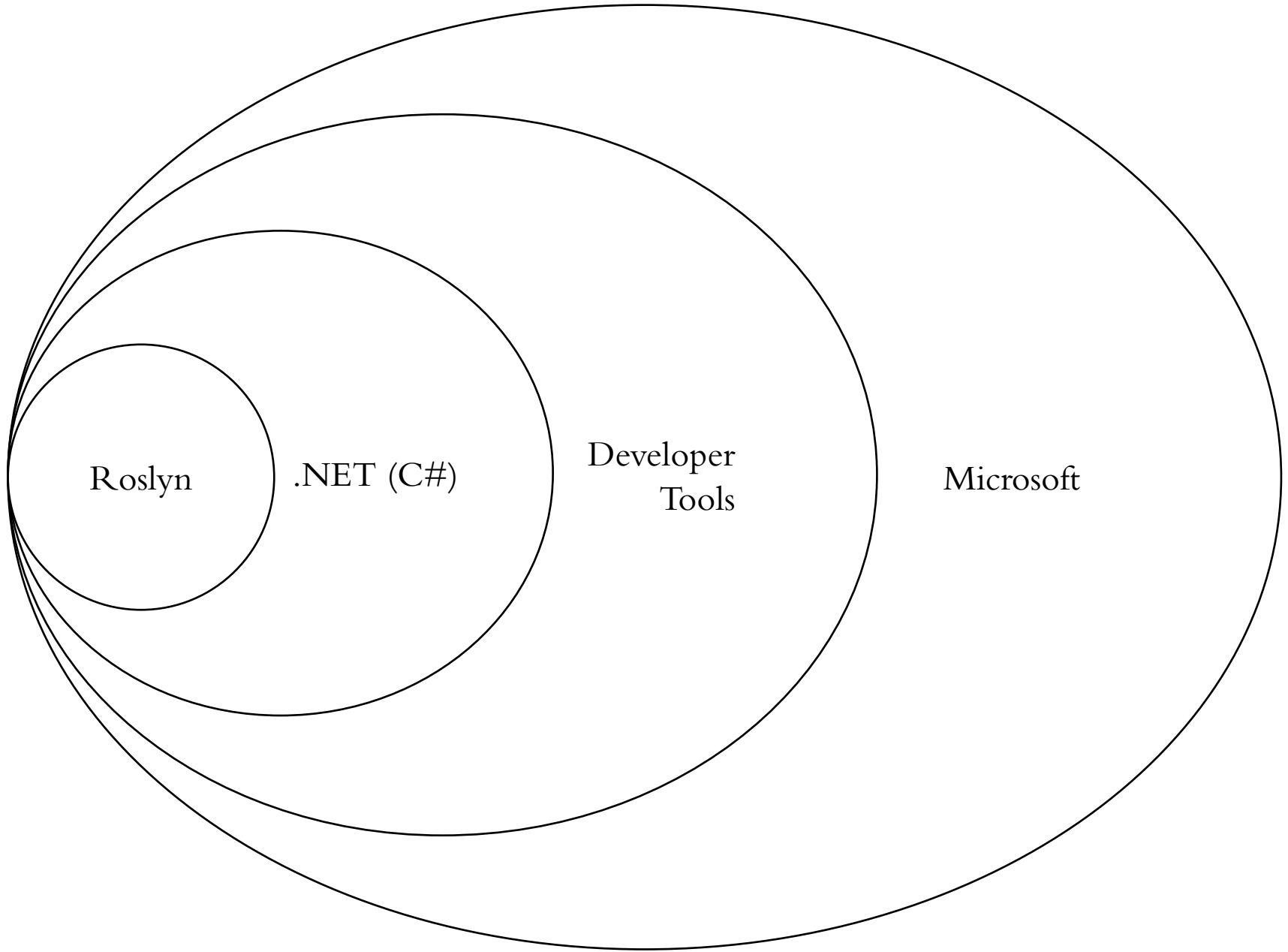


# Tools for C# developers

Introductions  
&  
Acknowledgements



Roslyn

.NET (C#)

Developer  
Tools

Microsoft

# Roslyn Team

- Speakers (Software Engineers):
  - Gen Lu
  - Manish Vasani
- Orchestrator (Engineering Manager)
  - Vatsalya Agrawal
- Designer (Program Manager)
  - Mika Dumont

# Our hearty thanks to...

- Professors
  - Roberta Gori
  - Laura Semini
  - Letterio Galletta
- Facilitators
  - Diego Colombo
  - Jon Sequeira

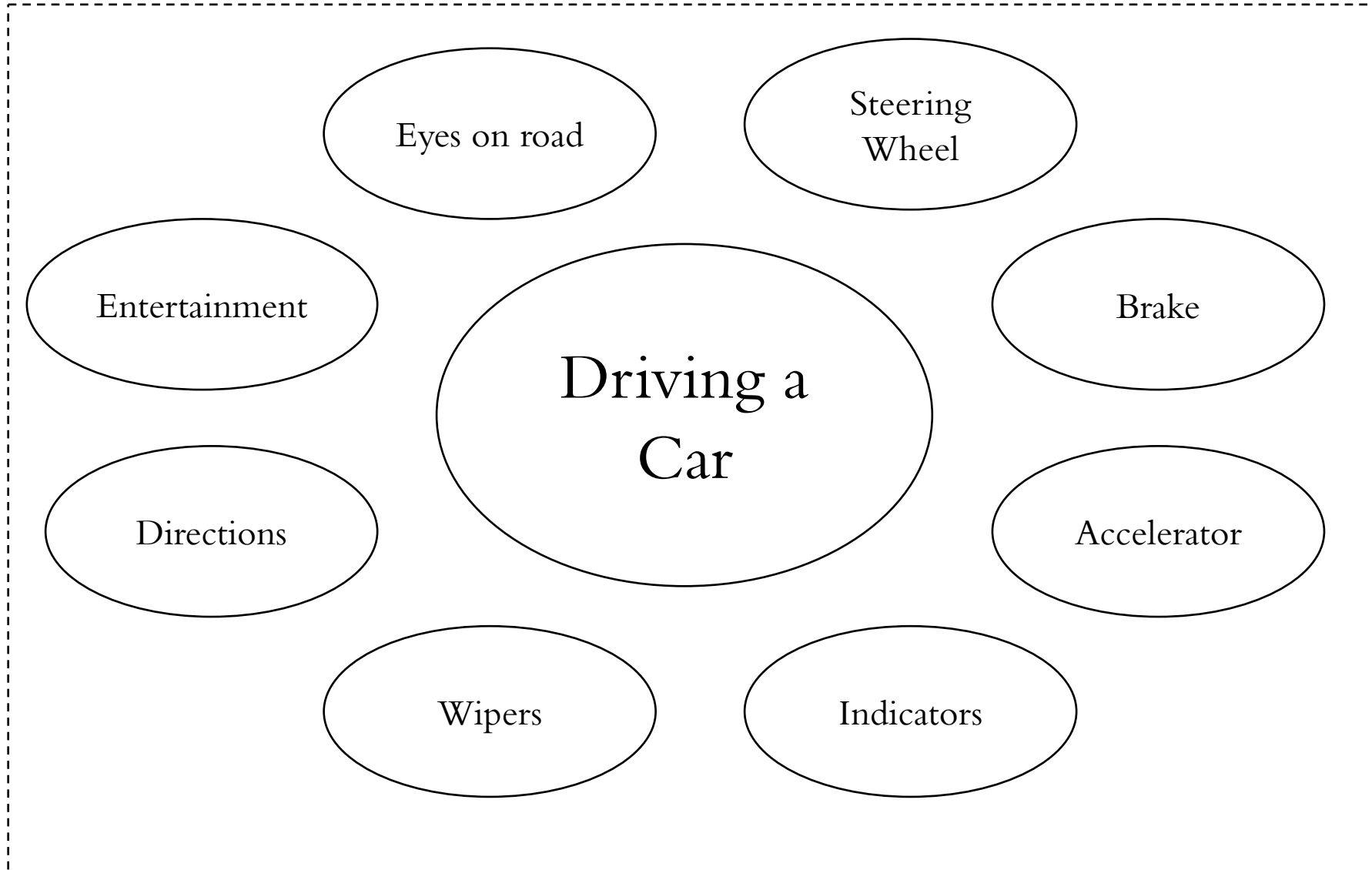
# Overview

- Introduction to the problem space
- How Roslyn helps C# developers
  - Tooling help for writing code
  - Easily search and navigate code
  - Identify code quality and style issues in code
  - Tooling help for testing code
- Demos
- Getting involved
- References and follow-up channels
- Q&A

Cars!!!

