МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

Факультет информационных технологий и программирования

КУРСОВАЯ РАБОТА

Тема: Анализ и моделирование программной архитектуры и архитектуры данных программного средства .NET Compiler Platform "Roslyn"

Работу выполнил студент: \_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_ группы \_\_\_

(фамилия, имя, отчество) (номер группы)

Руководитель *Маятин Александр Владимирович* \_\_\_

(фамилия, имя, отчество)

Работа защищена " " 201 г. с оценкой \_\_\_

Подписи членов комиссии: \_

\_

САНКТ-ПЕТЕРБУРГ

2018

# Содержание

Оглавление

[Содержание 2](#_Toc535665448)

[Теоретический этап 3](#_Toc535665449)

[Анализ и моделирование процессов 4](#_Toc535665450)

[Функциональные требования 4](#_Toc535665451)

[Нефункциональные требования 5](#_Toc535665452)

[Анализ средств автоматизации процессов 6](#_Toc535665453)

[Dependencies 6](#_Toc535665454)

[Compilers 7](#_Toc535665455)

[Features 9](#_Toc535665456)

[Workspaces 10](#_Toc535665457)

[Scripting 12](#_Toc535665458)

[EditorFeatures 13](#_Toc535665459)

[VisualStudio 14](#_Toc535665460)

[Interactive 15](#_Toc535665461)

[ExpressionEvaluator 17](#_Toc535665462)

[Синтез определенных уровней архитектуры ИС 19](#_Toc535665463)

[Вывод 25](#_Toc535665464)

[Список использованной литературы 26](#_Toc535665465)

# Теоретический этап

**.NET** – программная платформа для разработки программ предназначенная для языков C# и Visual Basic.

**Visual Basic (VB)** – язык программирования.

**Visual Studio** - интегрированная среда разработки (IDE) предоставляемая Microsoft. Данная IDE разрабатывает во многом для фреймворка \*\*.NET\*\* и его языков (C++/CLI, C#, F#), но также имеет поддержку и тулзы для работы с SQL, Python, Javascript/Typescript, Java etc.

**API** - программный интерфейс приложения; описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

**Код-стайл** - набор правил и соглашений, используемых при написании исходного кода на некотором языке программирования.

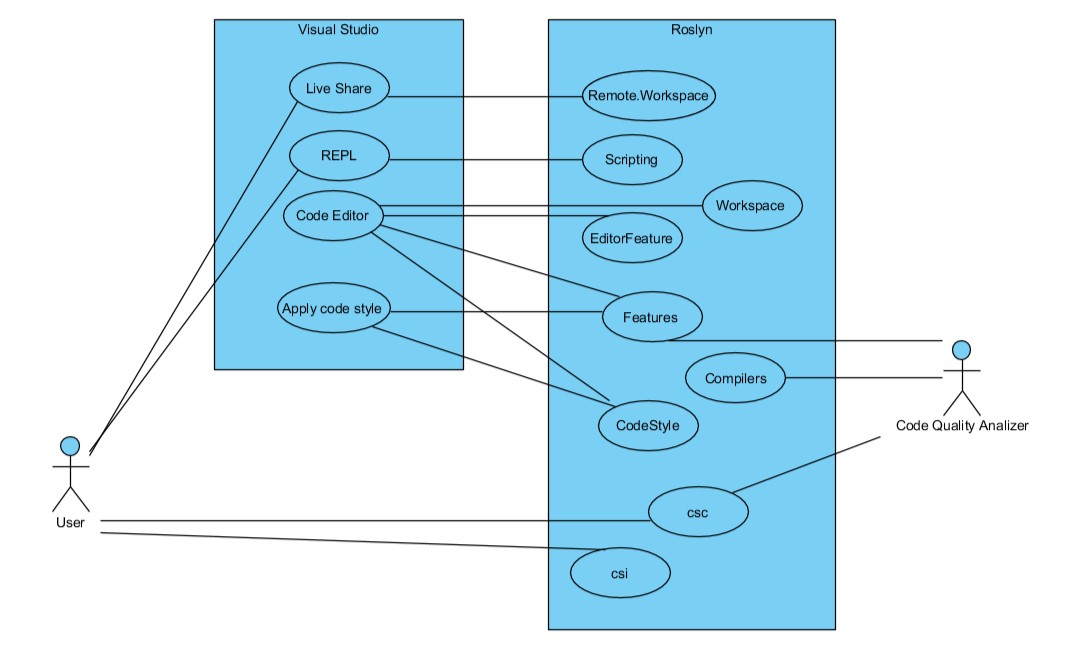
**Язык программирования** - формальный язык, предназначенный для записи компьютерных программ

**Компиляция** – трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду

# Анализ и моделирование процессов

**Roslyn** - компилятор исходного кода .NET, который предоставляется в рамках CaaP (Compiler-as-a-Platform). Является проектом Microsoft с открытым исходным кодом. Поддерживаемыми языками являются C# и VB, который используют .NET в качестве среды выполнения. Основная идея проекта Roslyn заключается в раскрытии процесса компиляции, предоставление доступа пользователям к взаимодействию с различными его этапами.

Гибкость в плане расширений и возможность кастомизации дают возможность применять данный инструмент в качестве внешнего анализатора кода.



*Рисунок 1.1. Диаграмма вариантов использования*

## Функциональные требования

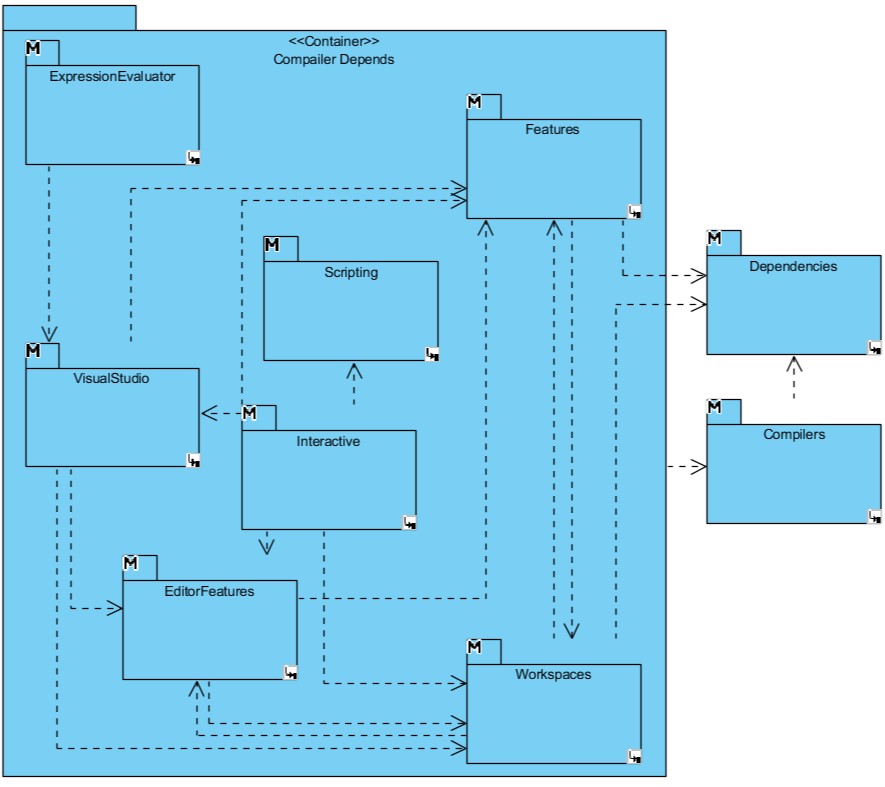
Основным требованием к системе является предоставление возможности взаимодействия с процессом компиляции исходного кода. Также можно выделить другие функциональные требования:

* построение и предоставление построенных семантических моделей
* предоставление API для имплементации кастомных анализаторов кода
* реализация возможности интерактивного взаимодействия с кодом
* поддержка .NET языков. Не только C#, но и Visual Basic.

