Kostenloses eBook

LERNEN

roslyn

Free unaffiliated eBook created from
Stack Overflow contributors.

#roslyn

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: roslyn

It is an unofficial and free roslyn ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official roslyn.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Roslyn

Bemerkungen

Um mit Roslyn zu beginnen, werfen Sie einen Blick auf:

- Syntaxbaum-API
- Semantisches Modell-API
- *Fügen Sie hier weitere Themen hinzu* .

Examples

Installation oder Setup

Um mit Roslyn zu basteln, benötigen Sie die folgenden NuGet-Pakete:

- Die C # - und VB-Compiler - `Microsoft.Net.Compilers` . Zur Installation können Sie den folgenden Befehl in der Package Manager Console ausführen:

  ```
  nuget install Microsoft.Net.Compilers
  ```

- Die Sprach-APIs und -Dienste - `Microsoft.CodeAnalysis` . Zur Installation können Sie den folgenden Befehl in der Package Manager Console ausführen:

  ```
  nuget install Microsoft.CodeAnalysis
  ```

Außerdem ist es ratsam, die .NET Compiler Platform SDK-Vorlagen zu installieren, die hier zu finden sind . Das wird dich bekommen:

- Vorlagen für C # und Visual Basic, die die Erstellung von Analysatoren, CodeFixes und eigenständigen Analysewerkzeugen ermöglichen.
- Das Syntax Visualizer-Tool für Visual Studio (`View -> Other Windows -> Syntax Visualizer`), das äußerst hilfreich für die Untersuchung des Syntaxbaums vorhandenen Codes ist.

Zusätzliche Tools und Ressourcen

- Der Roslyn Quoter

Ein Tool zum Konvertieren eines C # -Programmprogramms in Syntaxbaum-API-Aufrufe Das Tool selbst finden Sie hier .

- Verbesserter Source Viewer

Eine einfache Möglichkeit zum Anzeigen des Roslyn-Quellcodes finden Sie hier .

Erste Schritte mit Roslyn online lesen: https://riptutorial.com/de/roslyn/topic/2905/erste-schritte-mit-roslyn

Kapitel 2: Analysieren Sie den Quellcode mit Roslyn

Examples

Introspektive Analyse eines Analysators in C

1. Erstellen Sie eine neue **Konsolenanwendung**
2. Fügen Sie das **NuGet-** Paket `Microsoft.CodeAnalysis`
3. Importieren Sie die Namespaces `Microsoft.CodeAnalysis.MSBuild` , `System.Linq` und `Microsoft.CodeAnalysis.CSharp.Syntax`
4. Schreiben Sie den folgenden Beispielcode in die `Main` Methode:

```
// Declaring a variable with the current project file path.
const string projectPath = @"C:\<your path to the project\<project file name>.csproj";

// Creating a build workspace.
var workspace = MSBuildWorkspace.Create();

// Opening this project.
var project = workspace.OpenProjectAsync(projectPath).Result;

// Getting the compilation.
var compilation = project.GetCompilationAsync().Result;

// As this is a simple single file program, the first syntax tree will be the current file.
var syntaxTree = compilation.SyntaxTrees.First();

// Getting the root node of the file.
var rootSyntaxNode = syntaxTree.GetRootAsync().Result;

// Finding all the local variable declarations in this file and picking the first one.
var firstLocalVariablesDeclaration =
rootSyntaxNode.DescendantNodesAndSelf().OfType<LocalDeclarationStatementSyntax>().First();

// Getting the first declared variable in the declaration syntax.
var firstVariable = firstLocalVariablesDeclaration.Declaration.Variables.First();

// Getting the text of the initialized value.
var variableInitializer = firstVariable.Initializer.Value.GetFirstToken().ValueText;

// This will print to screen the value assigned to the projectPath variable.
Console.WriteLine(variableInitializer);

Console.ReadKey();
```

Beim Ausführen des Projekts wird die oben deklarierte Variable auf dem Bildschirm angezeigt. Dies bedeutet, dass Sie ein Projekt erfolgreich selbst analysiert und eine Variable darin gefunden haben.

Analysieren Sie eine einfache "Hello World" -Anwendung in C

Erstellen Sie eine neue Konsolenanwendung mit einer Zeile in der `Main` Methode:
`Console.WriteLine("Hello World")`

Merken Sie sich den Pfad zur `.csproj` Datei und ersetzen Sie ihn im Beispiel.