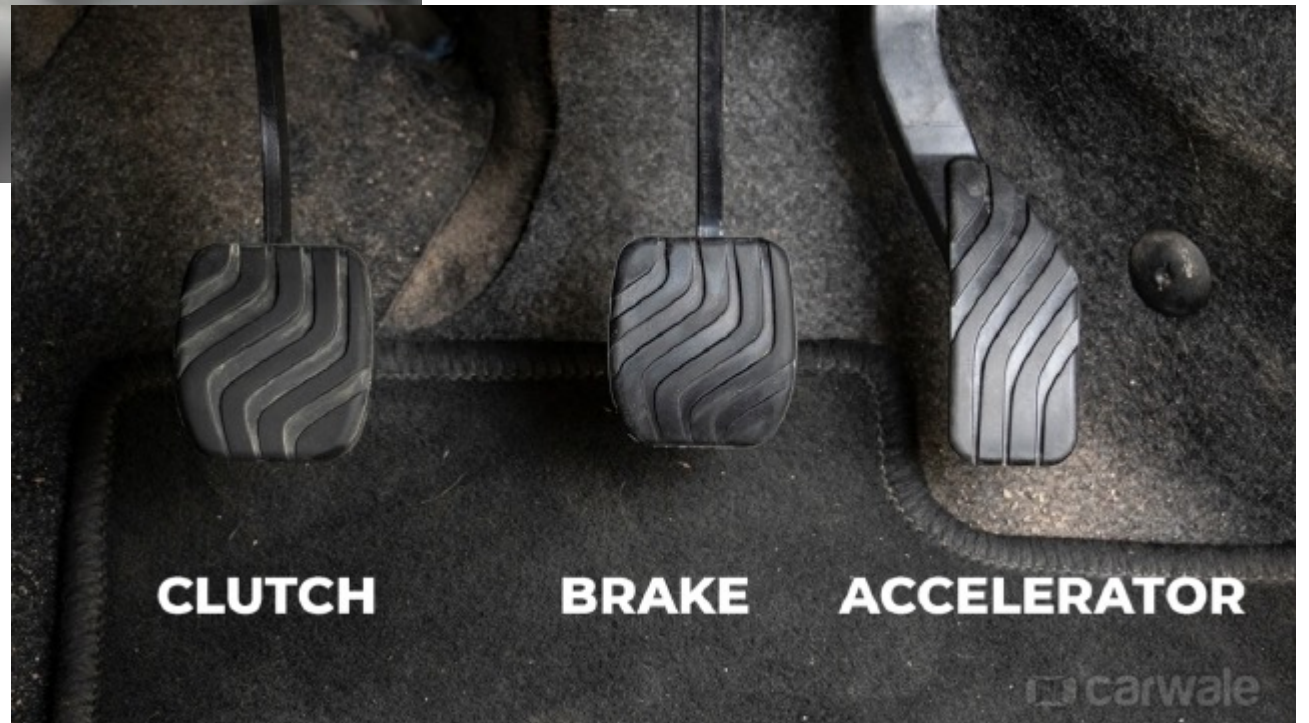


# Tasks to perform while...





# Manual transmission



# Automatic transmission



# Manual vs Automatic

- MANUAL

- Less Expensive
- Better Fuel economy

- AUTOMATIC

- Easier to drive
- Can focus on other tasks
- Better driving pleasure

# What if...??

- MANUAL

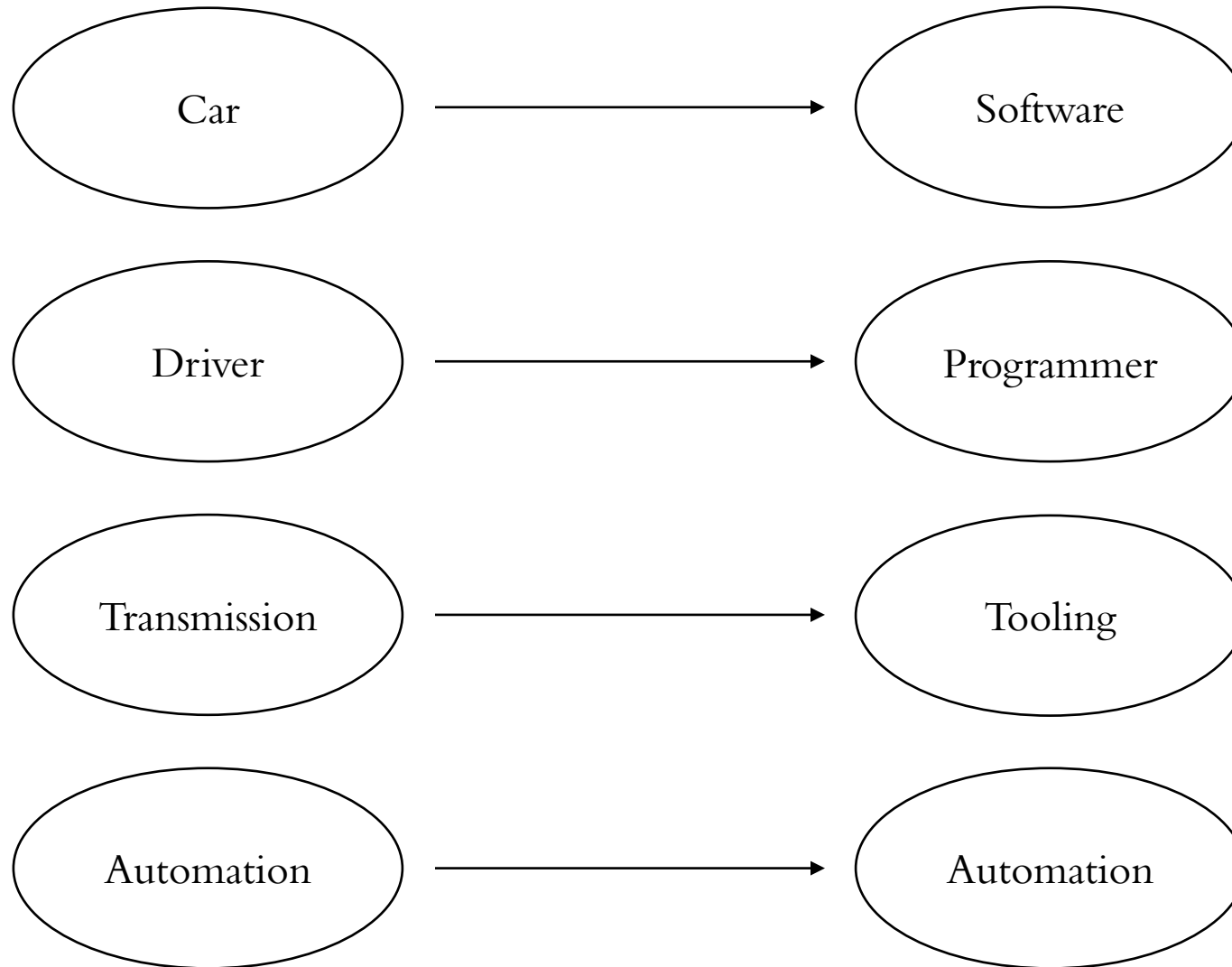
- Less Expensive
- Better Fuel economy

- AUTOMAGIC

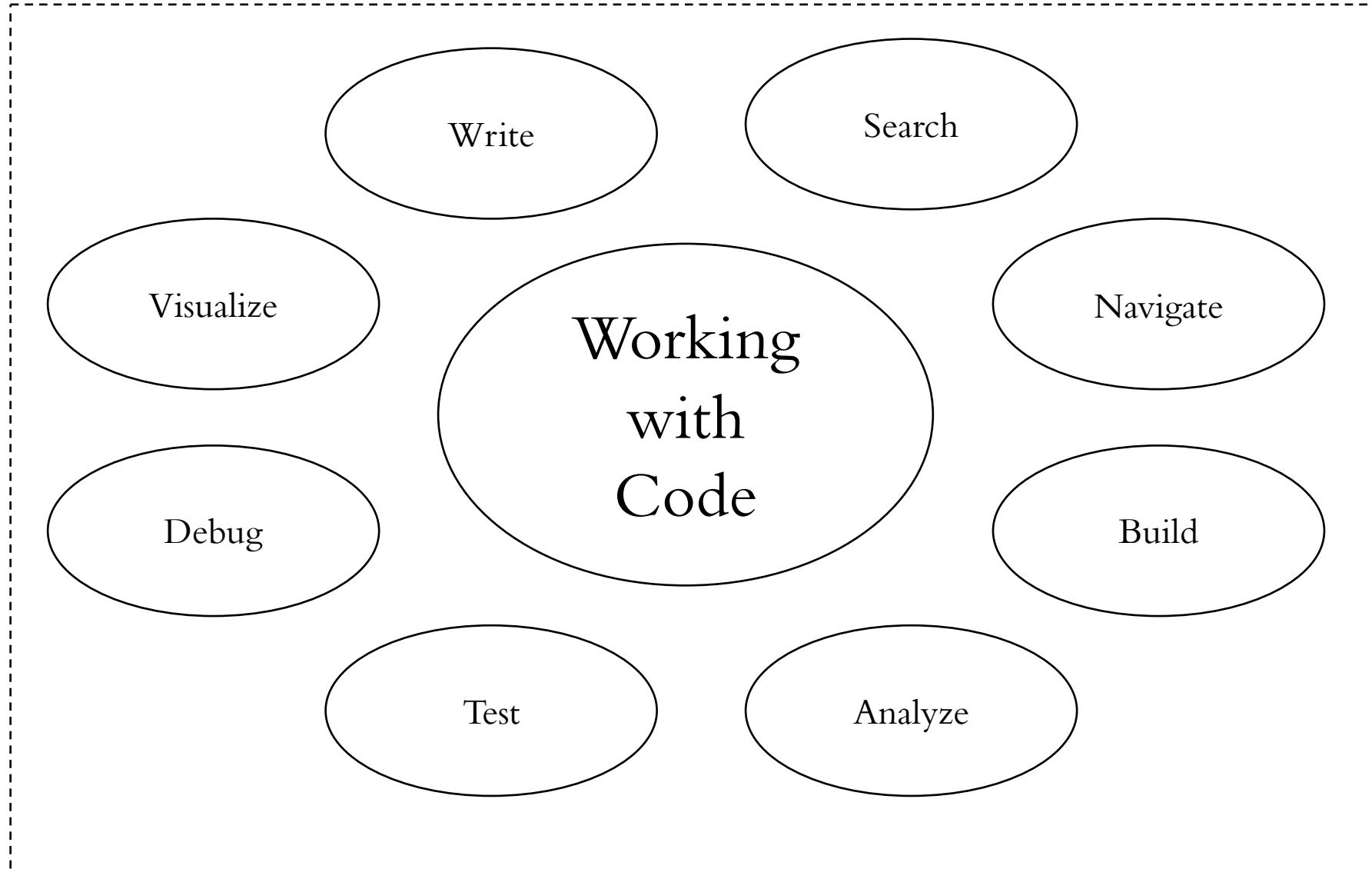
- Easier to drive
- Can focus on other tasks
- Better driving pleasure
  