## Нефункциональные требования

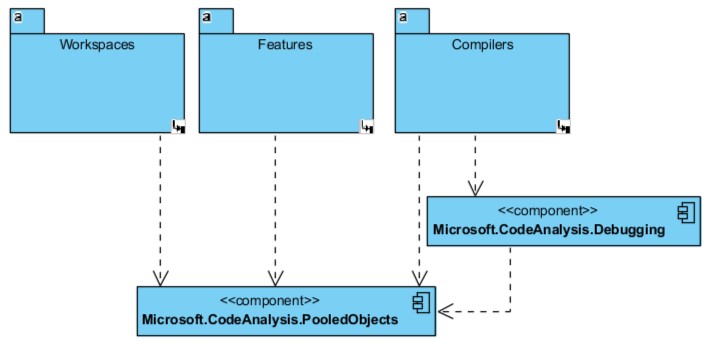
Среди не функциональных требований самым приоритетным является сохранность исходного кода – никакие операции компилятора не должны приводить к изменению исходного кода программы. Помимо этого, виду возможность использования одновременно многих анализаторов, должна поддерживаться целостность данных – ни на одном из этапов, данные не должны изменяться так, чтобы это отражалось на другие процессы. Ввиду того, что Roslyn является не первым компилятором языком C# и VB и не является основным, важным фактом является совместимость результата выполнения с стандартным компилятором .NET’а.

# Анализ средств автоматизации процессов

Архитектура Roslyn'а имеет ряд особенностей, которые объясняются набором функционал, который данное продукт реализует. В первую очередь, данное средство автоматизации является компилятором и именно логика процессов компиляции лежит в основании всей системы. Остальные компоненты являются функциональными расширениями процесса компиляции и являются зависимыми от компоненты "Compilers". Не смотря на то, что другие компонент группируются на основе какой-то функционала, они взаимодействуют между собой, переиспользуют структуры данных других модулей, используют их зависимости и расширяют их. Этим обусловлена сильная связанность между ними. На диаграмме представлена модель системы, где элементами являются именно эти компоненты.  


*Рисунок 2.1. Основные компоненты системы*

## Dependencies

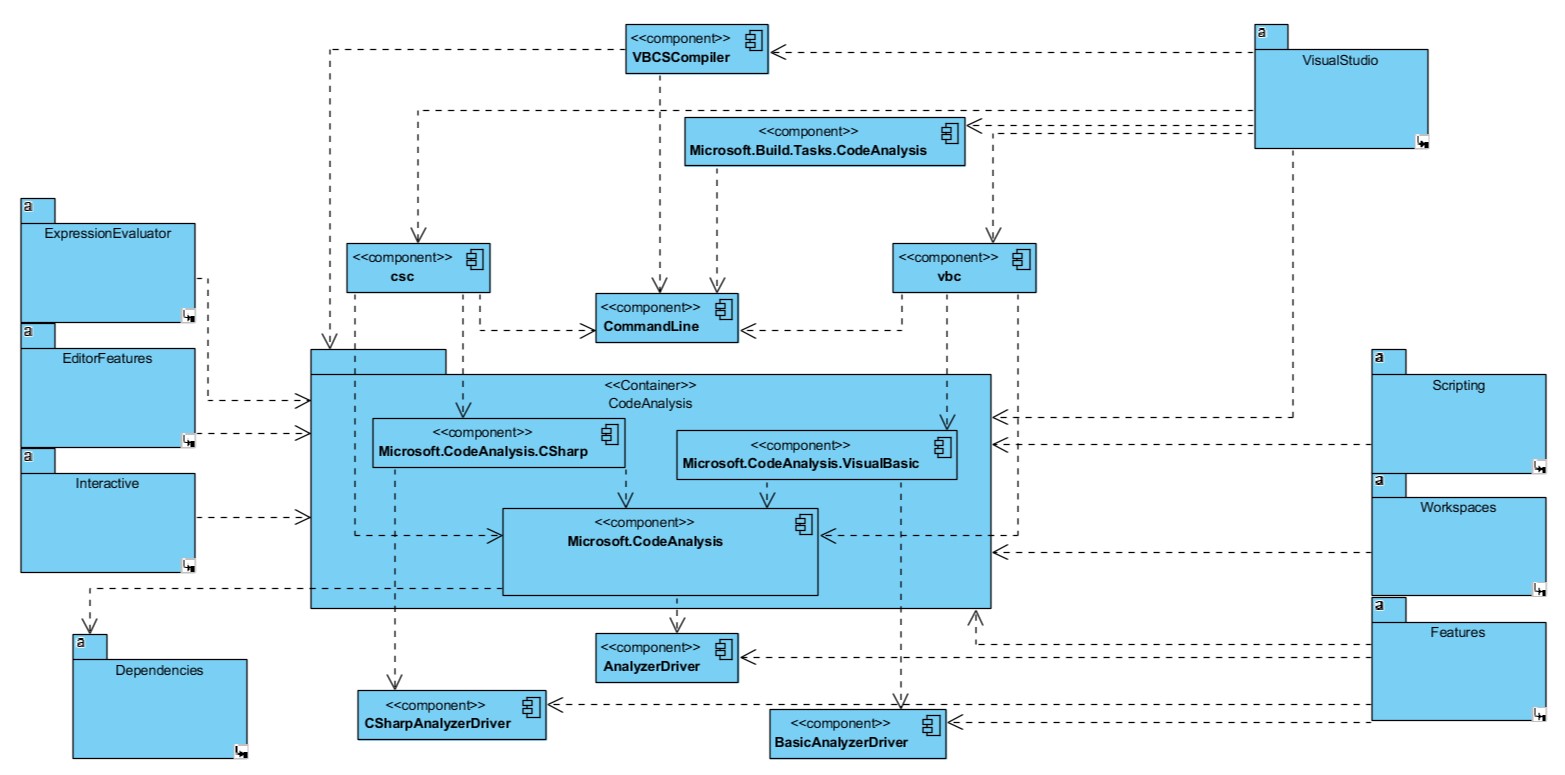
В отличии от всех остальных компонент, "Dependencies" не формируется по принципу общего функционала. Его основное назначение - описание базовых структуры данных, которые связанные с управлением памятью и временем жизни объектов. Иными словами, он агрегирует в себе общие для некоторых остальных компонент классы для минимизации дублирования схожего функционала, не создавая при этом не логичных зависимостей между компонентами.  


*Рисунок 2.2. Модули компонента Dependencies*

*Таблица 2.1*

|  |  |
| --- | --- |
| **Название модуля** | **Описание модуля** |
| Microsoft.CodeAnalysis .PooledObjects | Набор реализаций структур данных связанных с созданием объектов и управлением их временем жизни |
| Microsoft.CodeAnalysis .Debugging | Набор структур данных, которые агрегируют определённую информацию процесса отладки |