Erstellen Sie eine neue **Konsolenanwendung,** installieren Sie das `Microsoft.CodeAnalysis` NuGet-Paket und versuchen Sie den folgenden Code:

```
const string projectPath = @"C:\HelloWorldApplication\HelloWorldProject.csproj";

// Creating a build workspace.
var workspace = MSBuildWorkspace.Create();

// Opening the Hello World project.
var project = workspace.OpenProjectAsync(projectPath).Result;

// Getting the compilation.
var compilation = project.GetCompilationAsync().Result;

foreach (var tree in compilation.SyntaxTrees)
{
    Console.WriteLine(tree.FilePath);

    var rootSyntaxNode = tree.GetRootAsync().Result;

    foreach (var node in rootSyntaxNode.DescendantNodes())
    {
        Console.WriteLine($" *** {node.Kind()}");
        Console.WriteLine($"     {node}");
    }
}

Console.ReadKey();
```

Dadurch werden alle Dateien und alle **Syntaxknoten** in Ihrem **Hello World-** Projekt gedruckt.

Analysieren Sie eine einfache "Hello World" -Anwendung in VB.NET

Erstellen Sie eine neue Konsolenanwendung mit einer Zeile in der `Main` Methode:
`Console.WriteLine("Hello World")`

Merken Sie sich den Pfad zur `.vbproj` Datei und ersetzen Sie ihn im Beispiel.

Erstellen Sie eine neue **Konsolenanwendung,** installieren Sie das `Microsoft.CodeAnalysis` NuGet-Paket und versuchen Sie den folgenden Code:

```
Const projectPath = "C:\HelloWorldApplication\HelloWorldProject.vbproj"

' Creating a build workspace.
Dim workspace = MSBuildWorkspace.Create()

' Opening the Hello World project.
Dim project = workspace.OpenProjectAsync(projectPath).Result

' Getting the compilation.
Dim compilation = project.GetCompilationAsync().Result
```

```
For Each tree In compilation.SyntaxTrees

    Console.WriteLine(tree.FilePath)

    Dim rootSyntaxNode = tree.GetRootAsync().Result

    For Each node In rootSyntaxNode.DescendantNodes()

        Console.WriteLine($" *** {node.Kind()}")
        Console.WriteLine($"     {node}")
    Next
Next

Console.ReadKey()
```

Dadurch werden alle Dateien und alle **Syntaxknoten** in Ihrem **Hello World-** Projekt gedruckt.

Quellcode aus Text in C # analysieren

```
var syntaxTree = CSharpSyntaxTree.ParseText(
@"using System;
using System.Collections;
using System.Linq;
using System.Text;

namespace HelloWorldApplication
{
class Program
{
static void Main(string[] args)
{
Console.WriteLine(""Hello World"");
}
}
}");

var root = syntaxTree.GetRoot() as CompilationUnitSyntax;

var namespaceSyntax = root.Members.OfType<NamespaceDeclarationSyntax>().First();

var programClassSyntax = namespaceSyntax.Members.OfType<ClassDeclarationSyntax>().First();

var mainMethodSyntax = programClassSyntax.Members.OfType<MethodDeclarationSyntax>().First();

Console.WriteLine(mainMethodSyntax.ToString());

Console.ReadKey();
```

In diesem Beispiel wird die `Main` Methode aus dem Text gedruckt, der die Syntax analysiert.

Holen Sie sich den Typ von 'var'

Rufen Sie `GetSymbolInfo()` für `SemanticModel` auf, um den tatsächlichen Typ für eine mit `var` deklarierte Variable `GetSymbolInfo()` . Sie können eine vorhandene Lösung mit `MSBuildWorkspace` und dann ihre Projekte und ihre Dokumente `MSBuildWorkspace` . Verwenden Sie ein Dokument, um `SyntaxRoot`

und `SemanticModel SyntaxRoot` dann nach `VariableDeclarations` und `SyntaxRoot` die Symbole für den `Type` einer deklarierten Variablen ab:

```
var workspace = MSBuildWorkspace.Create();
var solution = workspace.OpenSolutionAsync("c:\\path\\to\\solution.sln").Result;

foreach (var document in solution.Projects.SelectMany(project => project.Documents))
{
    var rootNode = document.GetSyntaxRootAsync().Result;
    var semanticModel = document.GetSemanticModelAsync().Result;

    var variableDeclarations = rootNode
            .DescendantNodes()
            .OfType<LocalDeclarationStatementSyntax>();
    foreach (var varDeclaration in variableDeclarations)
    {
        var symbolInfo = semanticModel.GetSymbolInfo(varDeclaration.Declaration.Type);
        var typeSymbol = symbolInfo.Symbol; // the type symbol for the variable..
    }
}
```