- *Less Expensive*
- *Better Fuel economy*

We have a clear WINNER!!!

# Analogy



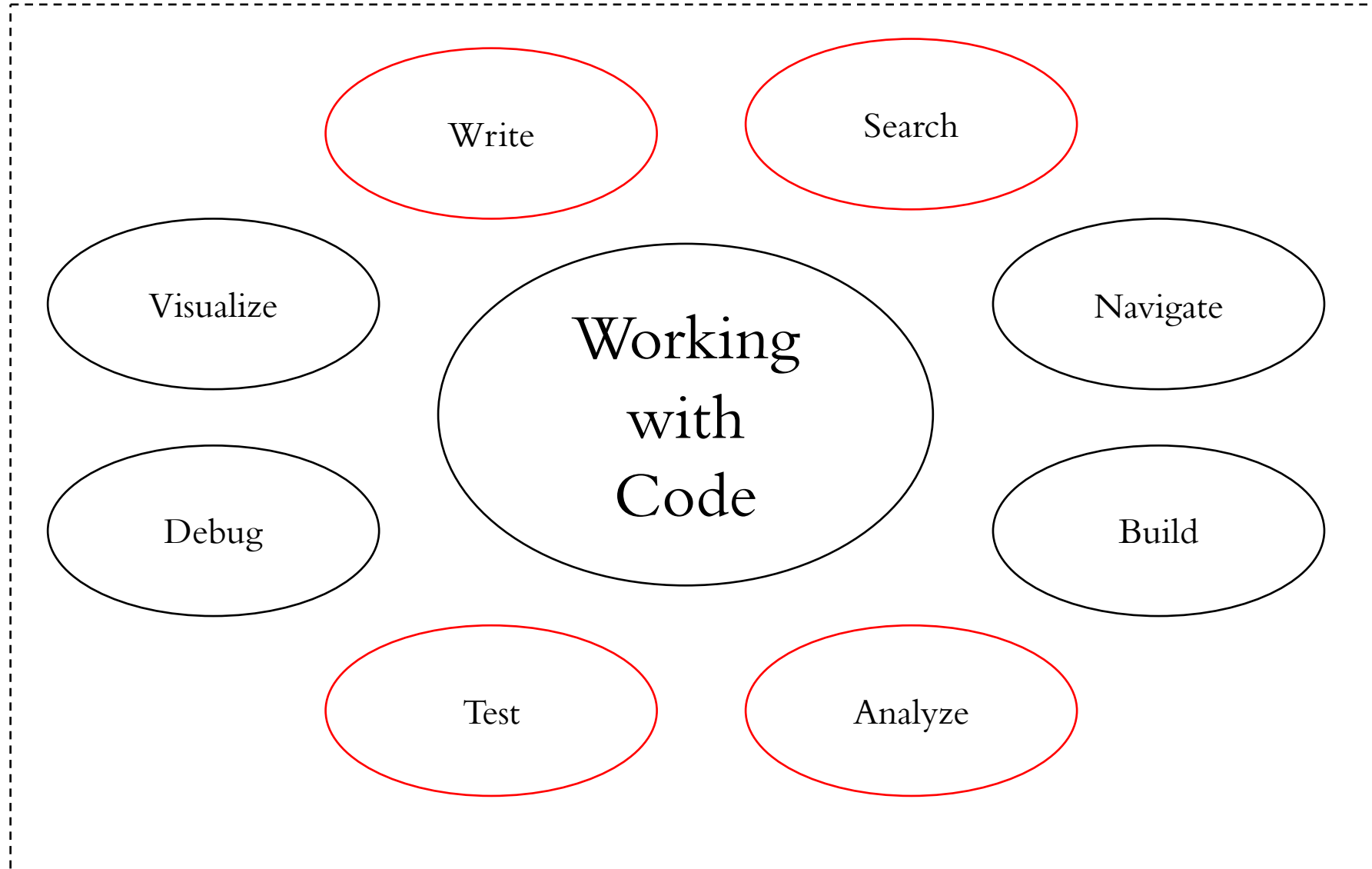
# Tasks to perform while...



# Roslyn == Automa**G**ic

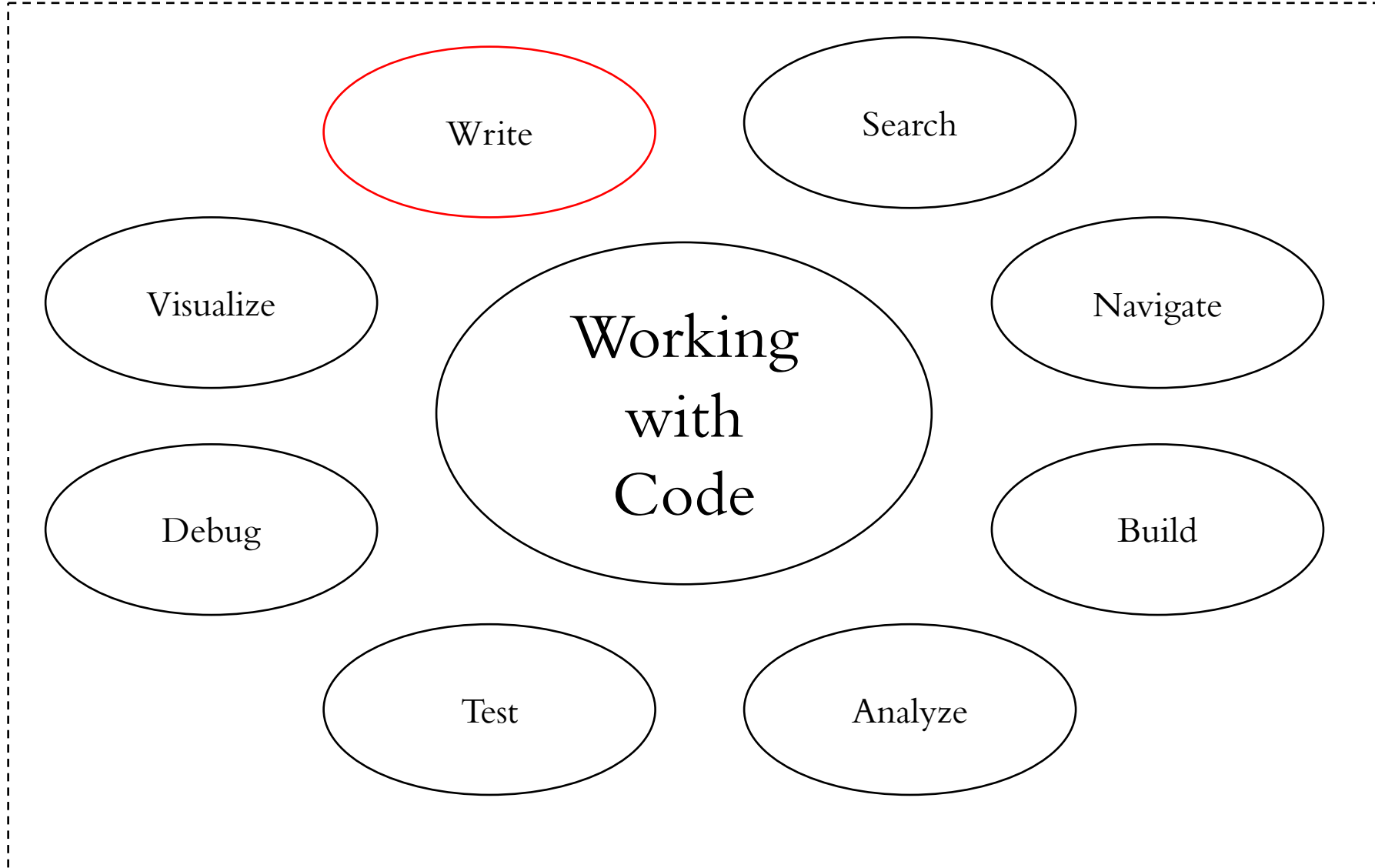
- Analyzes code in the background while you are typing in the editor
- Provides tooling help to automate and simplify these tasks for the programmers
- Programmers can focus on coding and enjoy coding