## Compilers

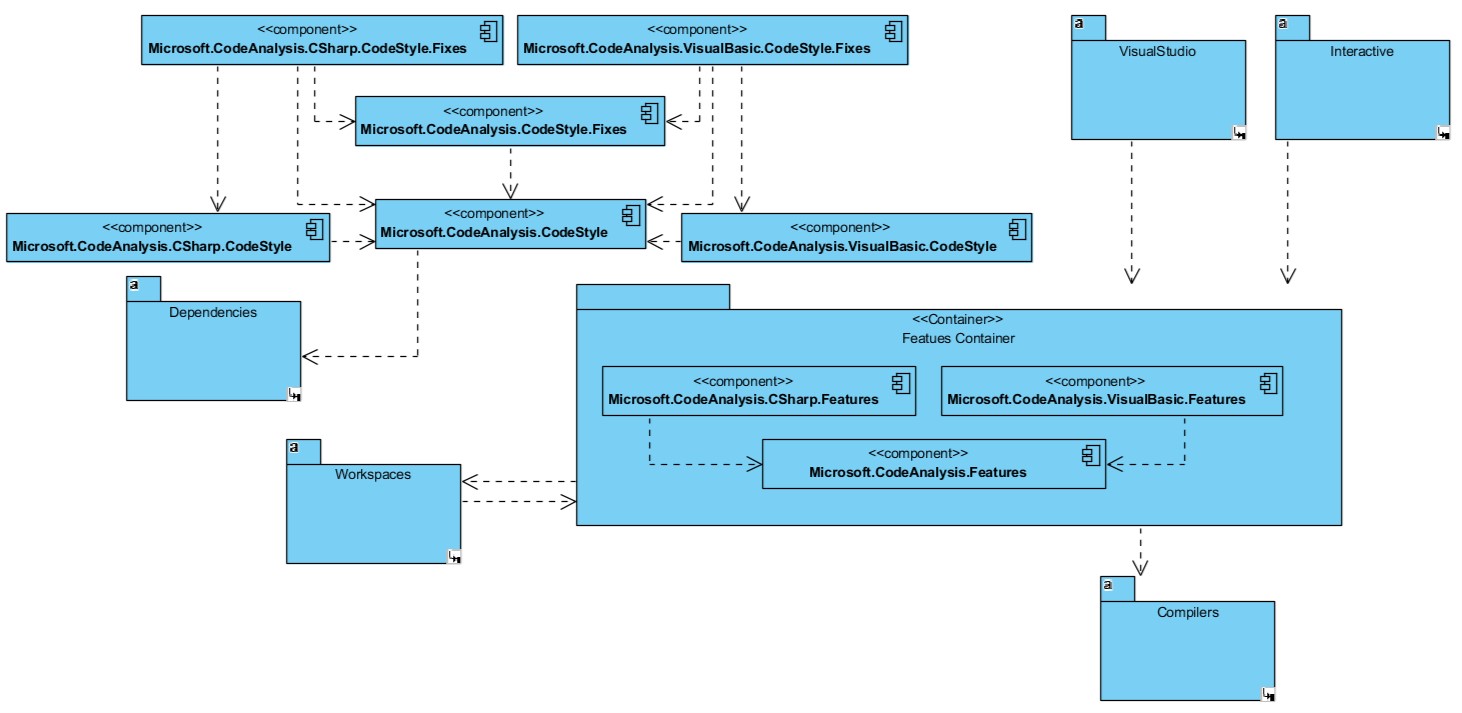
Как было указано ранее, фундаментальным функционалом данного программного средства является именно процесс компиляции. Ядром всей системы можно считать компонент, который отвечает за компиляцию исходного кода. Компонент Compilers предоставляет объектную модель, связанную с синтаксической и семантической информацией, которые представлены на каждом шаге пайплайна компилятора. Этот слой также включает иммутабельные структуры, связанные с работой компилятора [1, с. 35]. Compilers включает в себя логику разбора языков C# и VB. На этом уровне реализованы консольные инструменты для компиляции csc и vbc для указанных языков соответственно  


*Рисунок 2.2. Модули компонента Compilers*

*Таблица 2.2*

|  |  |
| --- | --- |
| **Название модуля** | **Описание модуля** |
| AnalyzerDriver | Структуры хранения исходного кода, синтаксического дерева и методы для работы с ними |
| BasicAnalyzerDriver | Расширение AnalyzerDriver с учетом особенностей языка VisualBasic |
| CommandLine | Вспомогательные классы для работы с консолью, в том числе вызова нативных методов по средствам обращения к kernel32.dll |
| csc | Модуль, который отвечает за логику взаимодействия с компилятором языка C# по средствам командной строки |
| CSharpAnalyzerDriver | Расширение AnalyzerDriver с учетом особенностей языка C# |
| Microsoft.Build.Tasks .CodeAnalysis |  |
| Microsoft.CodeAnalysis | Основной функционал системы, который связанный с анализом исходного кода и его разбором, общим для языков |
| Microsoft.CodeAnalysis.CSharp | Расширение Microsoft.CodeAnalysis с учетом особенностей языка C# |
| Microsoft.CodeAnalysis.VisualBasic | Расширение Microsoft.CodeAnalysis с учетом особенностей языка VisualBasci |
| vbc | Модуль, который отвечает за логику взаимодействия с компилятором языка VB по средствам командной строки |
| VBCSCompiler | Логика сервиса, который обрабатывает запросы на компиляцию. Агрегирует в себя функционал модулей csc и vbc |

## Features

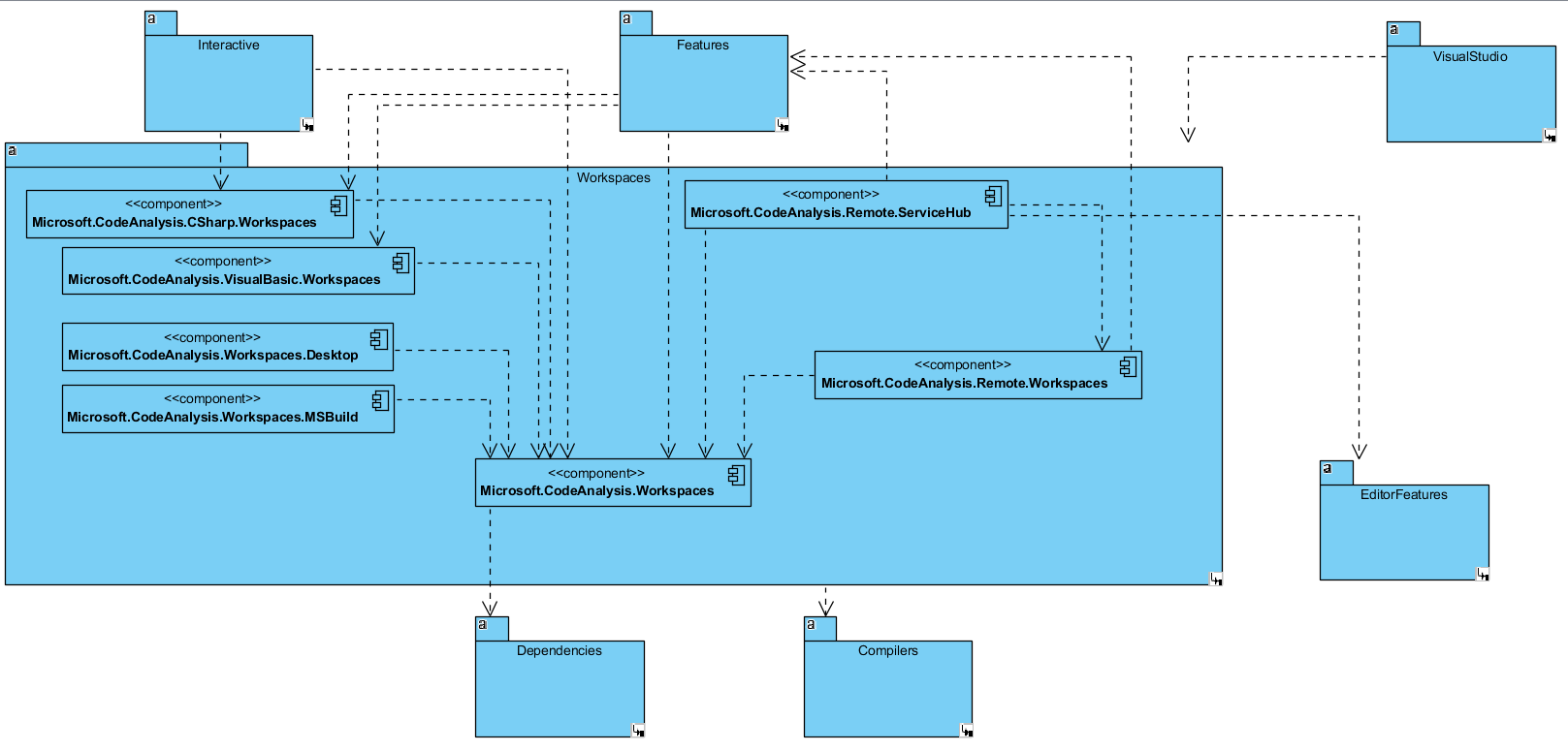
Помимо проверки исходного кода на валидность (что код компилируется без ошибок), есть более детальные анализаторы кода. Одним из частых требований к исходному кому – проверка код-стайла. При работе в команде, к примеру, очень важно соблюдать однообразность кода ввиду того, что это можешь отразиться на его читаемости. Roslyn имеет ряд функционала для анализа кода на соответствие какие-то установленным правилам, а также, автоматического изменения в случае не соответствия. Также, весомым функционалом данного компонента является то, что он реализует логику проверки кода на наличие участков, которые могут быть написаны, лучше применяя функционал новых версий языка.  


