**Why?**

Begginers and junior developers need mentoring. Senior developers are busy, they have limited bandwidth. Can tools help to find common mistakes, suggest fixes as the mistakes are made?

Help to find common mistakes and anti-patterns in code and suggest fixes for them

**Write robust code**

ExpressionStatement !== ReturnStatement, e.g. when trying to fix if-else without blocks do not look for ExpressionStatements explicitely, but look at whether what follows an if (or else) is NOT a block – then you can surface an error.

OOOR look for the base class of both – StatementSyntax, OOPS no, coz BloxcSyntax is a descendant fo StatementSyntax

**Analyzing non-valid C# code**

Watch out for invalid syntax that should be analyzed, e.g. var name.Concat(“next”);

This causes not robust analyzers to crash, as some arguments might be null. We should always write analyzers in a way so they can tell that and do not continue analyzing such nodes. In case they throw an exception, VisualStudio catches it, so that it does not cause VS to crash. Therefore, we cannot tell really something happened, but this has a significant impact on performance.

To test such a behavior, we would need to run the Experimental Hive of VS in debug and catch all the CLR exceptions (set this in Exception Settings).

We can also unit test this behavior by intentionally writing uncompilable code and tyring to run the analyzer. BUT! Again, VS catches that exception and tests are green. Therefore, test has to be run in debug mode, with CLR exceptions set to be all popped up.

* Debug tests for invalid syntax trees (mimics user typing)

**How fast should the analyzer be?**

It depends. There are two types of things to consider:

* Command line performance – how much does the Roslyn extension impacts the build time – should be less than 20%, ideally around 10%
* IDE responsiveness – Roslyn analysis does run on the separate thread, as it is asynchronous, so it does not affect the UI directly. Analyzer gets called upon every keystroke, so when the semantical model changes. That results in many objects being created and many thus it may cause memory management problems. Once the VS process consumes too much memory, garbage collector gets on stage and the IDE is suspended – well, not exactly user friendly. So it really depends, what you call an effective analyzer.

Always make sure to write analyzers that do not harm IDE performance.

**Per user vs. per project installations**

Per user = VISIX installation -> every project that user loads

Per project = NuGet installation -> evry developer that works on a project

Portable class libraries – can run enywhere where .net runs

Roslyn is a perfect way for Nuget authors to attach a simple analyzers for they code to show how to use their code.

Existing analyzers

Code-cracker

.NET Analyzers