# Tasks to perform while...



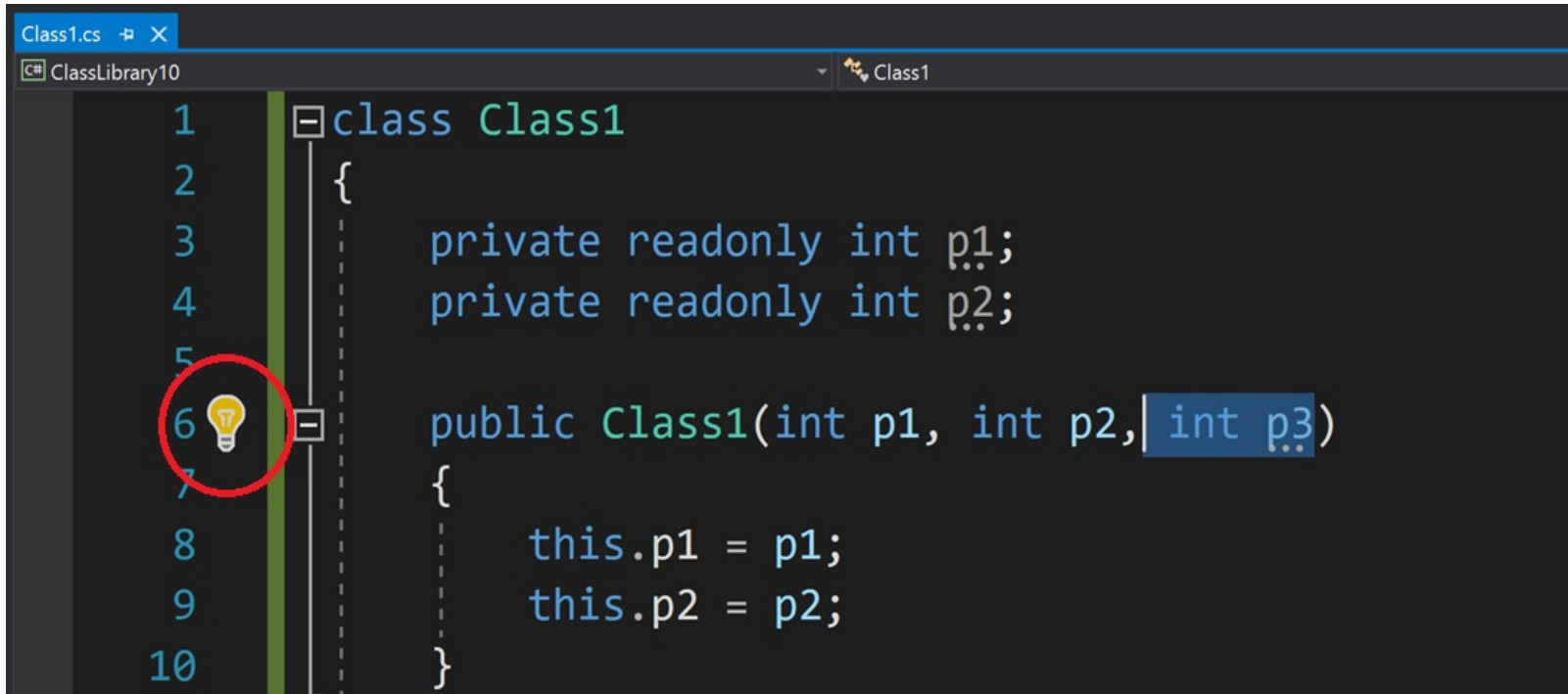


# Write new code



# Write new code

- #1 – Provides *code refactoring* suggestions to automatically edit code to perform most common code edits



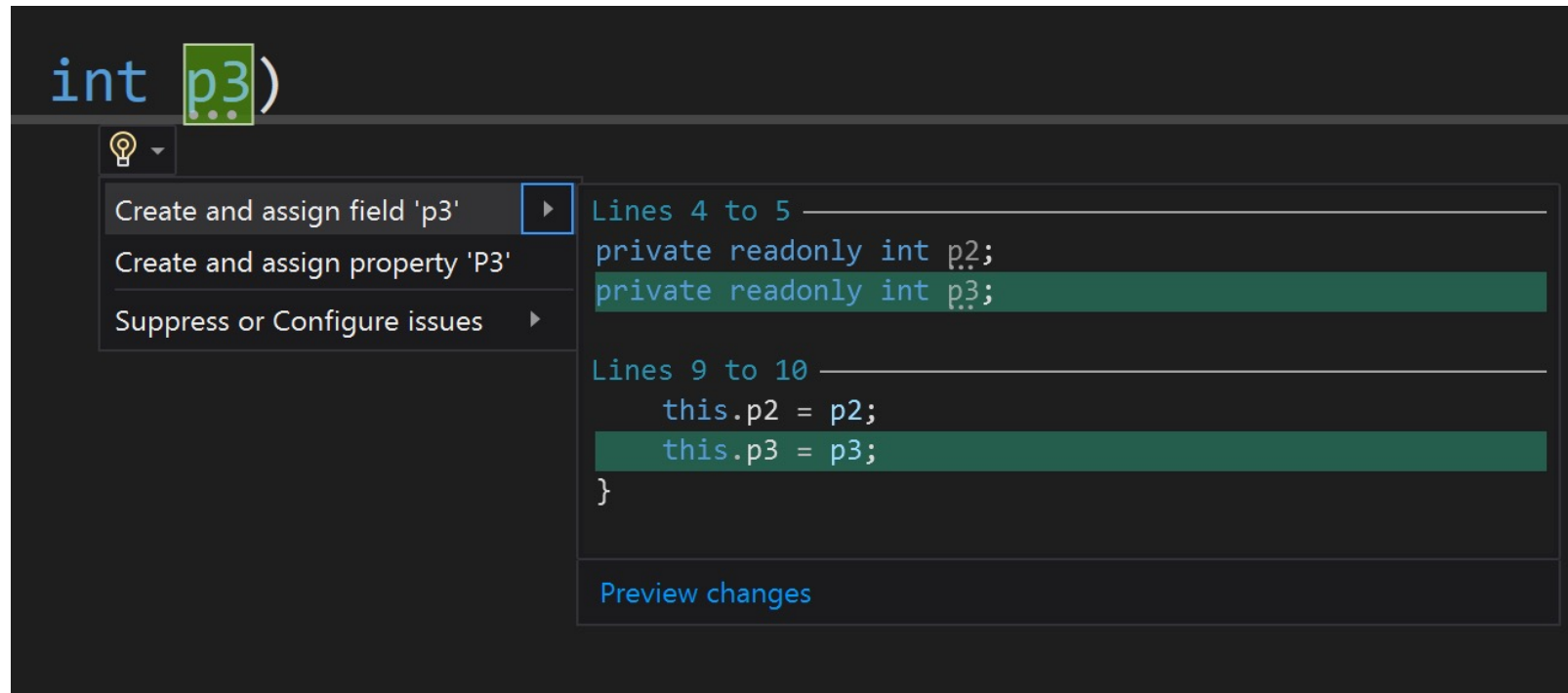
The screenshot shows a Visual Studio code editor window for a file named 'Class1.cs'. The editor displays the following C# code:

```
1 class Class1
2 {
3     private readonly int p1;
4     private readonly int p2;
5
6     public Class1(int p1, int p2, int p3)
7     {
8         this.p1 = p1;
9         this.p2 = p2;
10    }
```

A lightbulb icon is circled in red on line 6, indicating a refactoring suggestion. The parameter 'int p3' in the constructor signature is highlighted with a blue selection box.