*Рисунок 2.3. Модули компонента Features*

*Таблица 2.3*

|  |  |
| --- | --- |
| **Название модуля** | **Описание модуля** |
| Microsoft.CodeAnalysis.CodeStyle | Интерфейсы, абстрактные модели и общая логика для функционала проверки кода на соответствие код-стайлу |
| Microsoft.CodeAnalysis.CodeStyle.Fixes | Функционал направленный на модификацию исходного кода применяя к нему правила описанные в Microsoft.CodeAnalysis.CodeStyle |
| Microsoft.CodeAnalysis.CSharp .CodeStyle | Имплементация Microsoft.CodeAnalysis.CodeStyle для языка C# |
| Microsoft.CodeAnalysis.CSharp .CodeStyle.Fixes | Расширение Microsoft.CodeAnalysis.CodeStyle.Fixes с учетом особенностей языка C# и его код-стайлу |
| Microsoft.CodeAnalysis.CSharp.Features | Имплементация Microsoft.CodeAnalysis.Features для языка C# |
| Microsoft.CodeAnalysis.Features | Декларация интефейсов а также общая логика для различного рода функционала работы с исходным кодом |
| Microsoft.CodeAnalysis.VisualBasic .CodeStyle | Имплементация Microsoft.CodeAnalysis.CodeStyle для языка VB |
| Microsoft.CodeAnalysis.VisualBasic .CodeStyle.Fixes | Расширение Microsoft.CodeAnalysis.CodeStyle.Fixes с учетом особенностей языка VB и его код-стайлу |
| Microsoft.CodeAnalysis.VisualBasic .Features | Имплементация Microsoft.CodeAnalysis.Features для языка VB |

## Workspaces

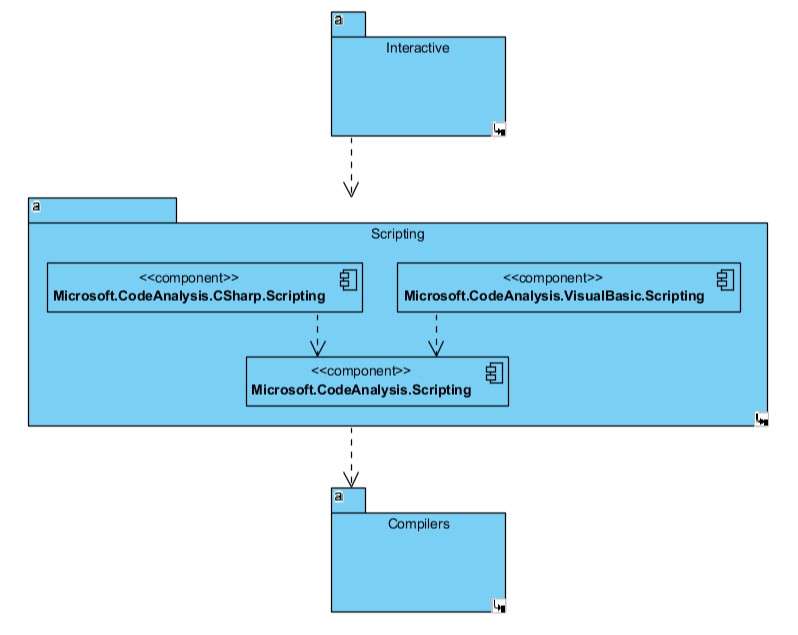
В большинстве ситуация, работа с кодовой базой сводится к тому, что есть определенный набор файлов исходного кода, который связан и к тому же имеет ряд внешних зависимостей. Применение анализаторов не на отдельно взятый код, а не всю структуру сильно расширяет возможность функционал. В случае Roslyn, используется абстракция Workspace (которая имеет и физическую имплементацию), которая включает в себя Solution с которым взаимодействует анализатор или отдельно взятый проект [1, с. 36].  


*Рисунок 2.4. Модули компонента Workspaces*

*Таблица 2.4*

|  |  |
| --- | --- |
| **Название модуля** | **Описание модуля** |
| Microsoft.CodeAnalysis.CSharp .Workspaces | Расширение воркспейса с учеток особенностей проектов ориентированных на C# |
| Microsoft.CodeAnalysis.Remote .ServiceHub | Функционал сервиса, который реализуется возможность хостинга Workspace для подключенных внешних клиентов |
| Microsoft.CodeAnalysis.Remote. Workspaces | Расширение Microsoft.CodeAnalysis.Workspaces для имплементации особенностей удаленных Workspace |
| Microsoft.CodeAnalysis.VisualBasic .Workspaces | Расширение воркспейса с учеток особенностей проектов ориентированных на Visual Basic |
| Microsoft.CodeAnalysis.Workspaces | Основная логика предствления и обработки Workspace, используемые в нем структуры данных |
| Microsoft.CodeAnalysis.Workspaces .Desktop | Расширение функционала Microsoft.CodeAnalysis.Workspaces, которое связанное с особеностями используемых Workspace в IDE |
| Microsoft.CodeAnalysis.Workspaces .MSBuild | Имплементация логики взаимодействия MSBuild с Workspace |

## Scripting

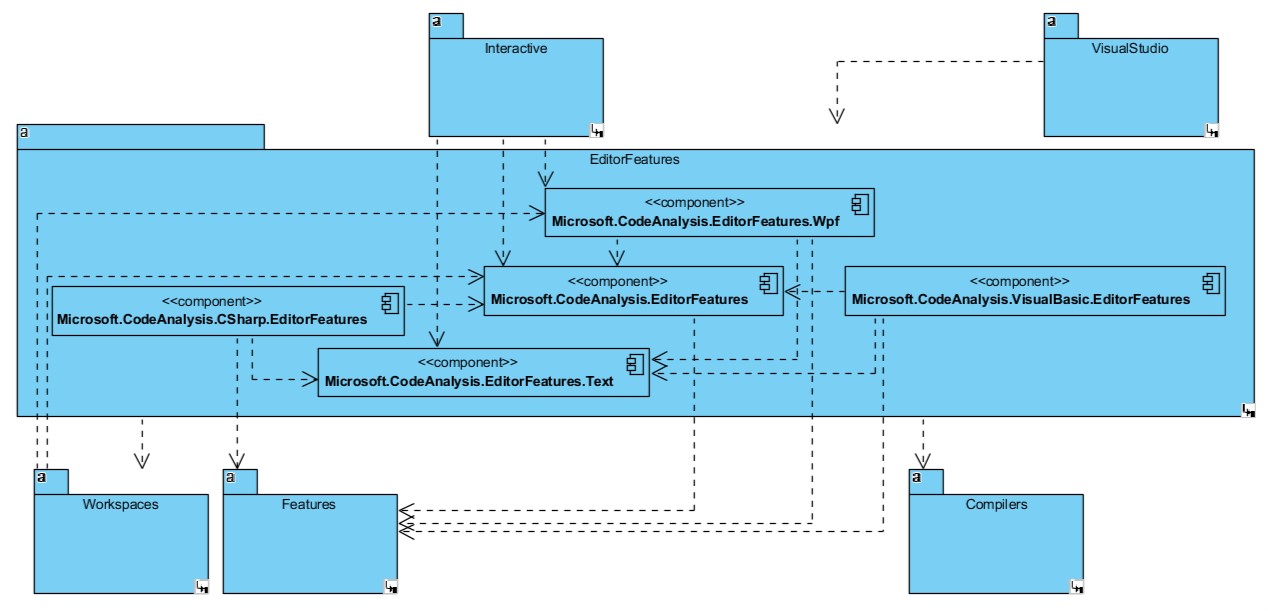
Данный компонент связанный с функционалом обработки скриптов. Скриптами в данном контексте называют отдельные исполняемые файлы, которые могут выполняться отдельно от проекта. [2, с.432]  


*Рисунок 2.5. Модули компонента Scripting*

*Таблица 2.5*

|  |  |
| --- | --- |
| **Название модуля** | **Описание модуля** |
| Microsoft.CodeAnalysis.Scripting | Инфраструктура для реализации скриптинга |
| Microsoft.CodeAnalysis.CSharp.Scripting | Имплементация возможности скриптинга на C# |
| Microsoft.CodeAnalysis.VisualBasic.Scripting | Имплементация возможности скриптинга на VB |

