Roslyn: мастерство статического анализа

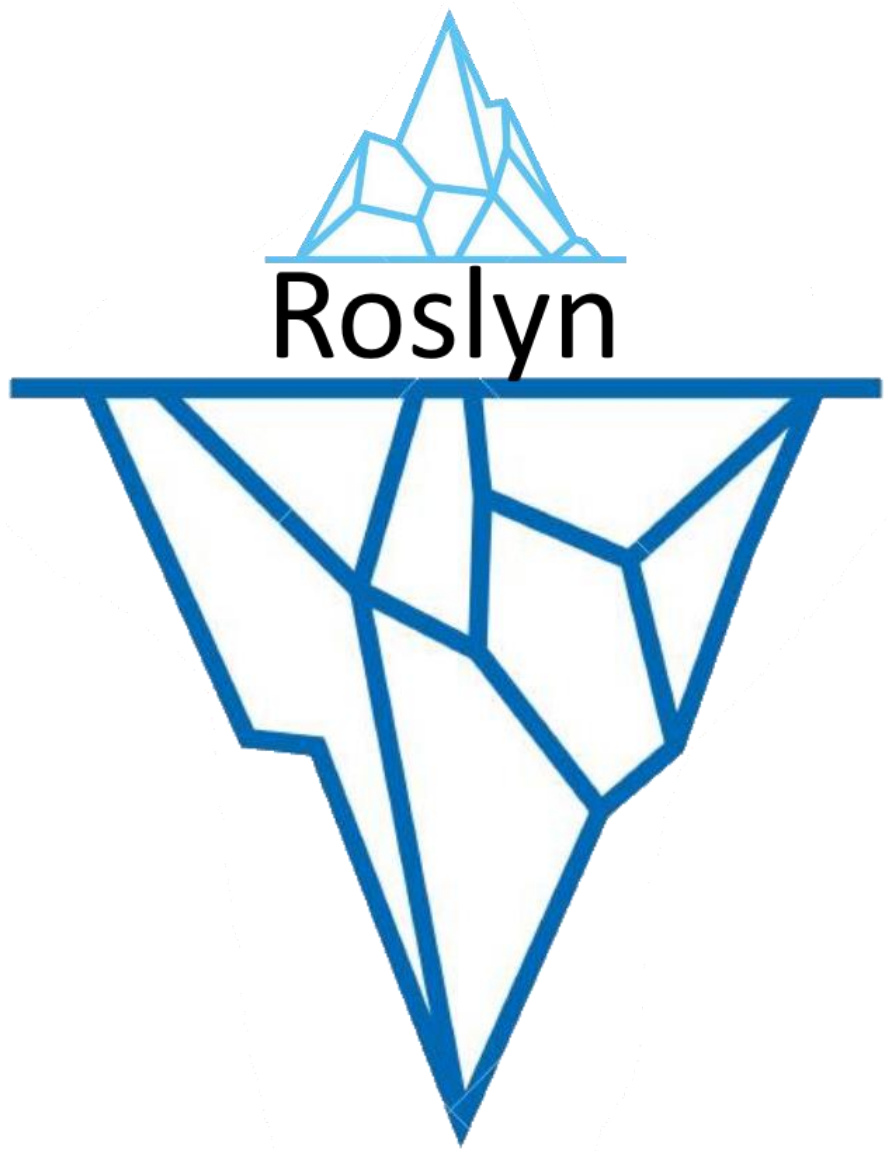
Владимир Панченко

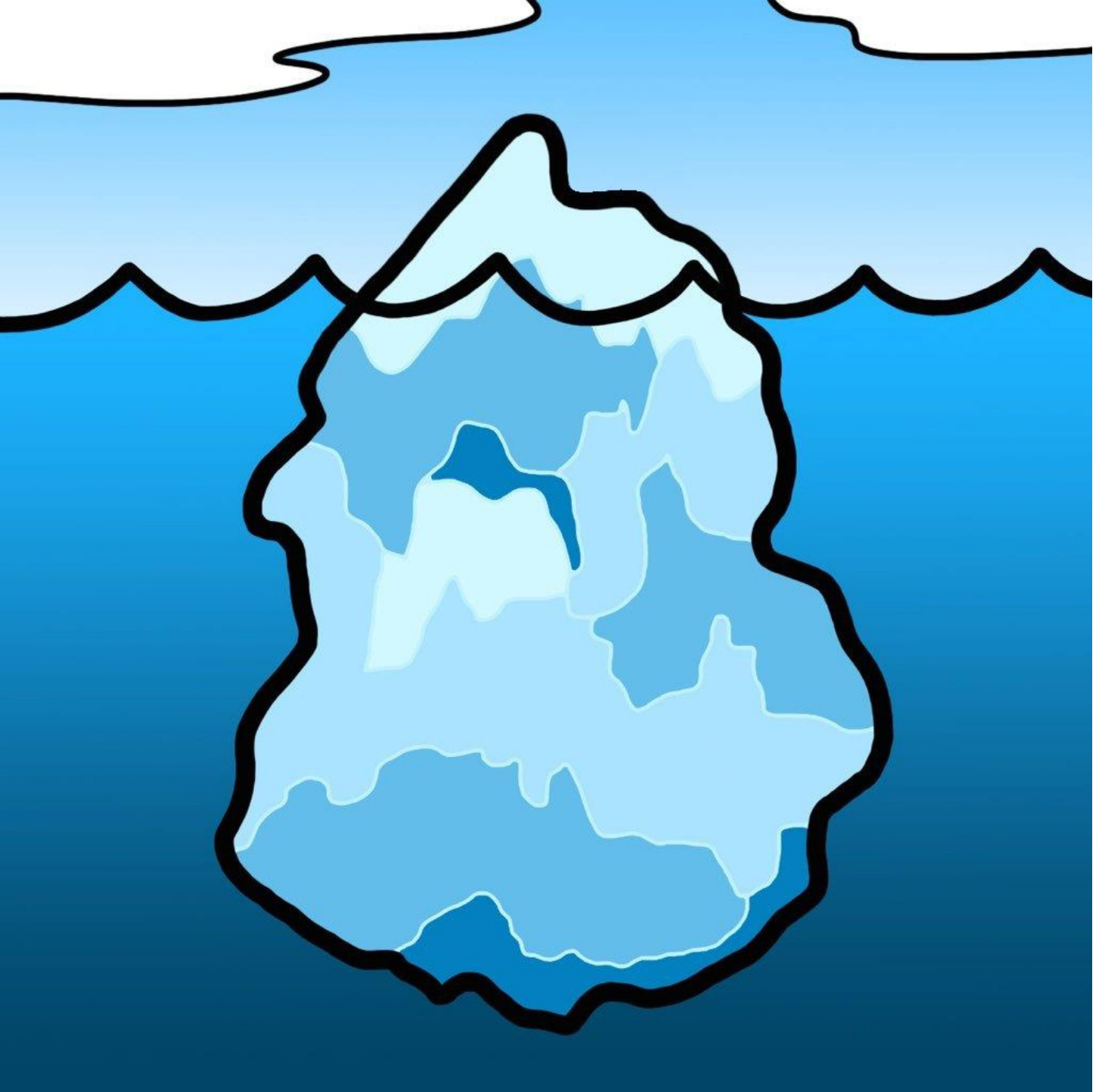
dropsonic.pan@gmail.com

Обо мне

- Team Lead в **Acumatica**
- **7** лет в коммерческой разработке на .NET
- **1.5** года в .NET Core и IoT
- **2** года в гражданском браке с **Roslyn**


roslyn


Roslyn



Примеры из документации и блогов

Подводные камни

О чём я буду рассказывать

- **«Подводные камни» при работе с Roslyn за пределами “Hello World!”**

О чём я буду рассказывать

- «Подводные камни» при работе с Roslyn за пределами “Hello World!”
- **Как полноценно интегрировать статический анализ на Roslyn в процесс разработки (работа с legacy кодом, интеграция с CI, etc.)**

О чём я буду рассказывать

- «Подводные камни» при работе с Roslyn за пределами “Hello World!”
- Как полноценно интегрировать статический анализ на Roslyn в процесс разработки (работа с legacy кодом, интеграция с CI, etc.)
- **Как ещё более ускорить процесс разработки, используя Roslyn для рефакторинга, подсветки синтаксиса, форматирования, навигации, создания своего IntelliSense и т.п.**