# Write new code

- #1 – Provides *code refactoring* suggestions to automatically edit code to perform most common code edits



The screenshot shows a code editor with a dark theme. The code being edited is:

```
int p3)
```

A lightbulb icon is visible, indicating a suggestion. A dropdown menu is open, showing three options:

- Create and assign field 'p3'
- Create and assign property 'P3'
- Suppress or Configure issues

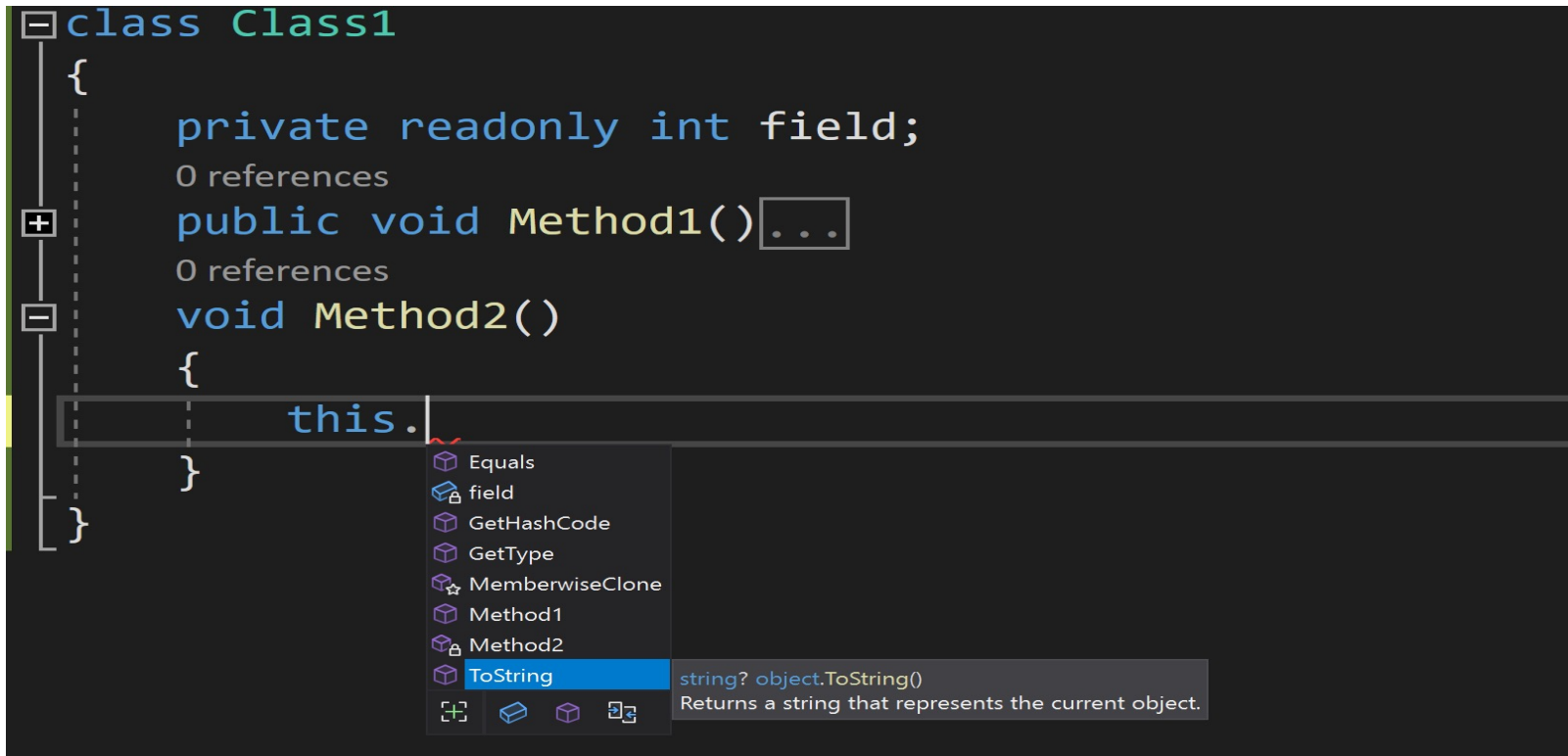
The first option, "Create and assign field 'p3'", is selected. To the right of the menu, a preview of the code changes is shown:

```
Lines 4 to 5  
private readonly int p2;  
private readonly int p3;  
  
Lines 9 to 10  
    this.p2 = p2;  
    this.p3 = p3;  
}
```

At the bottom of the preview, there is a link labeled "Preview changes".

# Write new code

- #2 - Provides *intellisense* suggestions to see available symbols in context of the code

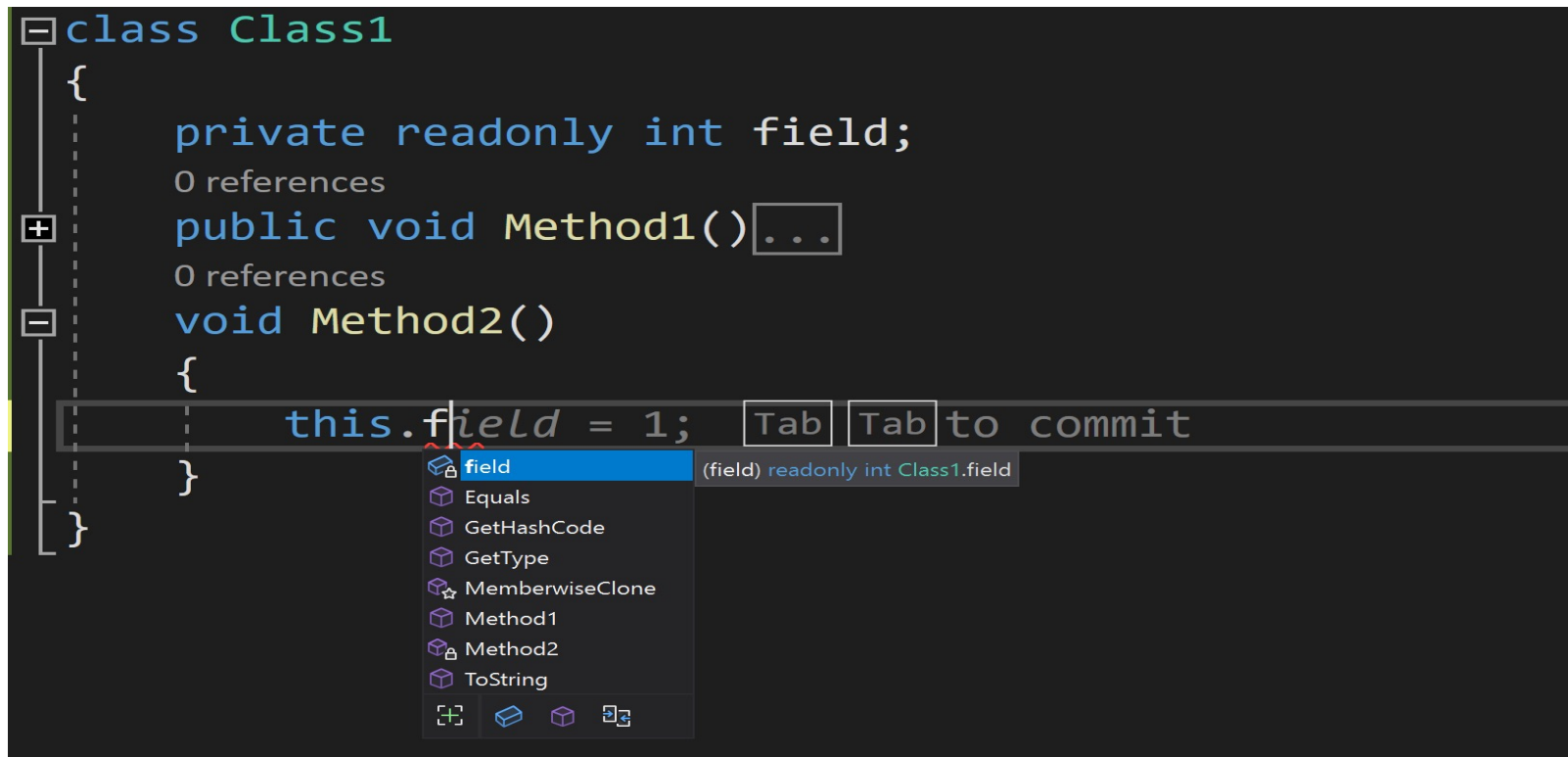


```
class Class1
{
    private readonly int field;
    0 references
    public void Method1() ...
    0 references
    void Method2()
    {
        this.
    }
}
```

The screenshot shows an IDE with a dark theme. A class named `Class1` is open, showing its members. The cursor is positioned at the end of the line `this.` inside the `Method2` method. A dropdown menu of intellisense suggestions is visible, listing members of the class: `Equals`, `field`, `GetHashCode`, `GetType`, `MemberwiseClone`, `Method1`, `Method2`, and `ToString`. The `ToString` option is highlighted in blue. A tooltip for `ToString` is also visible, showing the signature `string? object.ToString()` and the description "Returns a string that represents the current object."