Analysieren Sie den Quellcode mit Roslyn online lesen:
https://riptutorial.com/de/roslyn/topic/4712/analysieren-sie-den-quellcode-mit-roslyn

Kapitel 3: Arbeitsbereiche verwenden

Einführung

Der Arbeitsbereich ist eine programmatische Darstellung der C # - Hierarchie, die aus einer Lösung, untergeordneten Projekten und untergeordneten Dokumenten besteht.

Bemerkungen

- Derzeit gibt es keinen MSBuild-Arbeitsbereich, der Projekte unterstützt, die mit .NET Standard kompatibel sind. Weitere Informationen finden Sie hier .

Examples

Erstellen eines AdhocWorkspace und Hinzufügen eines neuen Projekts und einer Datei.

Die Idee hinter `AdhocWorkspace` ist die Erstellung eines Arbeitsbereichs im `AdhocWorkspace` .

```
var workspace = new AdhocWorkspace();

string projectName = "HelloWorldProject";
ProjectId projectId = ProjectId.CreateNewId();
VersionStamp versionStamp = VersionStamp.Create();
ProjectInfo helloWorldProject = ProjectInfo.Create(projectId, versionStamp, projectName,
projectName, LanguageNames.CSharp);
SourceText sourceText = SourceText.From("class Program { static void Main() {
System.Console.WriteLine(\"HelloWorld\"); } }");

Project newProject = workspace.AddProject(helloWorldProject);
Document newDocument = workspace.AddDocument(newProject.Id, "Program.cs", sourceText);
```

Erstellen eines MSBuildWorspace, Laden einer Lösung und Abrufen aller Dokumente in der gesamten Lösung

Der `MSBuildWorspace` basiert auf dem Konzept der Handhabung von MSBuild-Lösungen (`.sln` Dateien) und ihrer jeweiligen Projekte (`.csproj` , `.vbproj`). Das Hinzufügen neuer Projekte und Dokumente zu diesem Arbeitsbereich wird nicht unterstützt.

```
string solutionPath = @"C:\Path\To\Solution\Sample.sln";

MSBuildWorkspace workspace = MSBuildWorkspace.Create();
Solution solution = await workspace.OpenSolutionAsync(nancyApp);

var allDocumentsInSolution = solution.Projects.SelectMany(x => x.Documents);
```

Abrufen des VisualStudioWorkspace aus einer Visual Studio-Erweiterung

Im Gegensatz zu den anderen Arten von Arbeitsbereichen kann der `VisualStudioWorkspace` nicht manuell erstellt werden. Sie können darauf zugreifen, wenn Sie eine Visual Studio-Erweiterung erstellen.

`[YourVSPackage]Package.cs` in Ihrem Erweiterungspaketprojekt zur Datei `[YourVSPackage]Package.cs` . Dort können Sie den Arbeitsbereich auf zwei Arten erwerben:

```
protected override void Initialize()
{
    // Additional code...

    var componentModel = (IComponentModel)this.GetService(typeof(SComponentModel));
    var workspace = componentModel.GetService<VisualStudioWorkspace>();
}
```

Oder mit MEF:

```
[Import(typeof(VisualStudioWorkspace))]
public VisualStudioWorkspace ImportedWorkspace { get; set; }
```

Ein großartiges Video-Tutorial zum `VisualStudioWorkspace` finden Sie hier .

Arbeitsbereiche verwenden online lesen:
https://riptutorial.com/de/roslyn/topic/9755/arbeitsbereiche-verwenden

Kapitel 4: Quellcode mit Roslyn ändern

Einführung

Praktische Beispiele für die Verwendung von Roslyn für Quelltextumwandlungen.