Чего мы достигли

70 **статических анализаторов**

28 **code fix’ов**

1600 **пользователей**

</> **Подсветка синтаксиса**

{…} **Форматирование кода**

Навигация по коду

± **Collapse / Expand**

Примеры

https://github.com/dropsonic/dotNext2019

Как всё начиналось

Исходные данные

2 000 000+ LoC на **C#**

Большой и сложный framework

Naming Conventions

80+ разработчиков внутри компании

450+ компаний-партнёров

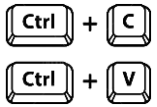
140+ сторонних решений в marketplace

Проблемы

Naming Conventions

Проблемы

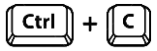
Naming Conventions

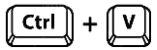


Ошибки типа copy&paste

Проблемы

Naming Conventions



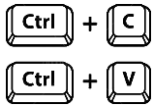


Ошибки типа copy&paste

Сложно запомнить все правила фреймворка

Проблемы

Naming Conventions



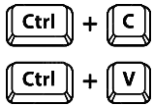
Ошибки типа copy&paste

Сложно запомнить все правила фреймворка

Невозможно задать их на уровне языка

Проблемы

Naming Conventions

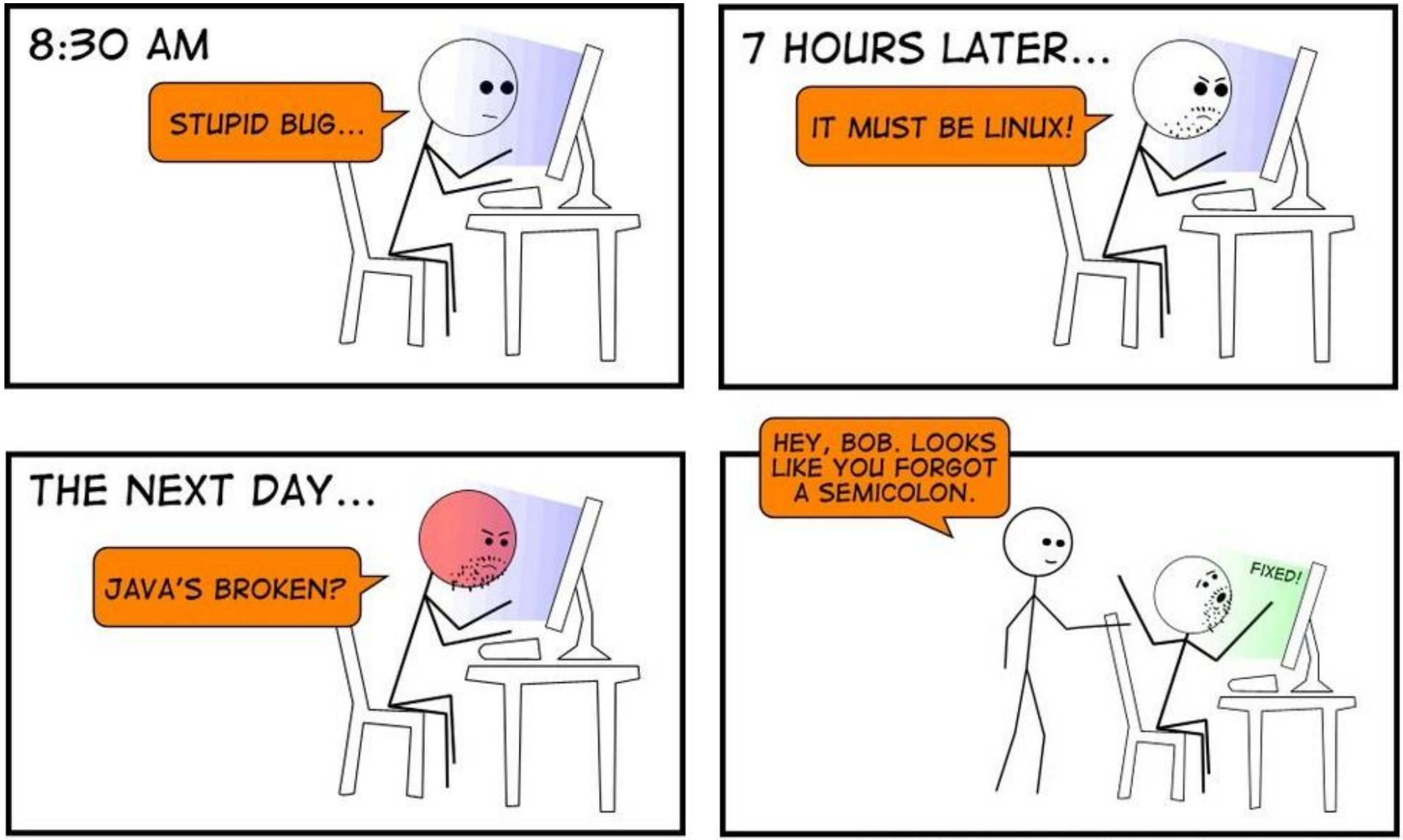


Ошибки типа copy&paste

Сложно запомнить все правила фреймворка

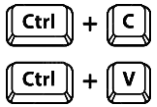
Невозможно задать их на уровне языка

Трата времени на дебаг


8:30 AM
STUPID BUG...
7 HOURS LATER...
IT MUST BE LINUX!
THE NEXT DAY...
JAVA'S BROKEN?
HEY, BOB. LOOKS LIKE YOU FORGOT A SEMICOLON.
FIXED!
STUFFTHATHAPPENS.COM BY ERIC BURKE