# Write new code

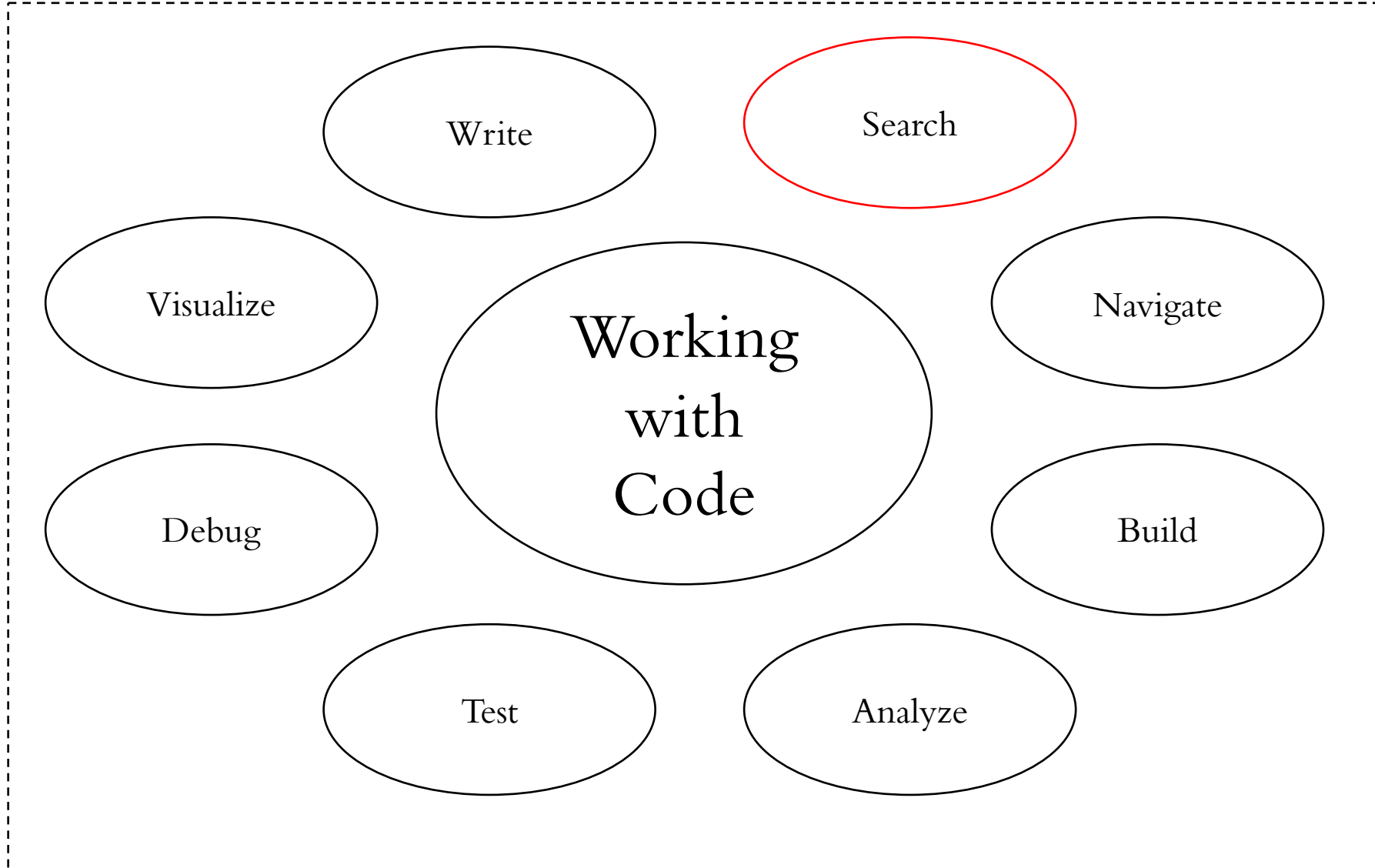
- #3 - Provides *automatic completions* to reduce manual typing



```
class Class1
{
    private readonly int field;
    0 references
    public void Method1() ...
    0 references
    void Method2()
    {
        this.field = 1;
    }
}
```

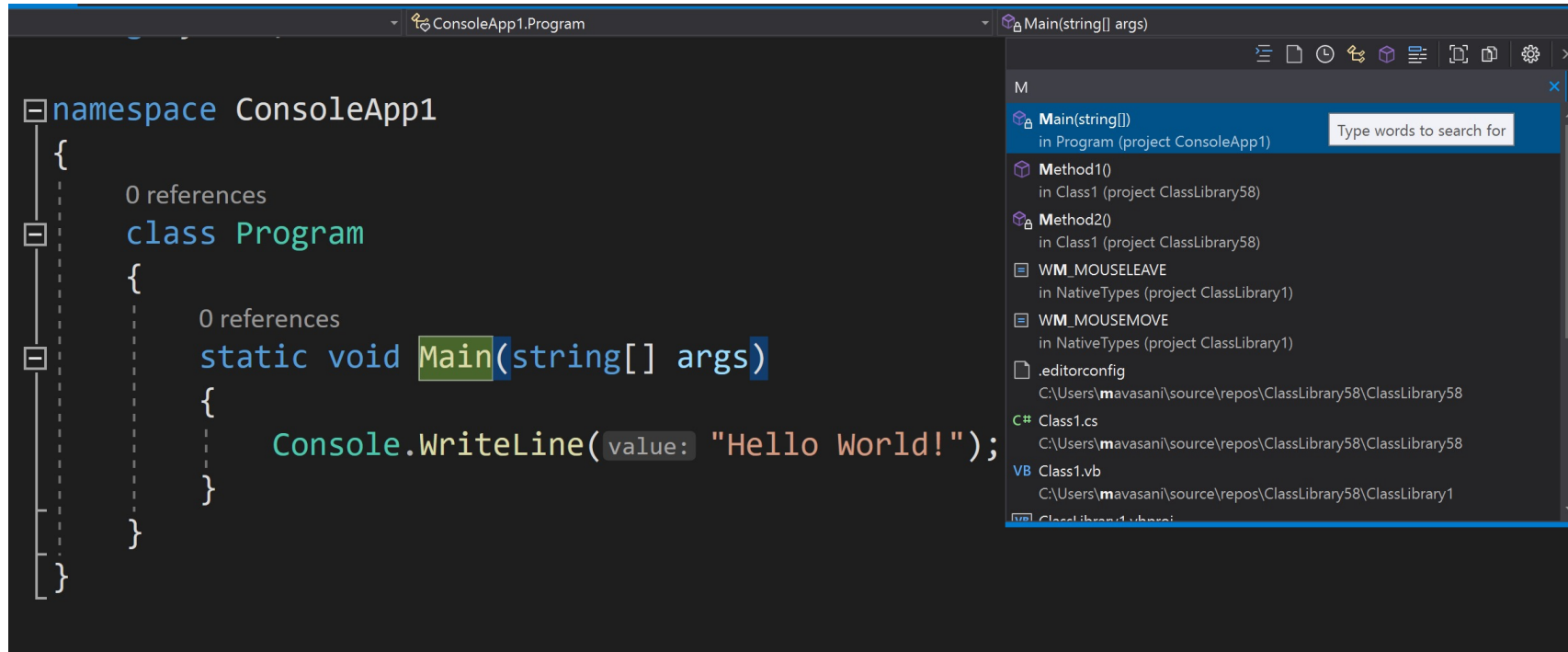
The screenshot shows an IDE with a dark theme. The code editor displays a C# class named `Class1`. Inside the class, there is a private readonly integer field named `field`. Below the field declaration, there are two methods: `Method1()` and `Method2()`. The `Method2()` method is currently being edited, and the cursor is positioned at the end of the line `this.field = 1;`. A dropdown menu is visible, showing a list of members for `Class1`, including `field`, `Equals`, `GetHashCode`, `GetType`, `MemberwiseClone`, `Method1`, `Method2`, and `ToString`. The `field` member is highlighted, and a tooltip shows its type: `(field) readonly int Class1.field`. The text `Tab Tab to commit` is visible in the background, indicating the next steps to complete the code.