Bemerkungen

- Roslyn-Syntaxbäume sind unveränderlich. Durch Aufrufen einer Methode wie ReplaceNodes generieren wir einen neuen Knoten, anstatt den vorhandenen zu ändern. Dazu müssen Sie immer das Objekt ändern, an dem Sie gearbeitet haben.

Examples

Ersetzen Sie vorhandene Attribute für alle Methoden in C # mithilfe des Syntaxbaums

Das folgende Snippet ersetzt alle Attribute mit dem Namen `PreviousAttribute` durch ein Attribut mit dem Namen `ReplacementAttribute` für eine gesamte Lösung. Das Beispiel durchsucht manuell den Syntaxbaum und ersetzt alle betroffenen Knoten.

```
    static async Task<bool> ModifySolution(string solutionPath)
    {
        using (var workspace = MSBuildWorkspace.Create())
        {
            // Selects a Solution File
            var solution = await workspace.OpenSolutionAsync(solutionPath);
            // Iterates through every project
            foreach (var project in solution.Projects)
            {
                // Iterates through every file
                foreach (var document in project.Documents)
                {
                    // Selects the syntax tree
                    var syntaxTree = await document.GetSyntaxTreeAsync();
                    var root = syntaxTree.GetRoot();
                    // Finds all Attribute Declarations in the Document
                    var existingAttributesList =
root.DescendantNodes().OfType<AttributeListSyntax>()
                        // Where the Attribute is declared on a method
                        .Where(curr => curr.Parent is MethodDeclarationSyntax)
                        // And the attribute is named "PreviousAttribute"
                        .Where(curr => curr.Attributes.Any(currentAttribute =>
currentAttribute.Name.GetText().ToString() == "PreviousAttribute"))
                        .ToList();
                    if (existingAttributesList.Any())
                    {
                        // Generates a replacement for every attribute
                        var replacementAttribute = SyntaxFactory.AttributeList(
                            SyntaxFactory.SingletonSeparatedList(
```

```
SyntaxFactory.Attribute(SyntaxFactory.IdentifierName("ReplacementAttribute"),
                                SyntaxFactory.AttributeArgumentList(
                                    SyntaxFactory.SeparatedList(new[]
                                    {
                                    SyntaxFactory.AttributeArgument(
                                        SyntaxFactory.LiteralExpression(
                                            SyntaxKind.StringLiteralExpression,
SyntaxFactory.Literal(@"Sample"))
                                        )
                                    })))));
                        // Replaces all attributes at once.
                        // Note that you should not use root.ReplaceNode
                        // since it would only replace the first note
                        root = root.ReplaceNodes(existingAttributesList, (node, n2) =>
replacementAttribute);
                        // Exchanges the document in the solution by the newly generated
document
                        solution = solution.WithDocumentSyntaxRoot(document.Id, root);
                    }
                }
            }
            // applies the changes to the solution
            var result = workspace.TryApplyChanges(solution);
            return result;
        }
    }
```

Das obige Beispiel kann für die folgende Klasse getestet werden:

```
public class Program
{
    [PreviousAttribute()]
    static void Main(string[] args)
    {
    }
}
```

Sie sollten die Methode root.ReplaceNode nicht verwenden, um mehrere Knoten zu ersetzen. Da der Baum unveränderlich ist, werden Sie an verschiedenen Objekten arbeiten. Die Verwendung des folgenden Snippets im obigen Beispiel würde nicht das erwartete Ergebnis liefern:

```
foreach(var node in existingAttributesList){
    root = root.ReplaceNode(node, replacementAttribute);
}
```

Beim ersten Aufruf von `ReplaceNode` ein neues `ReplaceNode` erstellt. Die Elemente in `existingAttributesList` gehören jedoch zu einem anderen Stamm (dem vorherigen Stammelement) und können daher nicht ersetzt werden. Dies würde dazu führen, dass das erste Attribut ersetzt wird und die folgenden Attribute unverändert bleiben, da alle aufeinanderfolgenden Aufrufe auf einem Knoten ausgeführt werden, der nicht im neuen Baum vorhanden ist.