## EditorFeatures

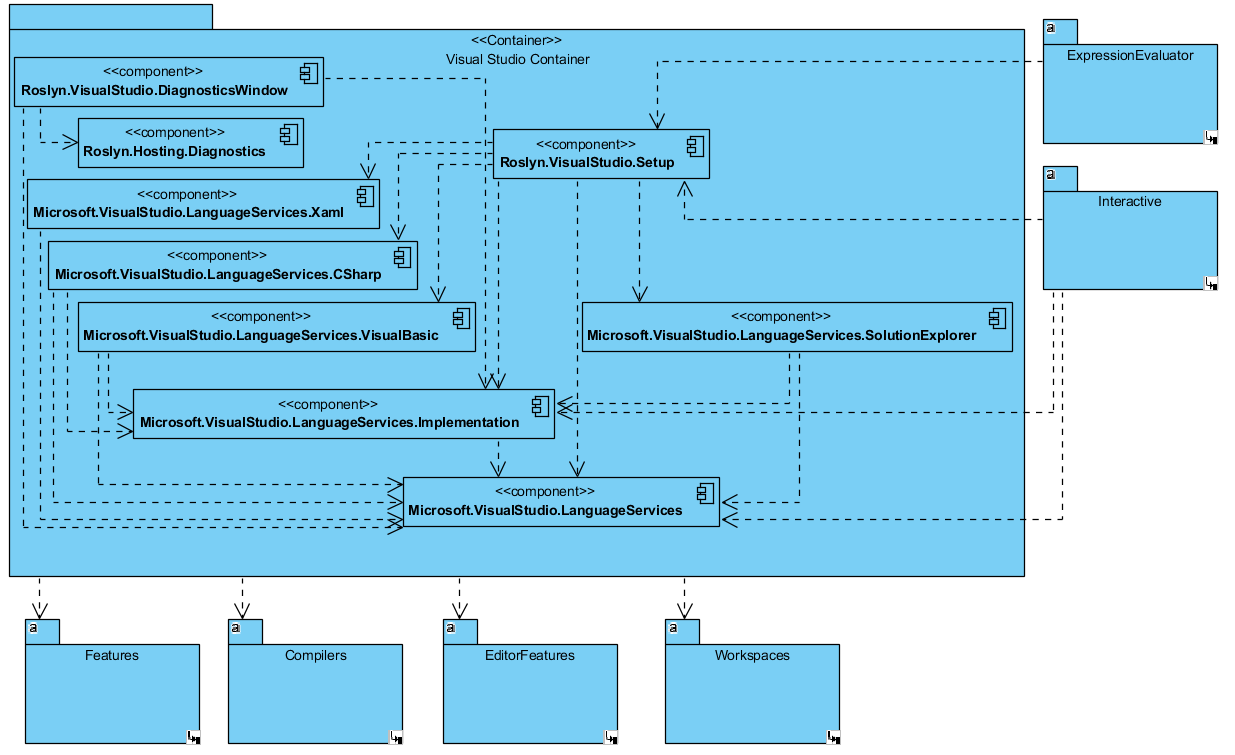
Данный компонент включает в себя функционал работы с текстом исходного кода в рамках различных редакторов, которые реализуют его.  


*Рисунок 2.6. Модули компонента EditorFeatures*

*Таблица 2.6*

|  |  |
| --- | --- |
| **Название модуля** | **Описание модуля** |
| Microsoft.CodeAnalysis.CSharp .EditorFeatures | Расширение Microsoft.CodeAnalysis.EditorFeatures с учетом особенностей языка C# |
| Microsoft.CodeAnalysis.EditorFeatures | Набор различного функционала связанного с отображением, редактированием и форматированием кода |
| Microsoft.CodeAnalysis.EditorFeatures .Text | Базовые компоненты для имплементации функционала связанных с отображением и редактированием текста исходного кода |
| Microsoft.CodeAnalysis.EditorFeatures .Wpf | Расширение Microsoft.CodeAnalysis.EditorFeatures для применения при работе с xaml кодом в рамках технологии WPF |
| Microsoft.CodeAnalysis.VisualBasic .EditorFeatures | Расширение Microsoft.CodeAnalysis.EditorFeatures с учетом особенностей языка VB |

## VisualStudio

Ввиду того, что основной IDE для языком фреймворка .NET является именно Visual Studio, часть функционала Roslyn’а пишется с ориентацией на нее. Данный компонент представляет собой агрегацию всего функционала Roslyn собранную с учетом особенностей именно Visual Studio. Так, например, частью данного компонента является конфигурационный модуль, который отвечает за установку Roslyn компонентов в данную IDE.  


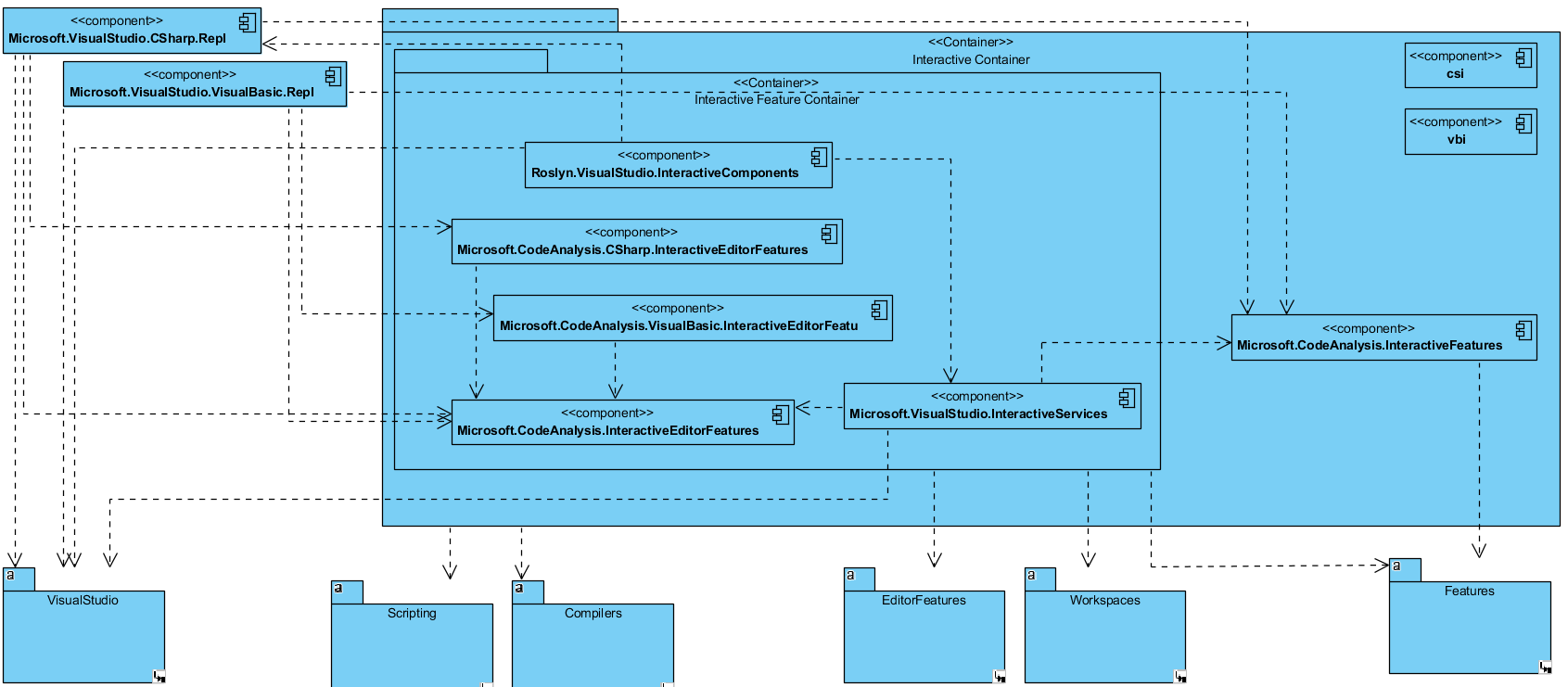
*Рисунок 2.7. Модули компонента VisualStudio*