# Search code

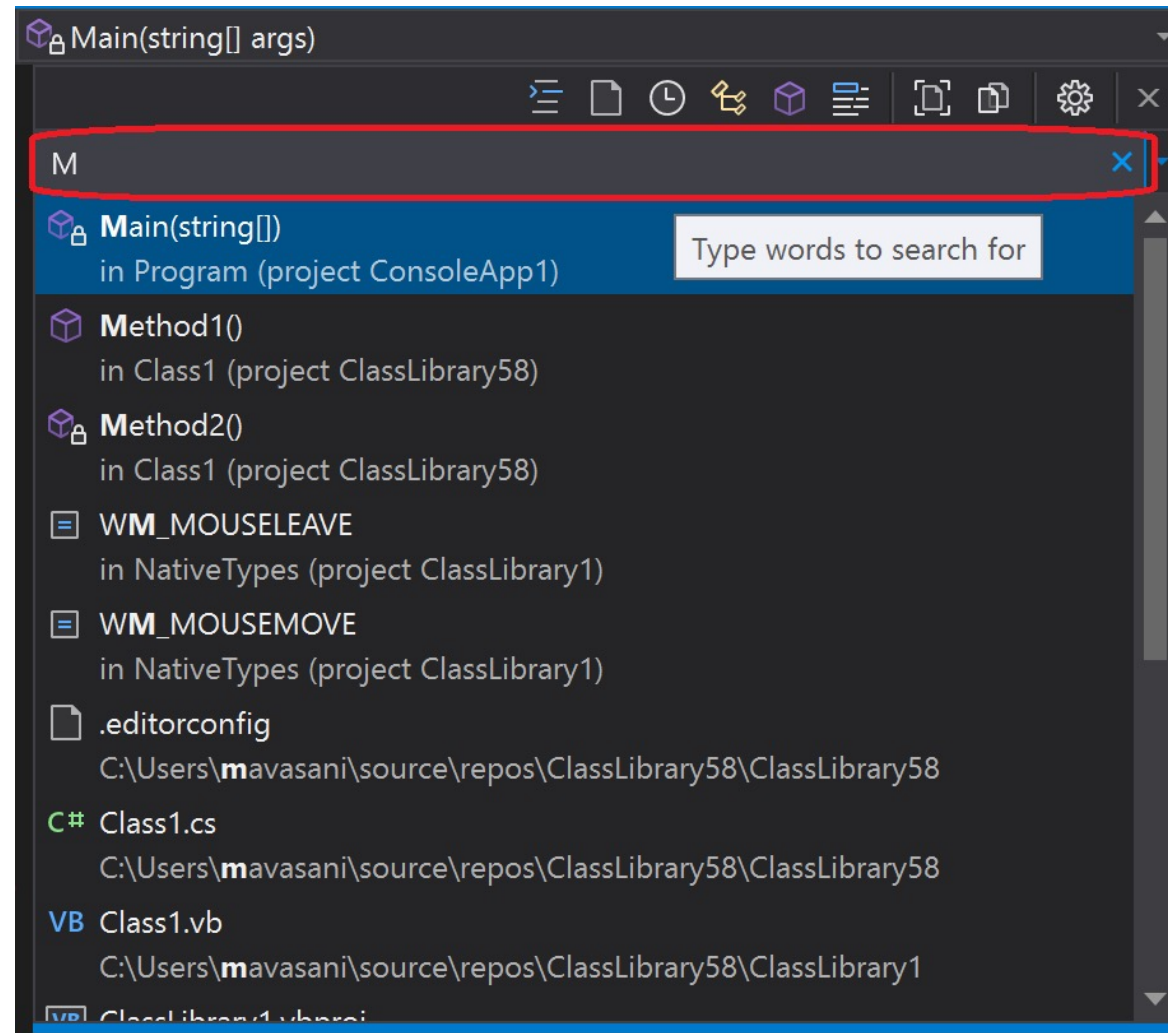


# Search code

- #1 – Search *symbols* in solution (locals, parameters, types, fields, methods)



# Search code





# Search code

- #2 – Find *references* to symbols

The screenshot displays the Visual Studio IDE with a code editor and a search results window. The code editor shows the following code for `Class1`:

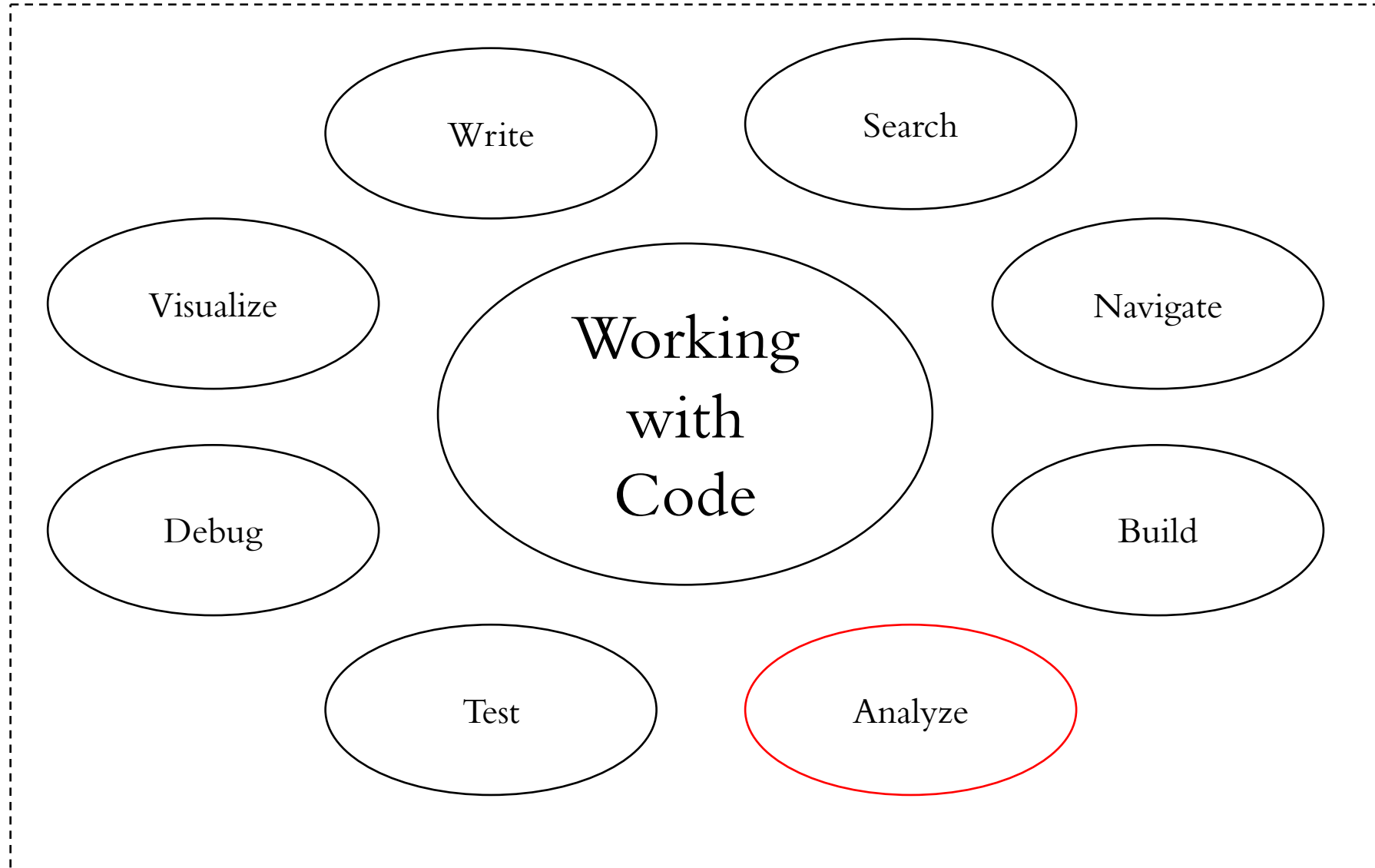
```
1 class Class1
2 {
3     private int field;
4     void Method()
5     {
6         this.field = 0;
7         int local = this.field;
8         this.field++;
9     }
10 }
```

The search results window, titled "'field' references", shows the following table:

Code	Kind	Line	Col	File
<code>int Class1.field (3)</code>				
<code>this.field = 0;</code>	Write	6	14	Class1.cs
<code>int local = this.field;</code>	Read	7	26	Class1.cs
<code>this.field++;</code>	Read, Write	8	14	Class1.cs

Red circles highlight the `int Class1.field (3)` entry in the search results and the corresponding code lines in the editor.