Ersetzen Sie vorhandene Attribute für alle Methoden in C # mit einem SyntaxRewriter

Der folgende Ausschnitt ersetzt alle Attribute mit dem Namen "PreviousAttribute" durch ein Attribut mit dem Namen "ReplacementAttribute" für eine gesamte Lösung. Das Beispiel verwendet manuell einen SyntaxRewriter, um die Attribute auszutauschen.

```
/// <summary>
/// The CSharpSyntaxRewriter allows to rewrite the Syntax of a node
/// </summary>
public class AttributeStatementChanger : CSharpSyntaxRewriter
{
    /// Visited for all AttributeListSyntax nodes
    /// The method replaces all PreviousAttribute attributes annotating a method by
ReplacementAttribute attributes
    public override SyntaxNode VisitAttributeList(AttributeListSyntax node)
    {
        // If the parent is a MethodDeclaration (= the attribute annotes a method)
        if (node.Parent is MethodDeclarationSyntax &&
            // and if the attribute name is PreviousAttribute
            node.Attributes.Any(
                currentAttribute => currentAttribute.Name.GetText().ToString() ==
"PreviousAttribute"))
        {
            // Return an alternate node that is injected instead of the current node
            return SyntaxFactory.AttributeList(
                            SyntaxFactory.SingletonSeparatedList(

SyntaxFactory.Attribute(SyntaxFactory.IdentifierName("ReplacementAttribute"),
                                SyntaxFactory.AttributeArgumentList(
                                    SyntaxFactory.SeparatedList(new[]
                                    {
                                    SyntaxFactory.AttributeArgument(
                                        SyntaxFactory.LiteralExpression(
                                            SyntaxKind.StringLiteralExpression,
SyntaxFactory.Literal(@"Sample"))
                                        )
                                    }))))));
        }
        // Otherwise the node is left untouched
        return base.VisitAttributeList(node);
    }
}

/// The method calling the Syntax Rewriter
private static async Task<bool> ModifySolutionUsingSyntaxRewriter(string solutionPath)
{
    using (var workspace = MSBuildWorkspace.Create())
    {
        // Selects a Solution File
        var solution = await workspace.OpenSolutionAsync(solutionPath);
        // Iterates through every project
        foreach (var project in solution.Projects)
        {
            // Iterates through every file
            foreach (var document in project.Documents)
            {
                // Selects the syntax tree
                var syntaxTree = await document.GetSyntaxTreeAsync();
                var root = syntaxTree.GetRoot();

                // Generates the syntax rewriter
                var rewriter = new AttributeStatementChanger();
```

```
                root = rewriter.Visit(root);

                // Exchanges the document in the solution by the newly generated document
                solution = solution.WithDocumentSyntaxRoot(document.Id, root);
            }
        }
        // applies the changes to the solution
        var result = workspace.TryApplyChanges(solution);
        return result;
    }
}
```

Das obige Beispiel kann für die folgende Klasse getestet werden:

```
public class Program
{
    [PreviousAttribute()]
    static void Main(string[] args)
    {
    }
}
```

Quellcode mit Roslyn ändern online lesen: https://riptutorial.com/de/roslyn/topic/5221/quellcode-mit-roslyn-andern

Kapitel 5: Semantisches Modell

Einführung

Im Gegensatz zur Syntax-API, die alle Arten von Informationen auf Syntaxebene bereitstellt, gibt das semantische Modell unserem Code mehr "Bedeutung" und ermöglicht uns die Beantwortung von Fragen wie "Welche Namen sind an diesem Ort?" diese Methode? "," Welche Variablen werden in diesem Textblock verwendet? "," Worauf bezieht sich dieser Name / Ausdruck? ".

Bemerkungen

- Das Abfragen des semantischen Modells ist teurer als das Abfragen des Syntaxbaums, da am häufigsten eine Zusammenstellung ausgelöst wird.

Examples

Das semantische Modell erhalten

Es gibt einige Möglichkeiten, um das Sematic-Modell zu erhalten.