Проблемы

Naming Conventions



Ошибки типа copy&paste

Сложно запомнить все правила фреймворка

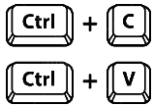
Невозможно задать их на уровне языка

Трата времени на дебаг

Много «внешних» разработчиков

Проблемы

Naming Conventions



Ошибки типа copy&paste

Сложно запомнить все правила фреймворка

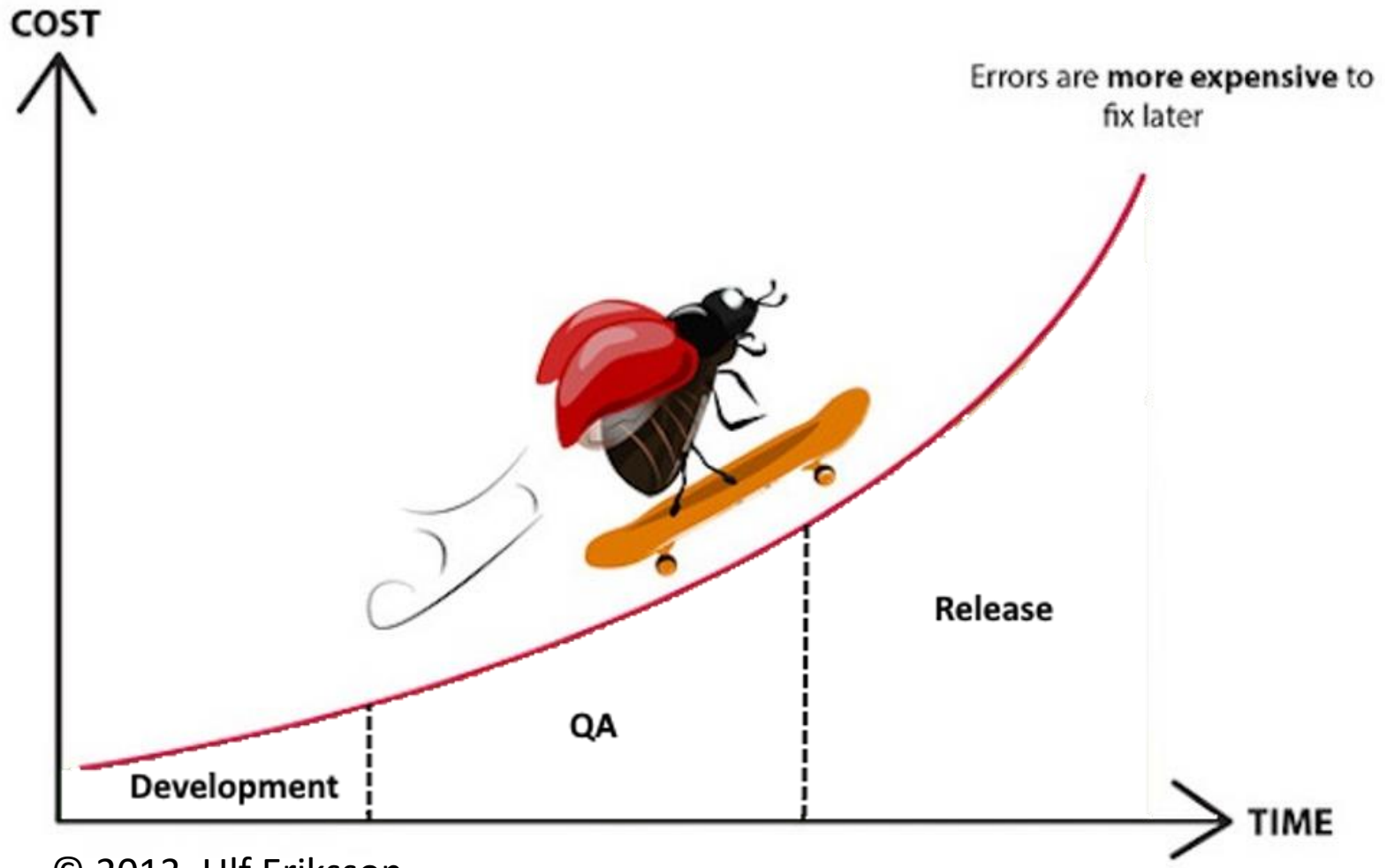
Невозможно задать их на уровне языка

Трата времени на дебаг

Много «внешних» разработчиков

«Ручная» сертификация решений партнёров

Чем раньше, тем лучше




NIGHT GATHERS, AND NOW MY_
seasonsCount = 8; // Global
var groupsCount = 2; // Global
(function(){
pagesCount = 3; // Global
var postsCount = 8; // Local
})()
- THE NIGHT'S WATCH OATH
ACUMATICA
HACKATHON
SORRY LADIES
I'M IN THE
NIGHT'S
WATCH

Как всё начиналось

За 24 часа мы написали:

- **12** анализаторов
- **8** code fix’ов
- Подсветку синтаксиса для DSL

Как тестировать analyzers и code fixes

throw in Dispose

```
class Foo : IDisposable
{
    public void Dispose()
    {
        throw new Exception();
    }
}
```

throw in Dispose

```
class Foo : IDisposable
{
    public void Dispose()
    {
        var e = new Exception();
    }
}
```

Как тестировать analyzers и code fixes

- Много кейсов

Как тестировать analyzers и code fixes

МНОГО КЕЙСОВ!

Тестовый фреймворк

Тестовый фреймворк

1. **Создаёт новый VS solution и project «на лету»**

Тестовый фреймворк

1. Создаёт новый VS solution и project «на лету»
2. **Добавляет в project исходники / файлы конфигурации**

Тестовый фреймворк

1. Создаёт новый VS solution и project «на лету»
2. Добавляет в project исходники / файлы конфигурации
3. **Добавляет в project нужные references (mscorlib, etc., ваши сборки)**

Тестовый фреймворк

1. Создаёт новый VS solution и project «на лету»
2. Добавляет в project исходники / файлы конфигурации
3. Добавляет в project нужные references (mscorlib, etc., ваши сборки)
4. **Применяет анализатор и собирает полученные диагностики**

Тестовый фреймворк

1. Создаёт новый VS solution и project «на лету»
2. Добавляет в project исходники / файлы конфигурации
3. Добавляет в project нужные references (mscorlib, etc., ваши сборки)
4. Применяет анализатор и собирает полученные диагностики
5. **Применяет к каждой диагностике code fix и проверяет его**

Пример

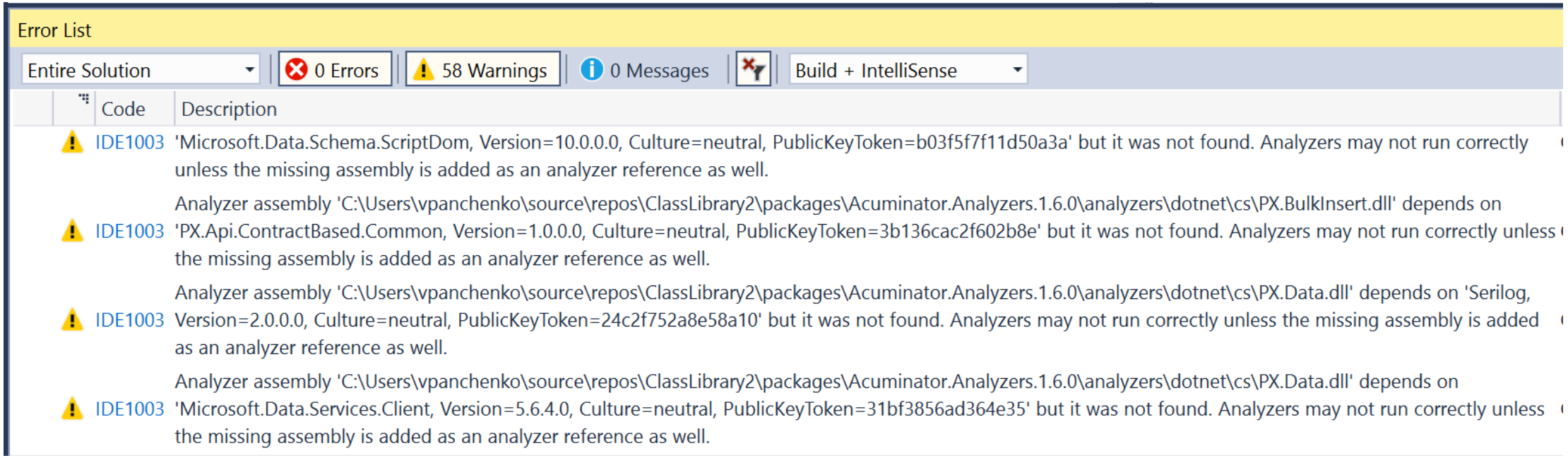
```
public class WhatTheHeckTests : CodeFixVerifier
{
    protected override DiagnosticAnalyzer GetCSharpDiagnosticAnalyzer() => new WhatTheHeckAnalyzer();
    protected override CodeFixProvider GetCSharpCodeFixProvider() => new WhatTheHeckCodeFixProvider();

    [Theory]
    [EmbeddedFileData("PoliteComment.cs")]
    public Task PoliteComment_ShouldNotShowDiagnostic(string actual) =>
        VerifyCSharpDiagnosticAsync(actual);

    [Theory]
    [EmbeddedFileData("ImpoliteComment_StandaloneWord.cs")]
    public Task ImpoliteComment_StandaloneWord_ShouldShowDiagnostic(string actual) =>
        VerifyCSharpDiagnosticAsync(actual, Descriptors.DN1000_WhatTheHeckComment.CreateFor(line: 5, column: 3));
```

Фигак-фигак, и в продакшн!

- 12 **реально работающих** анализаторов, 8 code fix’ов и подсветка синтаксиса для DSL
- Релиз версии 1.0 в **Visual Studio Marketplace** и **NuGet**

