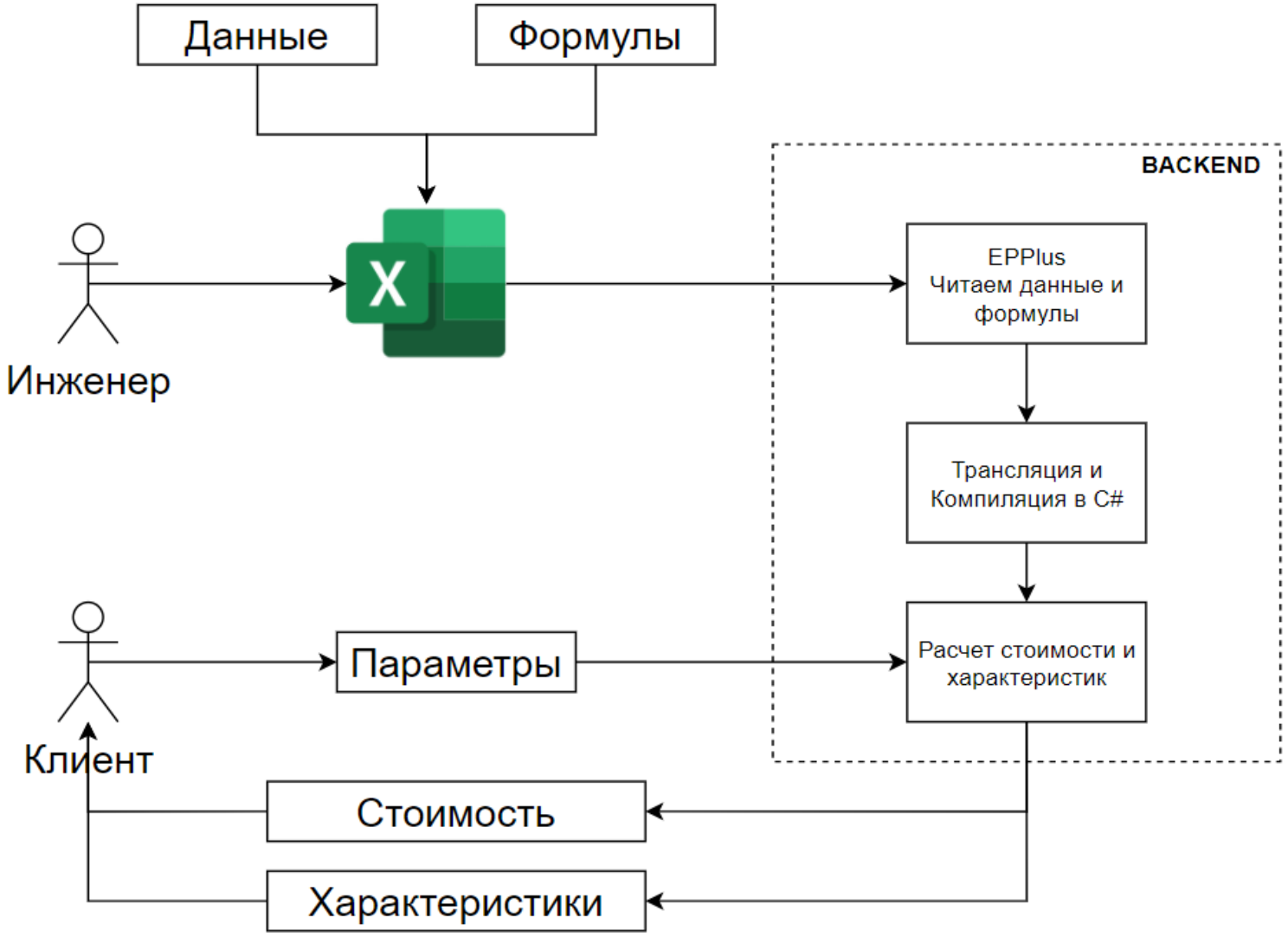
Результат



Результат


Данные
Формулы
Инженер
BACKEND
EPPlus
Читаем данные и формулы
Трансляция и Компиляция в C#
Расчет стоимости и характеристик

Результат


Данные
Формулы
BACKEND
Инженер
EPPlus
Читаем данные и формулы
Трансляция и Компиляция в C#
Клиент
Параметры
Расчет стоимости и характеристик
Стоимость
Характеристики

Результат

- Интерфейс управления не изменился
  - Сохранили время на разработку интерфейса редактора
- Быстрые расчеты данных
  - Производительность уровня ручного переноса
  - Возможно выполнение задачи оптимизации параметров
- Ограничения:
  - Поддерживаются не все форматы ссылок
  - Не полная поддержка функций
  - Работа только с одной страницей

Хабр: https://habr.com/ru/company/arcadia/blog/498032/