- Aus einer `Document`

```
Document document = ...;
SemanticModel semanticModel = await document.GetSemanticModelAsync();
```

- Aus einer `Compilation`

```
CSharpCompilation compilation = ...;
var semanticModel = await compilation.GetSemanticModel(syntaxTree);
```

- Aus einem `AnalysisContext` . Für ein Beispiel in einem `DiagnosticAnalyzer` Sie Folgendes tun:

```
public override void Initialize(AnalysisContext context)
{
    context.RegisterSemanticModelAction(x =>
    {
        var semanticModel = x.SemanticModel;
        // Do magical magic here.
    });
}
```

Holen Sie sich alle Verweise auf eine Methode

```
var syntaxRoot = await document.GetSyntaxRootAsync();

var semanticModel = await document.GetSemanticModelAsync();
```

```
var sampleMethodInvocation = syntaxRoot
    .DescendantNodes()
    .OfType<InvocationExpressionSyntax>()
    .First();

var sampleMethodSymbol = semanticModel.GetSymbolInfo(sampleMethodInvocation).Symbol;
var referencesToSampleMethod = await SymbolFinder.FindReferencesAsync(sampleMethodSymbol,
document.Project.Solution);
```

Semantisches Modell online lesen: https://riptutorial.com/de/roslyn/topic/9772/semantisches-modell

Kapitel 6: Syntaxbaum

Einführung

Ein wesentlicher Bestandteil des Roslyn-Compilers ist die Syntax-API. Es macht die Syntaxbäume sichtbar, mit denen die Compiler Visual Basic- und C # -Programme verstehen.

Bemerkungen

- Der Syntaxbaum ist ein Parse-Baum im Kontext des Roslyn-Compilers.

Examples

Den Syntaxbaumstamm aus einem Dokument abrufen.

Wenn Sie bereits über Ihren Arbeitsbereich (Using Workspaces) auf Ihre `Document` Klasse zugreifen können, können Sie einfach auf das Stammverzeichnis Ihres Syntaxbaums zugreifen.

```
Document document = ... // Get document from workspace or other source

var syntaxRoot = await document.GetSyntaxRootAsync();
```

Durchlaufen des Syntaxbaums mit LINQ

Mit LINQ können Sie leicht durch einen Syntaxbaum navigieren. Zum Beispiel ist es einfach, alle `ClassDeclarationSyntax` Knoten (deklarierte Klassen) zu erhalten, deren Namen mit dem Buchstaben `A` :

```
var allClassesWithNameStartingWithA = syntaxRoot.DescendantNodes()
    .OfType<ClassDeclarationSyntax>()
    .Where(x => x.Identifier.ToString().StartsWith("A"));
```

Oder alle Klassen mit Attributen abrufen:

```
var allClassesWithAttriutes = syntaxRoot.DescendantNodes()
    .OfType<ClassDeclarationSyntax>()
    .Where(x => x.AttributeLists.Any(y => y.Attributes.Any()));
```

Durchlaufen des Syntaxbaums mit einem CSharpSyntaxWalker

Die `CSharpSyntaxWalker` Klasse ist die `CSharpSyntaxWalker` Implementierung des Besuchermusters, mit der wir unseren `CSharpSyntaxWalker` durchlaufen können. Hier ist ein einfaches Beispiel für einen Syntax Walker, der alle `struct` -s sammelt, die einen Namen haben, beginnend mit dem Buchstaben `A` :

```
public class StructCollector : CSharpSyntaxWalker
{
    public StructCollector()
    {
        this.Structs = new List<StructDeclarationSyntax>();
    }

    public IList<StructDeclarationSyntax> Structs { get; }

    public override void VisitStructDeclaration(StructDeclarationSyntax node)
    {
        if (node.Identifier.ToString().StartsWith("A"))
        {
            this.Structs.Add(node);
        }
    }
}
```

Wir können unseren SyntaxWalker folgendermaßen verwenden:

```
var structCollector = new StructCollector();
structCollector.Visit(syntaxRoot); // Or any other syntax node
Console.WriteLine($"The number of structs that have a name starting with the letter 'A' is
{structCollector.Structs.Count}");
```

Syntaxbaum online lesen: https://riptutorial.com/de/roslyn/topic/9765/syntaxbaum

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Roslyn	Community, Teodor Kurtev
2	Analysieren Sie den Quellcode mit Roslyn	andyp, Michael Rätzel, SJP, Stefano d'Antonio
3	Arbeitsbereiche verwenden	Teodor Kurtev
4	Quellcode mit Roslyn ändern	SJP, Teodor Kurtev
5	Semantisches Modell	Teodor Kurtev
6	Syntaxbaum	Teodor Kurtev