Способы поставки нашего творчества

WellKnownTypes

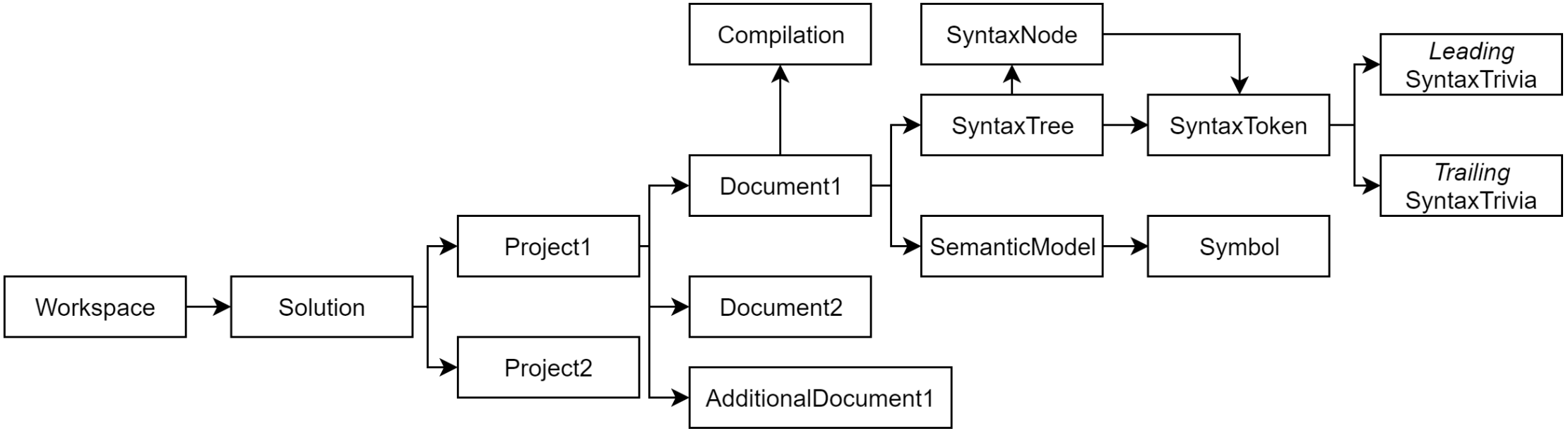


WellKnownTypes

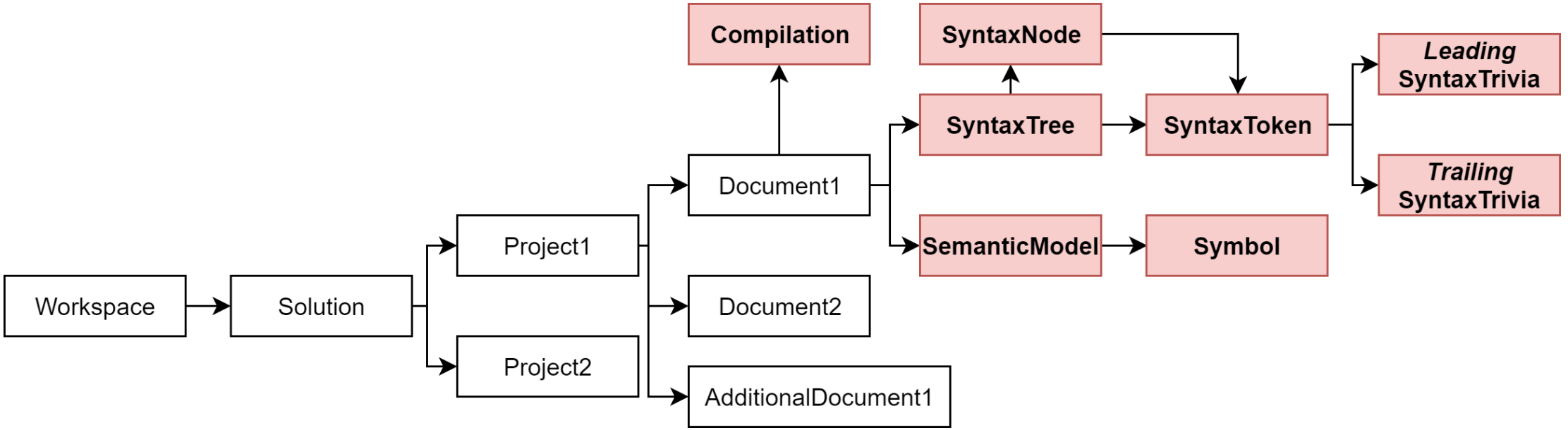
Code: IDE1003

Analyzer assembly MyAnalyzers\analyzers\dotnet\cs\Foo.dll' depends on 'Bar' but it was not found. Analyzers may not run correctly unless the missing assembly is added as an analyzer reference as well.

Немножко теории



Немножко теории



WellKnownTypes

```
public INamedTypeSymbol Foo =>
    _compilation.GetTypeByMetadataName(
        “*ИМЯ_ТИПА*");
```

WellKnownTypes

```
public INamedTypeSymbol Foo =>
    _compilation.GetTypeByMetadataName(
        typeof(Foo).FullName);
```

WellKnownTypes

```
public INamedTypeSymbol Foo =>
    _compilation.GetTypeByMetadataName(
        “MyProject.Foo");
```

Больше анализаторов!

- Пишем всё больше и больше новых анализаторов
- Они становятся всё сложнее...

Рекурсивный анализ кода

```
class Foo : IDisposable
{
    public void Dispose()
    {
        throw new Exception();
    }
}
```

```
public override void Initialize(AnalysisContext context)
{
    context.RegisterCompilationStartAction(compilationStartContext =>
    {
        compilationStartContext.RegisterOperationBlockStartAction(operationBlockContext =>
        {
            if (operationBlockContext.OwningSymbol is IMethodSymbol methodSymbol
                && IsDisposeMethod(methodSymbol))
            {
                operationBlockContext.RegisterOperationAction(operationContext =>
                {
                    operationContext.ReportDiagnostic(
                        Diagnostic.Create(
                            Descriptors.DN1001_ExceptionInDispose,
                            operationContext.Operation.Syntax.GetLocation()));
                }, OperationKind.Throw);
            }
        });
    });
}
```

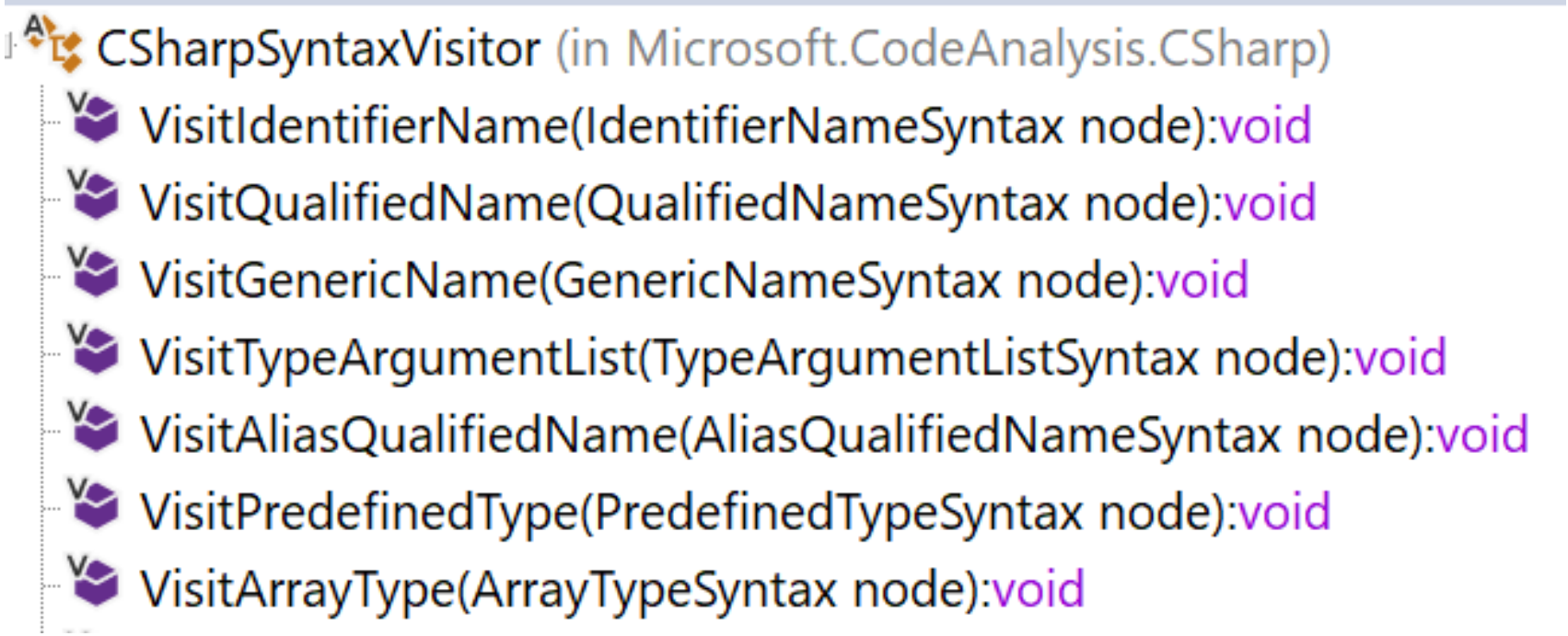
Рекурсивный анализ кода

```
class Foo : IDisposable
{
    public void Dispose()
    {
        throw new Exception();
    }
}
```

Рекурсивный анализ кода

```
class Foo : IDisposable
{
    public void Dispose() {
        if (Ready) ...
    }

    public bool Ready {
        get {
            if (!_initialized) throw new Exception();
        }
    }
}
```

CSharpSyntaxWalker



…и ещё 210+ методов

```
public override void Initialize(AnalysisContext context)
{
    context.RegisterCompilationStartAction(compilationStartContext =>
    {
        compilationStartContext.RegisterSymbolAction(symbolContext =>
        {
            if (symbolContext.Symbol is IMethodSymbol methodSymbol
                && IsDisposeMethod(methodSymbol))
            {
                var methodSyntax = methodSymbol.GetSyntax() as CSharpSyntaxNode;
                methodSyntax?.Accept(new ThrowExceptionWalker(symbolContext,
                    Descriptors.DN1001_ExceptionInDispose));
            }
        }, SymbolKind.Method);
    });
}
```

Рекурсивный анализ кода

```
public override void VisitThrowStatement(
    ThrowStatementSyntax node)
{
    operationContext.ReportDiagnostic(
        Diagnostic.Create(
            Descriptors.DN1001_ExceptionInDispose,
                node.GetLocation()
}
```

Рекурсивный анализ кода

```
class Foo : IDisposable
{
    public void Dispose() {
        if (Ready) ...
    }

    public Ready {
        get {
            if (!_initialized) throw new Exception();
        }
    }
}
```

Рекурсивный анализ кода

```
class Foo : IDisposable
{
    public void Dispose() {
        if (Ready) ...
    }

    public Ready {
        get {
            if (!_initialized) throw new Exception();
        }
    }
}
```

LIVE DEMO

Рекурсивный анализ кода

- **Недоступен «из коробки», требует самостоятельной реализации**

Рекурсивный анализ кода

- Недоступен «из коробки», требует самостоятельной реализации
- **Сильно ухудшает производительность**

Пишем ещё больше анализаторов

- Количество анализаторов подросло до 30+...