*Таблица 2.7*

|  |  |
| --- | --- |
| **Название модуля** | **Описание модуля** |
| Microsoft.VisualStudio.LanguageServices | Набор абстракций и структур, связанных с логикой обработкой исходного кода в Visual Studio основанных на остальных базовых компонентах системы |
| Microsoft.VisualStudio.LanguageServices.CSharp | Расширение Microsoft.VisualStudio.LanguageServices для реализации особенностей языка C# |
| Microsoft.VisualStudio.LanguageServices .Implementation | Модуль, который агрегирует все базовые функциональные компоненты, связанные с обработкой исходного кода для редактора в Visual Studio |
| Microsoft.VisualStudio.LanguageServices .SolutionExplorer | Функционал для взаимодействия с Workspace и элементами Solution'ов, который используется в Visual Studio |
| Microsoft.VisualStudio.LanguageServices .VisualBasic | Расширение Microsoft.VisualStudio.LanguageServices для реализации особенностей языка VB |
| Microsoft.VisualStudio.LanguageServices.Xaml | Расширение Microsoft.VisualStudio.LanguageServices для реализации особенностей языка Xaml |
| Roslyn.Hosting.Diagnostics | Дополнительный модуль более глубокого анализа кода, который разрабатывается отдельно. В основном сюда включается логика связанная с новым функционалом языков и студии |
| Roslyn.VisualStudio.DiagnosticsWindow | Модуль с метаданными и реализацией взаимодействия с Visual Studio, который распространяет Roslyn.Hosting.Diagnostics как расширение для Visual Studio |
| Roslyn.VisualStudio.Setup | Агрегатор всего функционала модулей для сборки как расширение для Visual Studio |

## Interactive

Несмотря на то, что C# и VB являются компилируемыми языками, с использованием базовых компонент Roslyn, было реализован возможность интерактивного взаимодействия. Компонент Interactive предоставляет функционал, который позволяет писать код на данных языках, который будет сразу же исполняться и возвращать результат подобно тому, как это реализуют в интерпретируемых языках. [4]

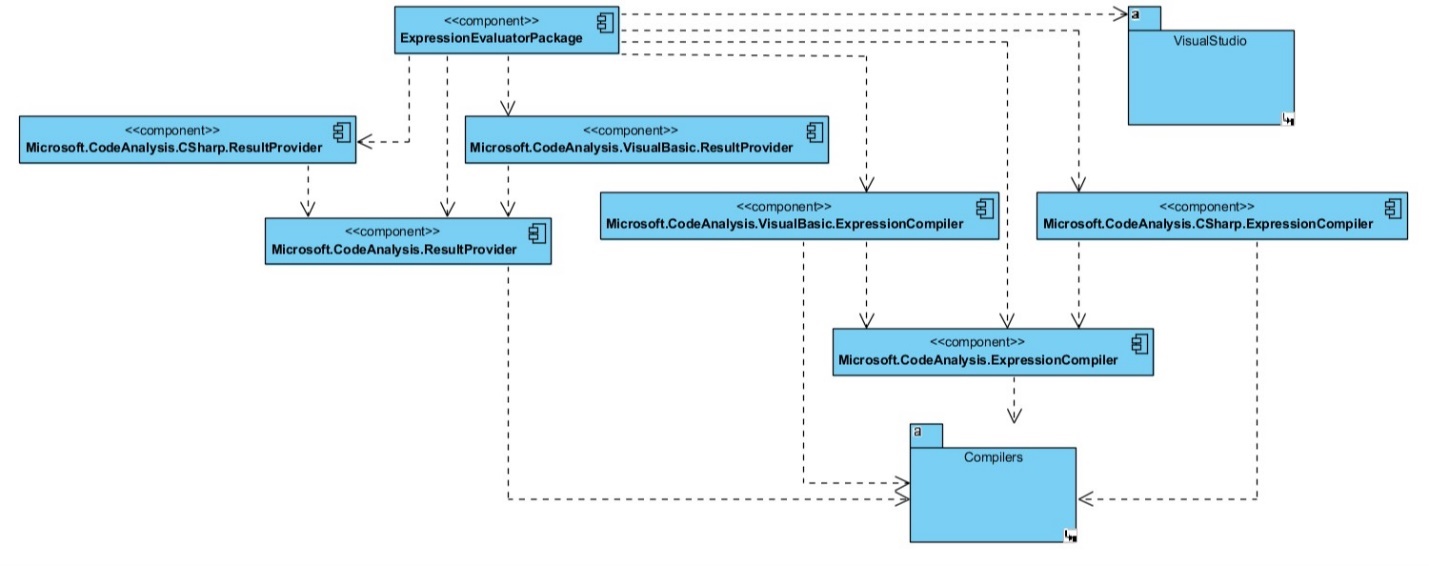


*Рисунок 2.8. Модули компонента Interactive*

*Таблица 2.8*

|  |  |
| --- | --- |
| **Название модуля** | **Описание модуля** |
| csi | Консольный инструмент для работы с Interactive с использованием языка C# |
| Microsoft.CodeAnalysis.CSharp .InteractiveEditorFeatures | Имплементация Microsoft.CodeAnalysis.InteractiveEditorFeatures для работы с языком C# |
| Microsoft.CodeAnalysis .InteractiveEditorFeatures | Базовый функционал редактора для использования Interactive |
| Microsoft.CodeAnalysis .InteractiveFeatures | Логика и описание данных связанных с функционалом Interactive |
| Microsoft.CodeAnalysis.VisualBasic .InteractiveEditorFeatures | Имплементация Microsoft.CodeAnalysis.InteractiveEditorFeatures для работы с языком VB |
| Microsoft.VisualStudio.CSharp.Repl | Модуль расширения для Visual Studio, который реализуется возможность взаимодействия с Interactive с использованием языка C# |
| Microsoft.VisualStudio .InteractiveServices | Логика связанная с интеграцией функционала Interactive с Visual Studio |
| Microsoft.VisualStudio.VisualBasic .Repl | Модуль расширения для Visual Studio, который реализуется возможность взаимодействия с Interactive с использованием языка VB |
| Roslyn.VisualStudio .InteractiveComponents | Модуль расширения для Visual Studio, который внедряет функционал взамодействия c REPL |
| vbi | Консольный инструмент для работы с Interactive с использованием языка VB |

## ExpressionEvaluator

Одним из видов анализа кода является разбор выражений. В отличии от низкоуровневого разбора на уровне компиляции, компонент ExpressionEvaluator отвечает за генерацию высокоуровневых сущностей из исходного кода. Отличным примером является обработка исходных строк с математическими выражениями и генерация на их основе лябда выражений.  


*Рисунок 2.9. Модули компонента ExpressionEvaluator*

*Таблица 2.9*

|  |  |
| --- | --- |
| **Название модуля** | **Описание модуля** |
| ExpressionEvaluatorPackage | Агрегатор модулей обработки выражений, который является расширением для Visual Studio |
| Microsoft.CodeAnalysis.CSharp .ExpressionCompiler | Расширение Microsoft.CodeAnalysis.ExpressionCompiler с учетом особенностей языковых конструкций в C# |
| Microsoft.CodeAnalysis.CSharp .ResultProvider | Раширение Microsoft.CodeAnalysis.ResultProvider с учетом особенностей языка C# |
| Microsoft.CodeAnalysis .ExpressionCompiler | Модуль в котором описана логика непосредственно разбора выражений |
| Microsoft.CodeAnalysis.ResultProvider | Набор структур, которые используются в рамках данного модуля для описания сущностей связанных с обработкой выражений |
| Microsoft.CodeAnalysis.VisualBasic .ExpressionCompiler | Расширение Microsoft.CodeAnalysis.ExpressionCompiler с учетом особенностей языковых конструкций в VB |
| Microsoft.CodeAnalysis.VisualBasic .ResultProvider | Раширение Microsoft.CodeAnalysis.ResultProvider с учетом особенностей языка VB |

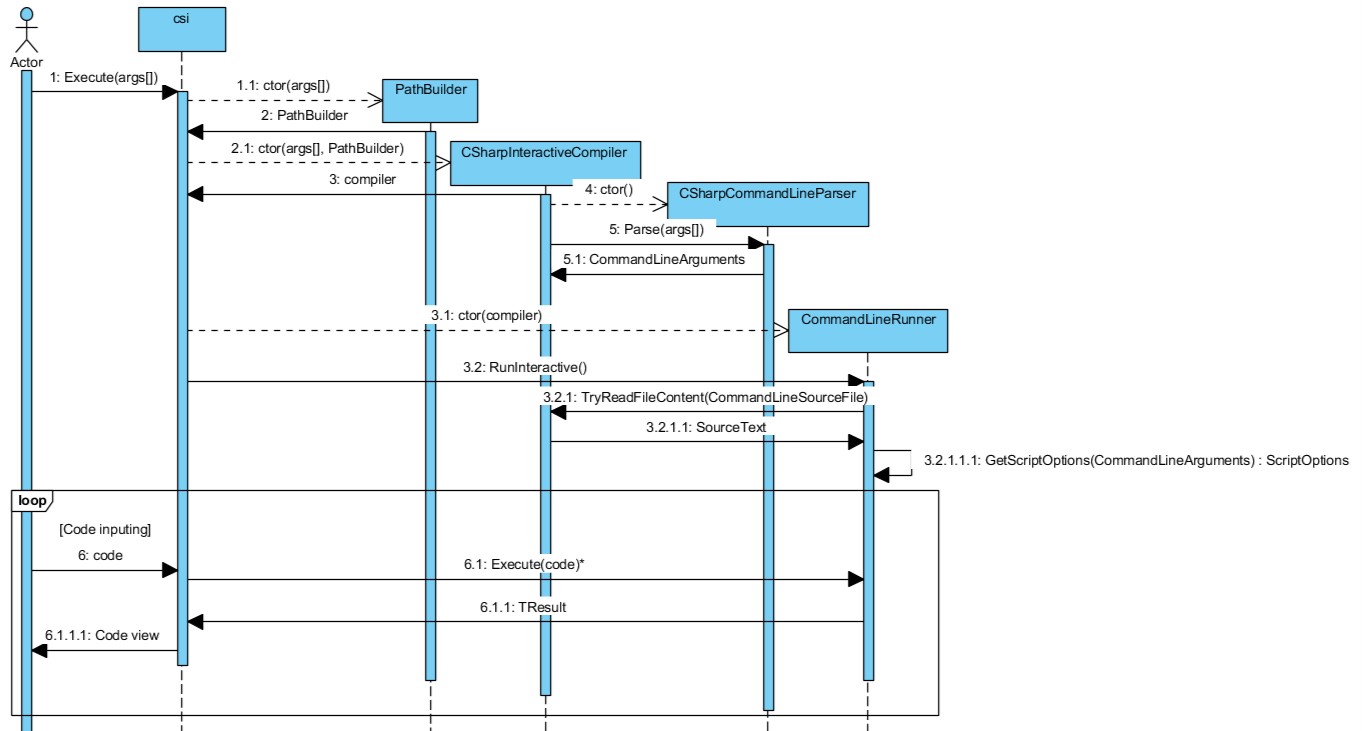
# Синтез определенных уровней архитектуры ИС

Как было заявлено ранее, Roslyn не является компилятором в обычном понимании этого термина, а предоставляет собой некую платформу взаимодействия с процессом компиляции. Он создает определенный уровень абстракции над типовыми операциями.

Один из примеров реализации функционала на основе Roslyn является Interactive. Интерактивное взаимодействие с кодом в Roslyn называют возможность написания код построчно без необходимости пересобрать проект. Очевидно, что компилируемые языки не могут выполняться также, как и интерпретируемые, но модули компонента Roslyn реализуют функционал обработки входного кода и репкомпиляции введенного ранее кода для создания подобия «не компилируемых языков».

Для взаимодействия с Interactive без использования IDE отвечает класс csi из компонента Interactive. Данный класс в процессе инициализации создает экземпляр класса CSharpInteractiveCompiler из компонента Scripting, который является расширением класса CSharpCompiler из компонента Compilers. Ввиду того, что взаимодействие пользователя и компилятора выполняет по средствам консоли, команды и настройки парсеру приходят в виде аргументов командной строки [5]. Для их разбора в данном контексте используется класс CSharpCommandLineParser вся логика которого и сосредоточена вокруг парсинга этих аргументов.

За непосредственно обработку кода в рамках Interactive отвечает класс CommandLineRunner из компонента Scripting. У данного класса можно выделить две основные зоны ответственности – генерация и вывод информации пользователю, а также перенаправление введенной пользователем информацию к компиляторам.



*Рисунок 3.1. Взаимодействие модулей при обращении пользователя к csi*

На данном уровне абстракции процесса были рассмотрены такие информационные объекты:

Для представления в системе аргументов, которые пользователь передает csi, используется набор строк, который в процессе парсинга приводится к стуктуре CommandLineArguments:

*Таблица 3.1*

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| ReferencePaths : ImmutableArray<string> | Набор путей, который используются для поиска внешних ссылок |
| RuleSetPath : string | Путь к файлу .ruleset |
| OutputDirectory : string | Путь к папке, в которую будет выводится результат выполнения |
| ErrorLogPath : string | Путь к файлу, в который логируется информация об ошибках |
| AnalyzerReferences : ImmutableArray<string> | Пути к файлам анализаторам кода |
| SourceFiles : ImmutableArray<string> | Пути на исходные файлы кода, которые загружаются перед скриптом |
| ParseOptionsCore : ParseOptions | Опции для процесса парсинга |
| CompilationOptionsCore : CompilationOptions | Опции для процесса компиляции |

ParseOptions:

*Таблица 3.2*

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| Kind : SourceCodeKind | Тип исходного кода - Regular или Script |
| Language : string | Язык, на котором написан данный код |
| Errors : ImmutableArray<Diagnostic> | Список синтаксических ошибок в файле |

Diagnostic:

*Таблица 3.3*

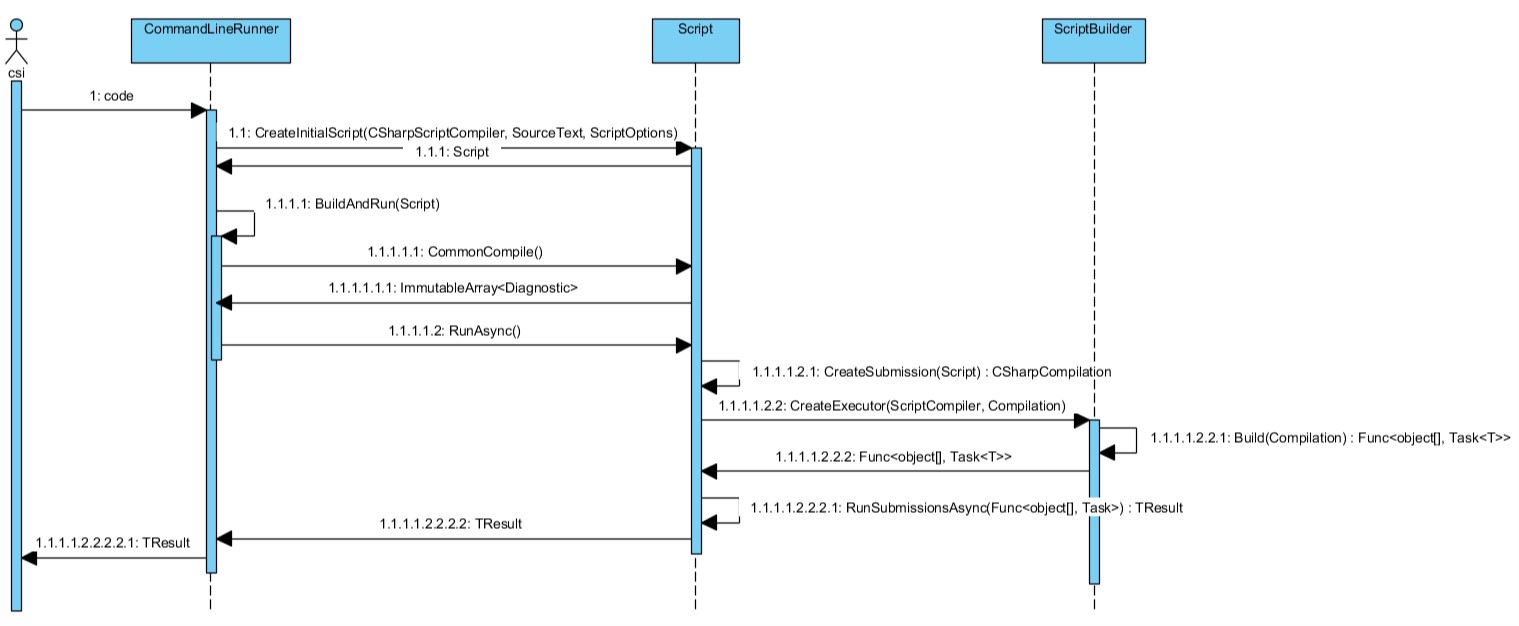
|  |  |
| --- | --- |
| **Поле** | **Описание** |
| Descriptor : string | Описание диагностики |
| Id : string | Идентификатор диагностики |
| Severity : DiagnosticSeverity | Тип диагностики - Hidden, Info, Warning, Error |
| Location : Location | Расположение источника, с которым связана диагностика |