Пишем больше анализаторов

- **Много анализаторов, проверяющих одни и те же конструкции в коде**

Пишем больше анализаторов

- Много анализаторов, проверяющих одни и те же конструкции в коде
- **Одна и та же семантическая информация о типах собирается много раз подряд**

Что получили?

Что получили?

- **Проблемы с производительностью**

Что получили?

- Проблемы с производительностью
- **Копипаста одного и того же кода**

```
[ApiController]
[RoutePrefix("foo")]
public class FooController : ControllerBase
{
    [HttpGet, Route]
    public IEnumerable<Foo> Get() => ...

    [HttpGet, Route("{id}")]
    public Foo GetById(int id) => ...

    [HttpPost, Route("{name}")]
    public void Add(string name) => ...

    private Foo Create(string name) => ...
}
```

```
public class ControllerModel
{
    public string RoutePrefix { get; }
    public INamedTypeSymbol Symbol { get; }
    public ImmutableArray<ControllerAction> Actions { get; }
}

public class ControllerAction
{
    public HttpMethod Method { get; }
    public string Route { get; }
    public IMethodSymbol Symbol { get; }
}
```

Агрегирующие анализаторы

- **Один родительский анализатор, который строит семантическую модель определённой сущности**

Агрегирующие анализаторы

- Один родительский анализатор, который строит семантическую модель определённой сущности
- **Запускает несколько дочерних анализаторов, передавая им семантическую модель**

LIVE DEMO

Наступаем на грабли

- Нам потребовался Roslyn SDK v2.0 для использования Operations API в анализаторах

Версионность Roslyn SDK

- **Каждая версия Roslyn SDK требует определённую минимальную версию Visual Studio**

Версионность Roslyn SDK

- Каждая версия Roslyn SDK требует определённую минимальную версию Visual Studio
- **Хотите поддерживать несколько версий Visual Studio? Вы обречены работать со старым SDK**

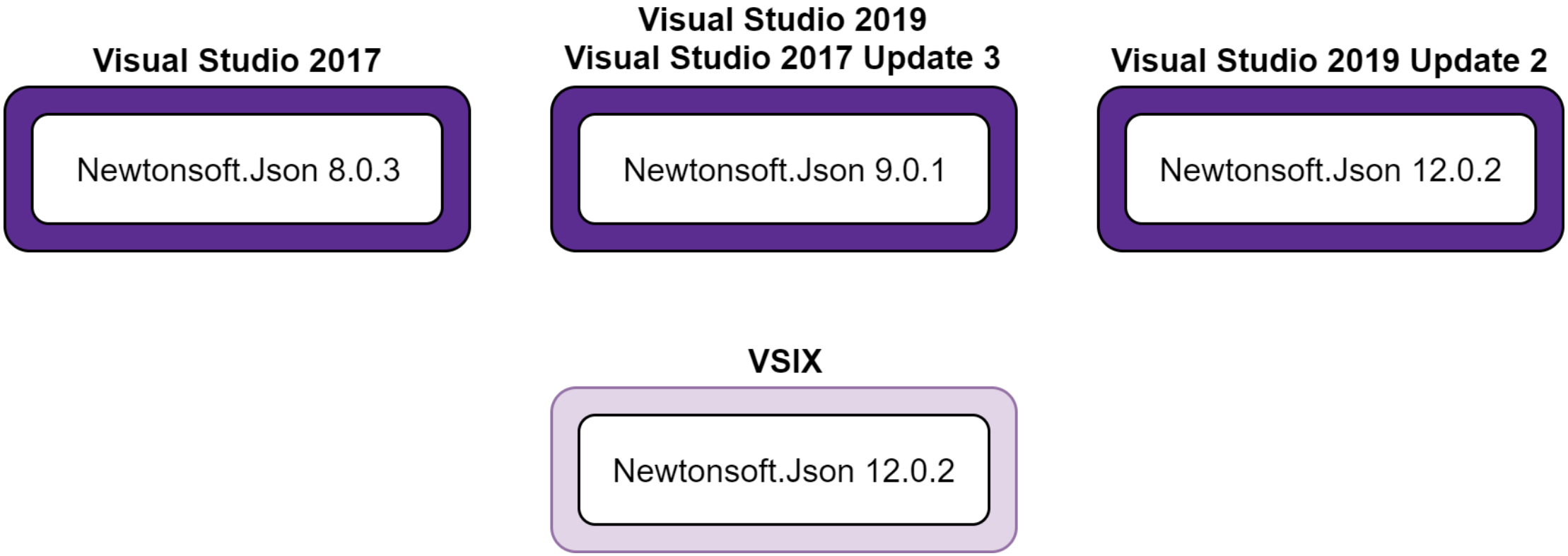
Версионность Roslyn SDK

- Каждая версия Roslyn SDK требует определённую минимальную версию Visual Studio
- Хотите поддерживать несколько версий Visual Studio? Вы обречены работать со старым SDK
- **Поставили новые анализаторы в старую версию Visual Studio? Это не будет работать**

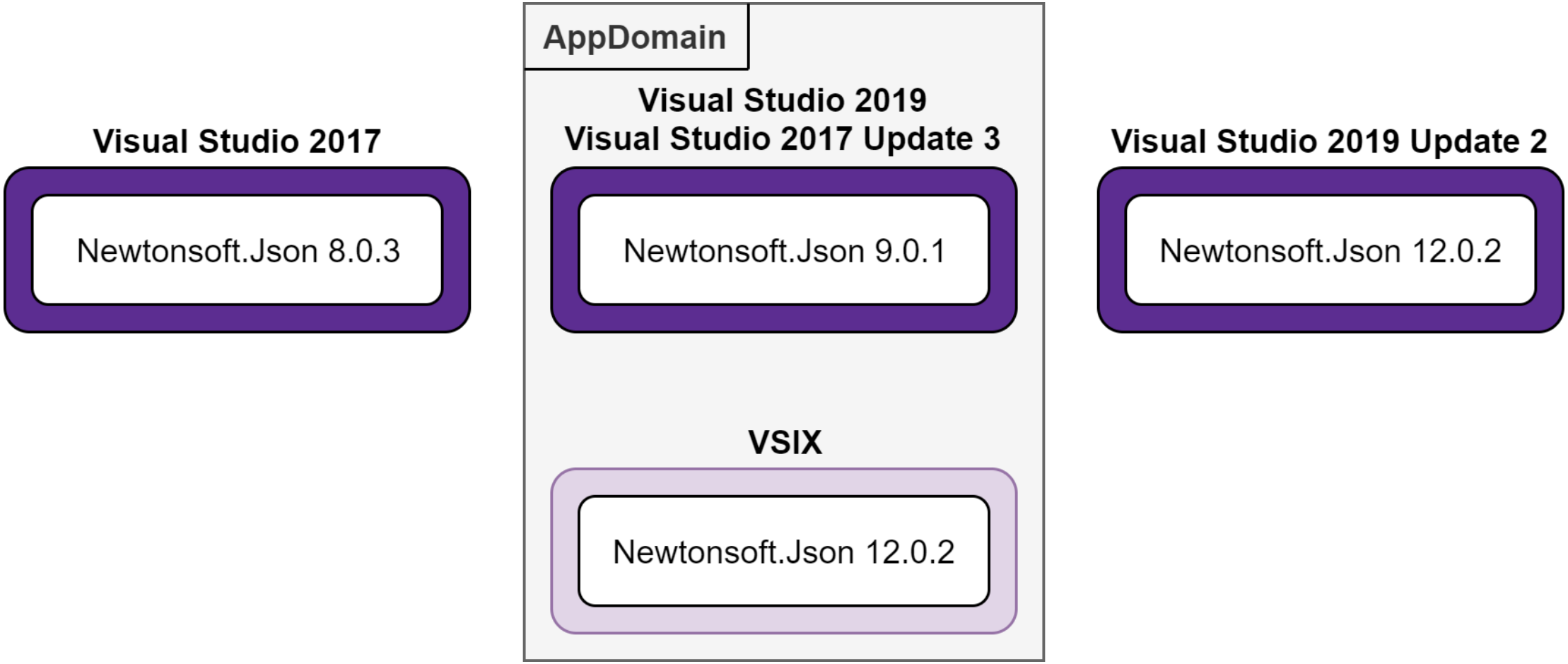
Версионность Roslyn SDK

Версия NuGet-пакетов Roslyn	Версия Visual Studio
3.X	Visual Studio 2019
2.10.0	Visual Studio 2017 Update 9
2.9.0	Visual Studio 2017 Update 8
…	…
2.0.0	Visual Studio 2017 RTM
1.3.2	Visual Studio 2015 Update 3
…	…
1.0.1	Visual Studio 2015 RTM