CompilationOptions:

*Таблица 3.4*

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| OutputKind : OutputKind | Тип асембли, что будет сгенерирован в результате - ConsoleApplication, WindowsApplication, DynamicallyLinkedLibrary, WindowsRuntimeApplication |
| ModuleName : string | Название модуля, что будет сгенерирован |
| Platform : Platform | Платформа под которую буде скомпилирован код - X86, X64, Arm |
| OptimizationLevel : OptimizationLevel | Уровень оптимизации кода - Debug, Release |

Рассмотрим более детально процесс непосредственно обращения на выполнение кода. На рисунку 3.1 отображен процесс обращения Actor’а к csi с кодом, который дальше перенаправляется к CommandLineRunner. Во время обработки этого вызов, данный класс обращается к классам Script и ScriptBuilder из компонента Scripting.



*Рисунок 3.2. Взаимодействие модулей при выполнении введенного кода*

Класс скрипт, помимо логики выполнения, хранит некоторую информацию, когда нужна для выполнения:

*Таблица 3.5*

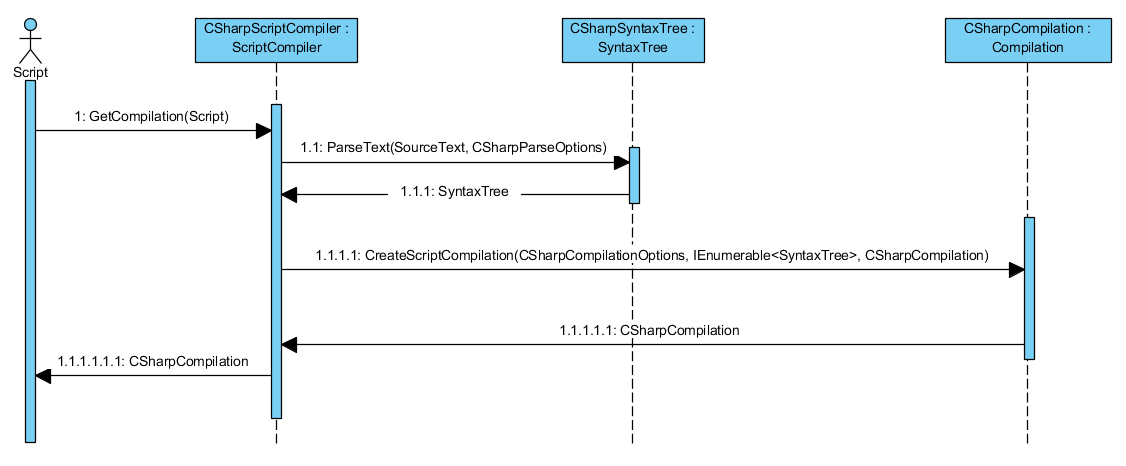
|  |  |
| --- | --- |
| **Поле** | **Описание** |
| Options : ScriptOptions | Настройки, которые применяются к данному скрипту |
| SourceText : SourceText | Исходный текст скрипта |
| ReturnType : Type | Тип значения, что возвращает скрипт |

ScriptOptions, это структура, которая описывает информацию как скрипте из Interactive, так и данные для работы с физическим файлом, который хранит в себе скрипт. Если рассматривать его в рамках Interactive, то в нем есть только одно используемое поле:

*Таблица 3.6*

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| MetadataReferences : ImmutableArray<MetadataReference> | Ссылки на метадату модулей, которые он использует |

Одним из этапов выполнения расписанного ранее, является вызов метода GetCompilation(Script) : CSharpCompilation. Это непосредственно та часть процесса обработки запроса, когда идет работа с компилятором. Если спуститься ниже по абстракции, то можно дополнить ранее описанные этапы данной последовательностью вызовов:



*Рисунок 3.3. Взаимодействие модулей для получения CSharpCompilation*

Класс CSharpScriptCompiler, который относится к компоненте Scripting, выполняет обработку скрипта в два этапа. На первом, использует CSharpSyntaxTree и его метод ParseTree для генерации SyntaxTree из исходного кода. Внутри PraseTree уже задействованы модули из компоненты Compilers. SyntaxTree: [3]

*Таблица 3.7*

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| Options : ParseOptions | Найстройки парсера |
| Length : int | Длинна синтаксического дерева |
| Root : SyntaxNode | Корневой элемент дерева |
| SourceText : SourceText | Исходный текст |

SyntaxNode:

*Таблица 3.8*

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| Parent : SyntaxNode | Родительский элемнт дерева |
| SyntaxTree : SyntaxTree | Дерево, в котором находится элемент |
| RawKind : int | Тип элемента |
| KindText : string | Текст элемента |

На втором этапе, полученное SyntaxTree, а также указанные в Script параметры компиляции, передается методу CreateScriptCompilation класса CSharpCompilation, который является частью компоненты Compilers. Результатом выполнения является CSharpCompilation, который и запрашивался из класса Script в рамках данного процесса.

CSharpCompilation:

*Таблица 3.9*

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| Language : string | Название используемого языка |
| Options : CSharpCompilationOptions | Настройки компилятора, что были использованны |
| IsCaseSensitive : bool | Является ли код чувствительным к регистру |
| LanguageVersion : LanguageVersion | Версия язык, которая была использована - C#5, 6, 7, 7.1, 7.2… |
| SyntaxTrees : IEnumerable<SyntaxTree> | Список синтаксических деревьев |

# Вывод

Раскрытие «черного ящика» компиляторов дает широкий ряд возможностей для улучшения условий работы с кодом. Программный компонент Roslyn, который является платформой предоставляющей функционал компилятора, как раз таки нацелена на интеграцию этих процессов, их управление и использование в процессе разработки.

В ходе анализа была рассмотрена структура проекта, выделены основные компоненты и описан их функционал. Часть этих компонент предоставляются как библиотеки для проектов с целью их расширения и имплементации на их основе нового функционала. А значит, полученные теоретические знания об их структуре буду полезными в дальнейшем при работе с Roslyn.

В процессе синтеза уровней архитектуры были выделены функциональные и программные уровни, выполнено их сопоставление. При анализе было отмечено, что программный компонент представляется на разных уровнях абстракции – с ним можно интегрироваться как на уровне добавления правил код-стайла, так и внедрить дополнительную логику разбора исходного кода.

Обобщив информацию полученную в ходе работы над Roslyn, можно прийти к заключению, что это современный инструмент, который значительно может упростить работы как с анализом кода, так и его написанием (например, кодогенерацией), а также дает возможность пользователям разрабатывать инструменты, которые решают непосредственно их задачу и настраивать уже существующие под себя.

# Список использованной литературы

1. Roslyn Succinctly / Alessandro Del Sole. – 2016 г.
2. Roslyn Cookbook / Manish Vasani. – 2017 г.
3. The .NET Compiler Platform SDK [Электронный ресурс] - <https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/work-with-syntax>
4. Interactive Window [Электронный ресурс] - <https://github.com/dotnet/roslyn/wiki/Interactive-Window>
5. Interactive Window [Электронный ресурс] - <https://github.com/dotnet/roslyn/wiki/C%23-Interactive-Walkthrough>