DLL Hell в Visual Studio



DLL Hell в Visual Studio



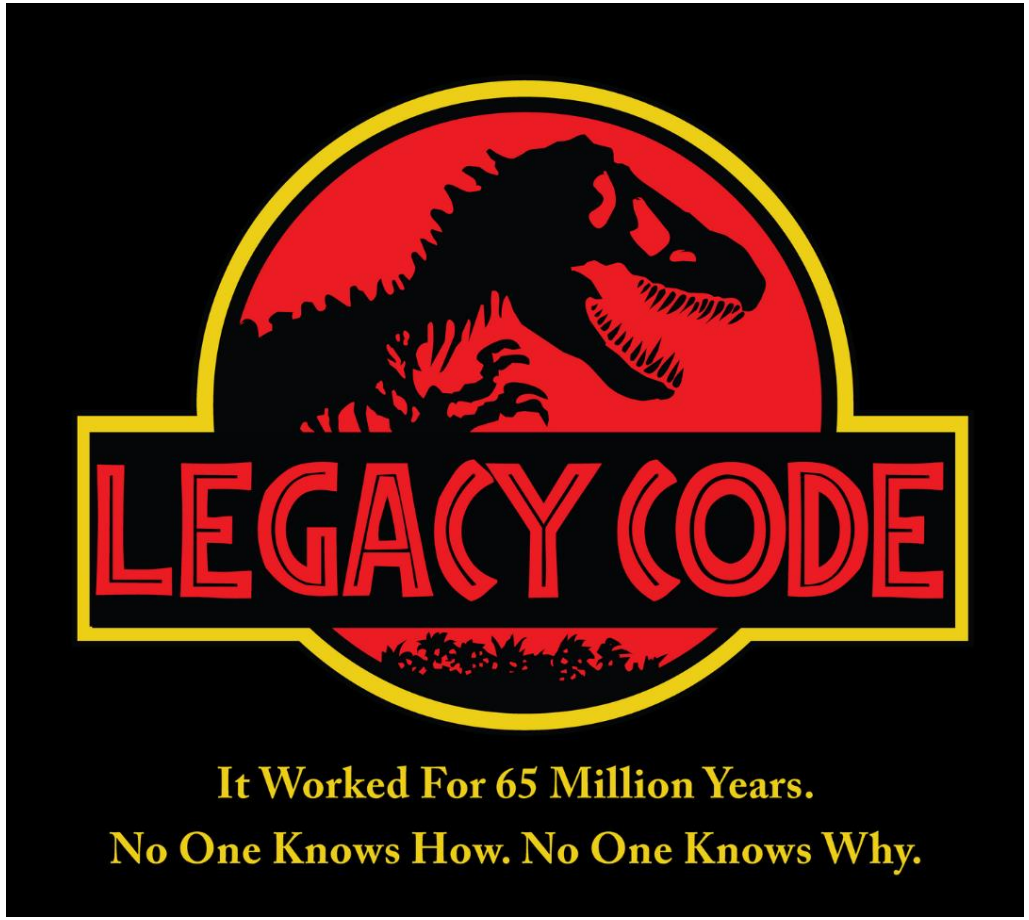
Внедряем статический анализ в процесс разработки и тестирования

Статический анализ в IDE при написании кода

Проверка всей кодовой базы в рамках CI

Сложности

Сложности


LEGACY CODE
It Worked For 65 Million Years.
No One Knows How. No One Knows Why.

Сложности

- Legacy code
- **False-positive срабатывания**

Сложности

- Legacy code
- False-positive срабатывания
- **Необходимость «выключать» диагностики по требованию в определённых местах в коде**

Ruleset

```
<?xml version="1.0" encoding="utf-8"?>
<RuleSet Name="New Rule Set"
         Description=" " ToolsVersion="15.0">
    <Rules AnalyzerId="Microsoft.CodeAnalysis.CSharp"
           RuleNamespace="Microsoft.CodeAnalysis.CSharp">
        <Rule Id="CS0168" Action="Hidden" />
    </Rules>
</RuleSet>
```

Ruleset

- **Позволяет полностью выключать определённые диагностики**

Ruleset

- Позволяет полностью выключать определённые диагностики
- **Можно изменить severity диагностики**

Ruleset

- Позволяет полностью выключать определённые диагностики
- Можно изменить severity диагностики
- **Невозможно «выключить» диагностику в определённом месте в коде**

Pragma

```
        static void Main(string[] args)
        {
#pragma warning disable 219
            int notUsed = 5;
#pragma warning restore 219
        }
```

Pragma

- **Придётся обновлять всю существующую кодовую базу**

Pragma

- Придётся обновлять всю существующую кодовую базу
- **Потеря history / blame**

Pragma

- Придётся обновлять всю существующую кодовую базу
- Потеря history / blame
- **Занимает много места в коде**

Pragma

- Придётся обновлять всю существующую кодовую базу
- Потеря history / blame
- Занимает много места в коде
- **Сильно выделяется по форматированию**

SuppressMessageAttribute

```
[SuppressMessage(
    category: "Compiler",
    checkId: "CS0219")]
private void DoSomething()
{
    int notUsed = 5;
}
```

SuppressMessageAttribute

- **Придётся обновлять всю существующую кодовую базу**

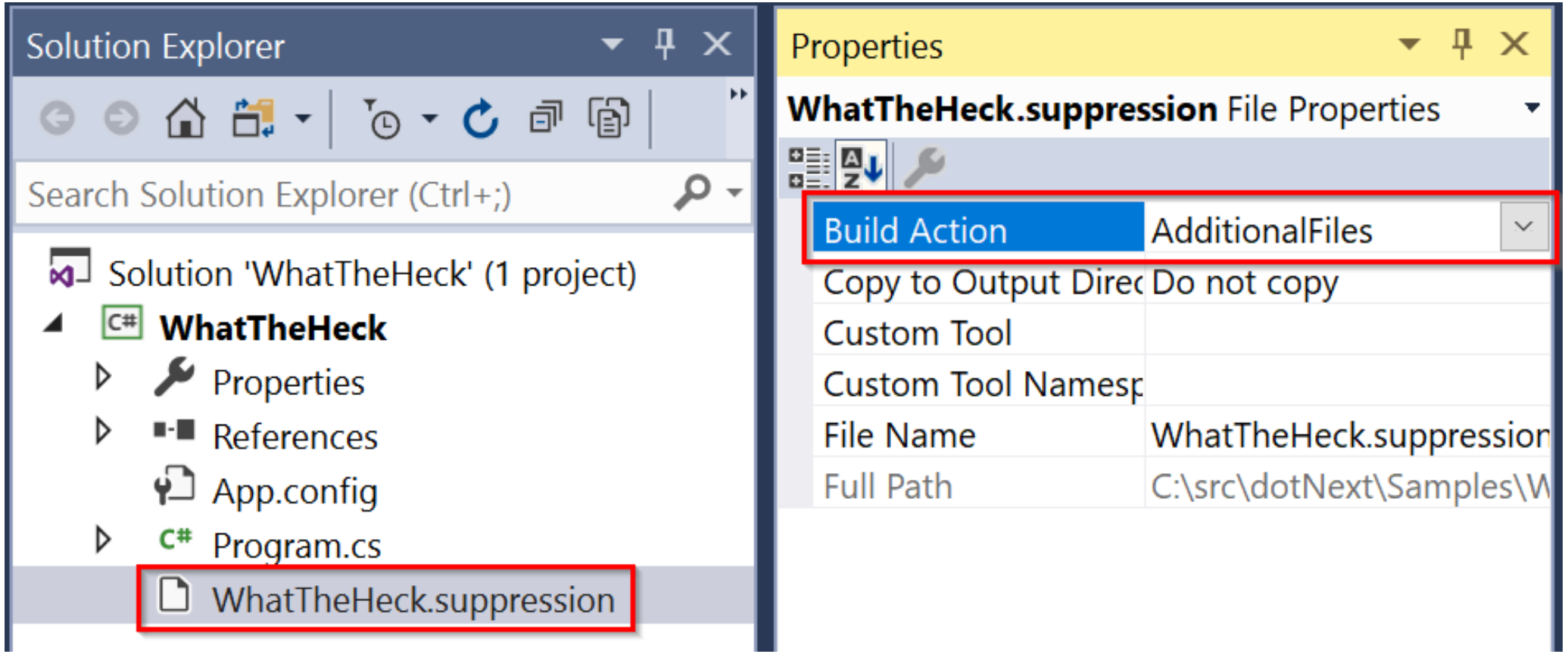
SuppressMessageAttribute

- Придётся обновлять всю существующую кодовую базу
- **Минимальный scope отключения диагностики — метод**

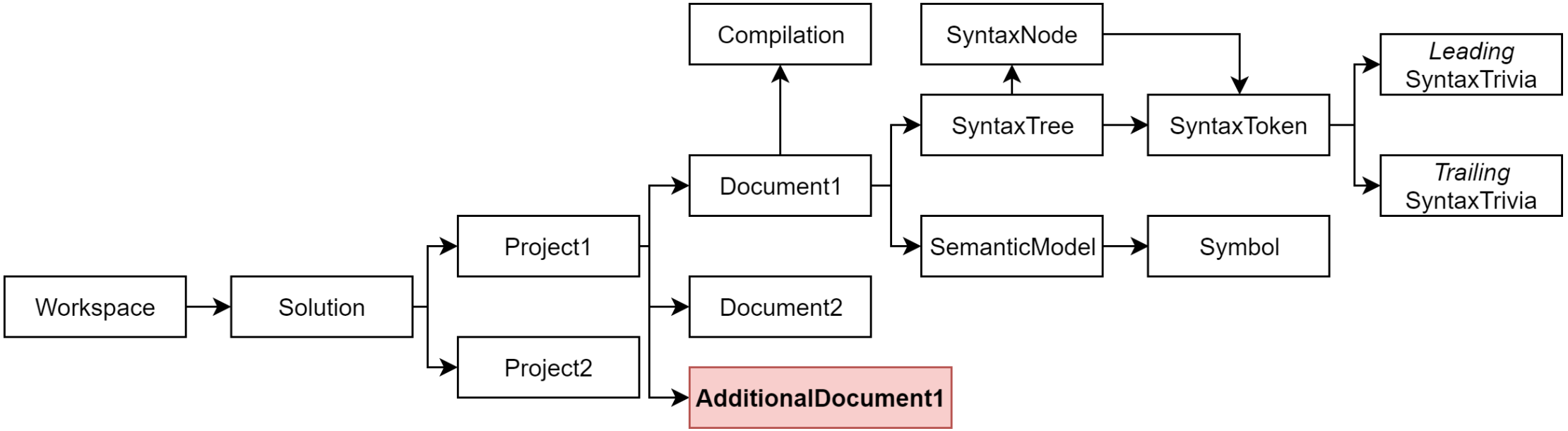
Suppression File

```
<?xml version="1.0" encoding="utf-8"?>
<suppressions>
    <suppressDiagnostic id="DN1234">
        <!-- symbol or syntax node -->
        <context>DotNext.Demo.MyController.Get()</context>
        <!-- token or trivia -->
        <target>FirstOrDefault</target>
    </suppressDiagnostic>
</suppressions>
```

Suppression File



Suppression File



Suppression File

- **Не требует изменения существующего кода**

Suppression File

- Не требует изменения существующего кода
- **Не поддерживается «из коробки», требует реализации с нуля**

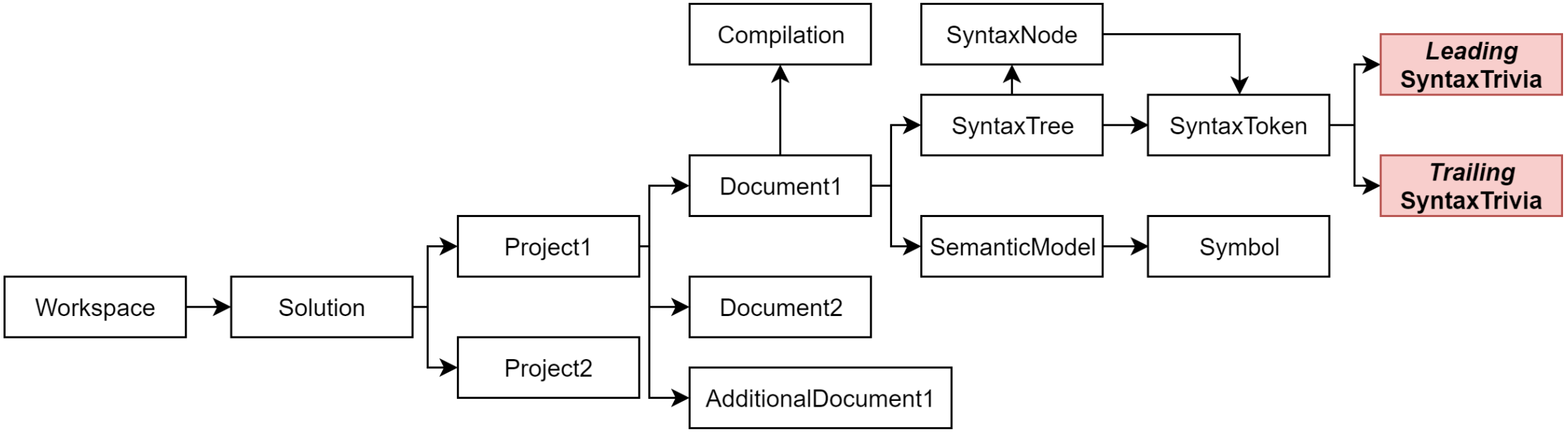
Suppression File

- Не требует изменения существующего кода
- Не поддерживается «из коробки», требует реализации с нуля
- **Низкая наглядность (особенно при code review)**

Code Comments

```
static void Main(string[] args)
{
    // MyExtension disable once DN1001
    Console.WriteLine("Hello dotNext!");
}
```

Code Comments



Code Comments

- **Не занимают много места в коде**

Code Comments

- Не занимают много места в коде
- **Не выделяются из общего форматирования**

Code Comments

- Не занимают много места в коде
- Не выделяются из общего форматирования
- **Высокая наглядность (видны при code review)**

Code Comments

- Не занимают много места в коде
- Не выделяются из общего форматирования
- Высокая наглядность (видны при code review)
- **Требуют изменения существующей кодовой базы**

Code Comments

- Не занимают много места в коде
- Не выделяются из общего форматирования
- Высокая наглядность (видны при code review)
- Требуют изменения существующей кодовой базы
- **Не поддерживаются «из коробки», требует реализации с нуля**

Решение

- Legacy code
- False-positive срабатывания

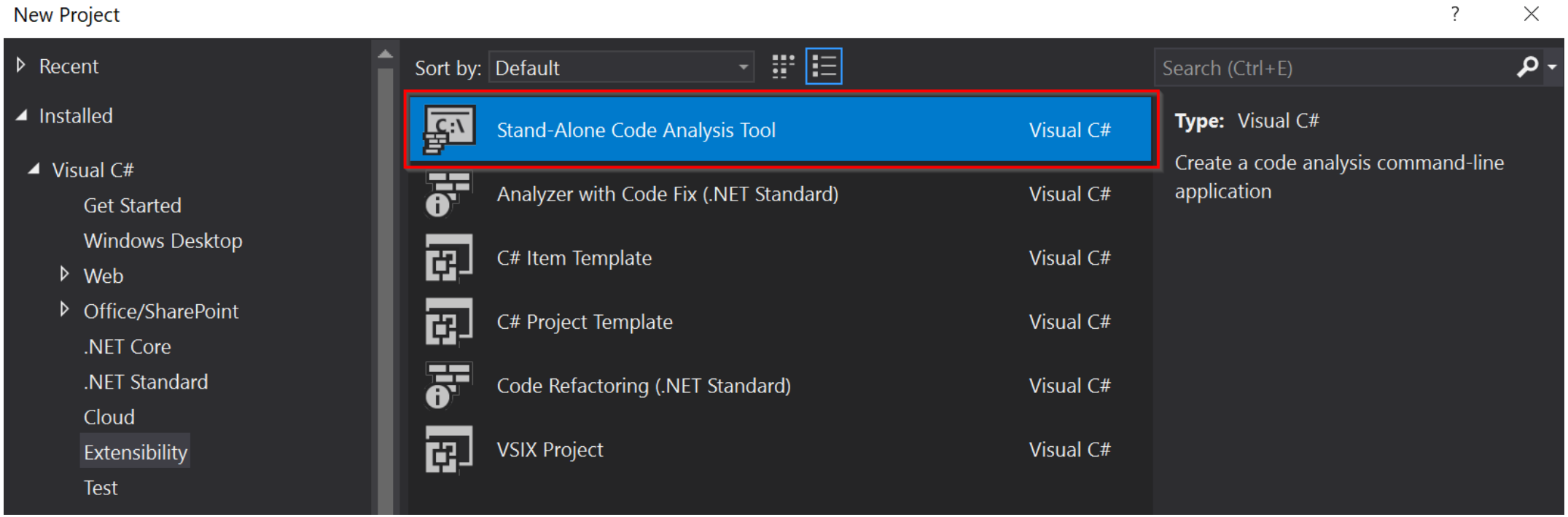
Suppression File

- Необходимость «выключать» диагностики по требованию в определённых местах в коде

Code Comments

LIVE DEMO

Запуск анализаторов «извне» MSBuild и IDE



Что ещё можно делать с помощью Roslyn?

Что ещё можно делать с помощью Roslyn?

- **Рефакторинги**

Что ещё можно делать с помощью Roslyn?

- Рефакторинги
- **Массовое изменение кода в случае обновления API**

Что ещё можно делать с помощью Roslyn?

- Рефакторинги
- Массовое изменение кода в случае обновления API
- **Расширения IntelliSense (например, для DSL)**

Что ещё можно делать с помощью Roslyn?

- Рефакторинги
- Массовое изменение кода в случае обновления API
- Расширения IntelliSense (например, для DSL)
- **Подсветка синтаксиса**

Что ещё можно делать с помощью Roslyn?

- Рефакторинги
- Массовое изменение кода в случае обновления API
- Расширения IntelliSense (например, для DSL)
- Подсветка синтаксиса
- **Навигация по коду**

Что ещё можно делать с помощью Roslyn?

- Рефакторинги
- Массовое изменение кода в случае обновления API
- Расширения IntelliSense (например, для DSL)
- Подсветка синтаксиса
- Навигация по коду
- **Сложное форматирование кода**

Что ещё можно делать с помощью Roslyn?

- Рефакторинги
- Массовое изменение кода в случае обновления API
- Расширения IntelliSense (например, для DSL)
- Подсветка синтаксиса
- Навигация по коду
- Сложное форматирование кода
- **Outlining (collapse / expand)**

Реализуем свой IntelliSense

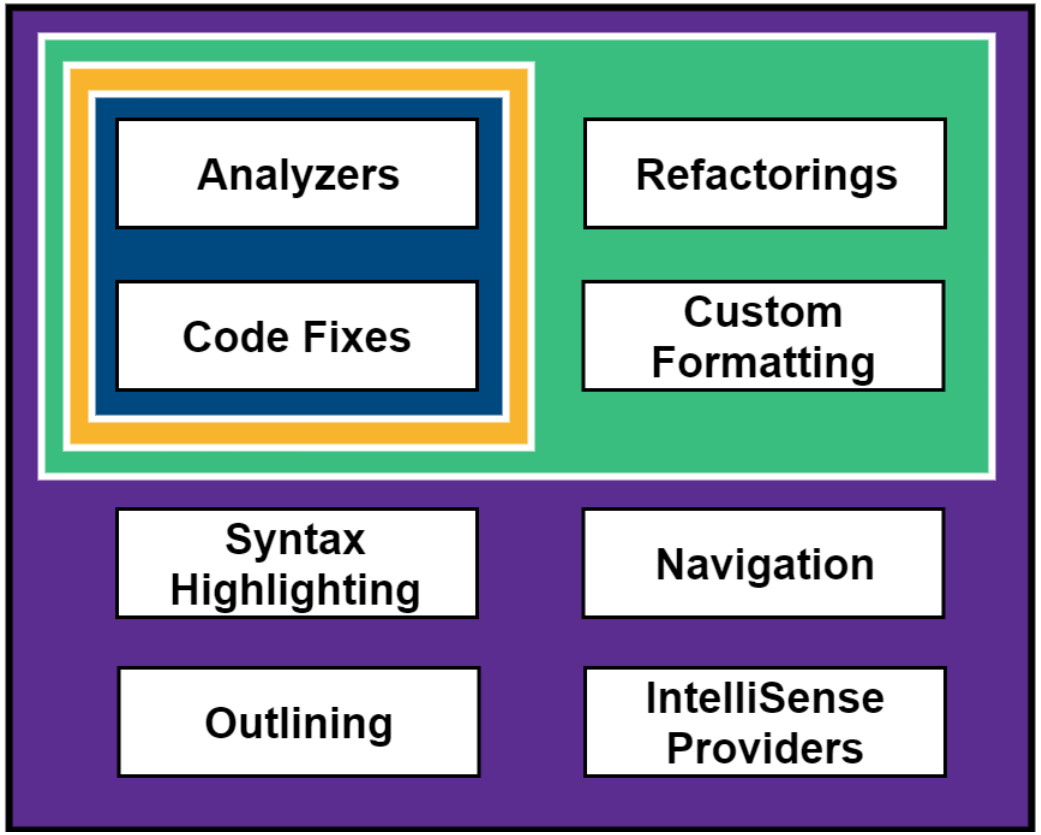
- SQL IntelliSense for Dapper queries

Навигация по коду

- Refactoring + custom CodeAction

nuget

Visual Studio



Rider

OmniSharp

Bottom Line

Свой статический анализ кода нужен, если:

- **У вас есть свой фреймворк, который используют другие люди**

Свой статический анализ кода нужен, если:

- У вас есть свой фреймворк, который используют другие люди
- **Вы используете сторонний фреймворк с большим количеством «подводных камней», и устали наступать на грабли**

Свой статический анализ кода нужен, если:

- У вас есть свой фреймворк, который используют другие люди
- Вы используете сторонний фреймворк с большим количеством «подводных камней», и устали наступать на грабли
- **Есть специфические требования к code style**

Свой статический анализ кода нужен, если:

- У вас есть свой фреймворк, который используют другие люди
- Вы используете сторонний фреймворк с большим количеством «подводных камней», и устали наступать на грабли
- Есть специфические требования к code style
- **Just for fun!**

Roslyn

- **Взрослая технология с отличными API**

Roslyn

- Взрослая технология с отличными API
- **Много интересных возможностей за пределами статического анализа**

Roslyn

- Взрослая технология с отличными API
- Много интересных возможностей за пределами статического анализа
- **Недостаток «фич» для production-сценариев:**

Roslyn

- Взрослая технология с отличными API
- Много интересных возможностей за пределами статического анализа
- Недостаток «фич» для production-сценариев:
  - **Нет рекурсивного анализа кода**

Roslyn

- Взрослая технология с отличными API
- Много интересных возможностей за пределами статического анализа
- Недостаток «фич» для production-сценариев:
  - Нет рекурсивного анализа кода
  - **Нет хороших встроенных suppression-механизмов**

Roslyn

- Взрослая технология с отличными API
- Много интересных возможностей за пределами статического анализа
- Недостаточно «фич» для сложных production-сценариев:
  - Нет рекурсивного анализа кода
  - Нет хороших встроенных suppression-механизмов
  - **Нет сложного data flow analysis**

Бенефиты

- **Ошибки находятся раньше**

Бенефиты

- Ошибки находятся раньше
- **Уменьшилось время сидения в дебаггере**

Бенефиты

- Ошибки находятся раньше
- Уменьшилось время сидения в дебаггере
- **Разработчики обучаются с помощью подсказок в IDE**

Бенефиты

- Ошибки находятся раньше
- Уменьшилось время сидения в дебаггере
- Разработчики обучаются с помощью подсказок в IDE
- **Сертификация сторонних решений стала автоматизированной**

Где найти информацию?

- Roslyn Cookbook by Manish Vasani
- Open-source проекты
  - Roslyn Analyzers: https://github.com/dotnet/roslyn-analyzers
  - Roslynator: https://github.com/JosefPihrt/Roslynator
  - xUnit Analyzers: https://github.com/xunit/xunit.analyzers
  - Acuminator: https://github.com/Acumatica/Acuminator
- Исходники Roslyn: https://github.com/dotnet/roslyn-sdk
- Доклады dotNext (Filip W)

Примеры

https://github.com/dropsonic/dotNext2019

Спасибо за внимание!

Владимир Панченко

dropsonic.pan@gmail.com

Acumatica

The Cloud ERP