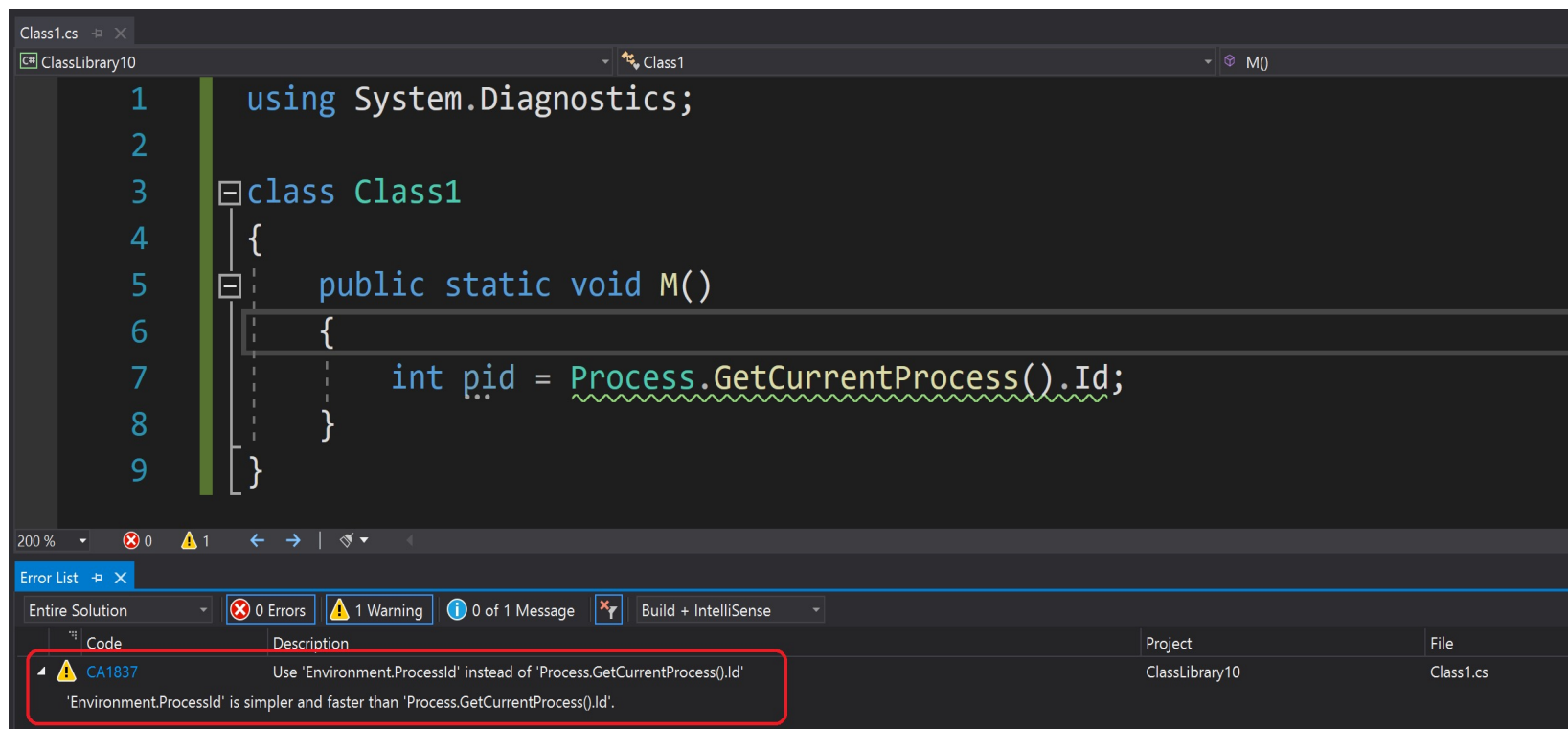
# Analyze code



# Analyze code

- #1 Find issues in code

While typing, you see *squiggles* in editor + entries in *error list*



```
1 using System.Diagnostics;
2
3 class Class1
4 {
5     public static void M()
6     {
7         int pid = Process.GetCurrentProcess().Id;
8     }
9 }
```

200 % 0 Errors 1 Warning 0 of 1 Message Build + IntelliSense

Code	Description	Project	File
CA1837	Use 'Environment.ProcessId' instead of 'Process.GetCurrentProcess().Id'. 'Environment.ProcessId' is simpler and faster than 'Process.GetCurrentProcess().Id'.	ClassLibrary10	Class1.cs

# Buckets of Code analysis issues

- #1 “Code quality” issues
- #2 “Code style” issues

# “Code quality” issues

- Performance
- Security
- Reliability
- Portability
- Maintenance
- Design
- And many more...

# “Code style” issues

- Whitespace formatting
- Newline formatting
- Naming styles

# Analyze code

- #2 Fix issues in code

```
class Class1
{
    public static void M()
    {
        int pid = Process.GetCurrentProcess().Id;
    }
}
```

Use 'Environment.ProcessId' | Suppress or Configure issues

CA1837 Use 'Environment.ProcessId' instead of 'Process.GetCurrentProcess().Id'

Lines 6 to 8

```
{
    int pid = Process.GetCurrentProcess().Id;
    int pid = System.Environment.ProcessId;
}
```

0 Errors | 1 Warning | 0 of 1 Message | Build + IntelliSense

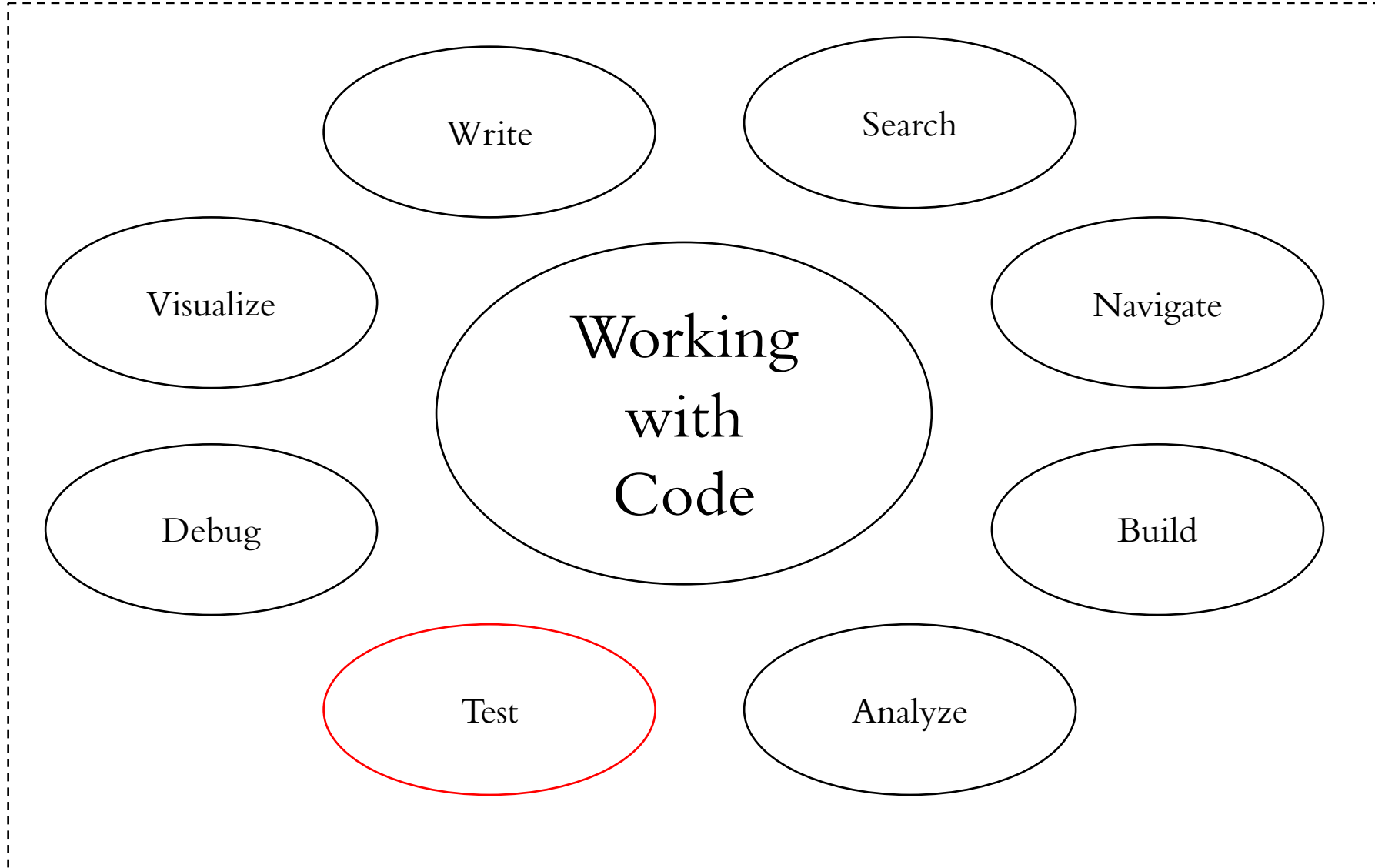
Description

Use 'Environment.ProcessId' instead of 'Process.GetCurrentProcess().Id'

Preview changes

Fix all occurrences in: Document | Project | Solution

# Write and Execute tests





# Test Code

- *Unit tests*: Validates an API or a small component
- *Integration tests*: Validates end-to-end working of group of components or the entire product

# Write and Execute tests

- Normal software engineering cycle:
  - Write product code
  - Write tests
  - Execute tests
  - Change/Fix code
  - Re-run existing tests
    - All tests
    - Manually identify affected tests
  - Write more tests

# Re-run existing tests

- All tests
  - Test execution takes lot of time
- Manually identify affected unit tests
  - Manual effort is cumbersome and also unreliable
- *Live unit testing (LUT)* or Automatic test execution

# Live unit testing

- Continuous background code analysis identifies unit tests which can be affected by product code changes
- Schedules automatic background execution of affected tests
- On test completion, user is given a message about test results

# So, what is Roslyn?

- Roslyn is an *open-source* platform for *.NET languages (C#/VB)*
  - Enables writing *tools* for .NET:
    - Compiler
    - Debugger
    - Editor or IDE (Integrated Development Environment)
  - Enables writing *extensions* to these tools:
    - Code refactorings
    - Diagnostic analyzers for static analyses
    - Code fixes

Demos

Thank you,  
Gen!

# Recap..

- Code refactorings
  - Suggestions for automatic code changes
- Code analysis diagnostics (code quality + code style)
  - Build time enforcement
  - Suggestions for automatic code fixes
- Intellisense + code completion
  - Tooling help while editing code
- Testing



# Fully Extensible



# Write custom extensions

- Visual Studio ships with project templates to create custom analyzer, code fix or a code refactoring
- Use Roslyn APIs to analyze syntax trees, symbols and executable code
- You can write an extension from scratch in less than 50 lines of code!

# Publish custom extensions

- Project template contains functionality to create a package on build
- Package can be easily uploaded on the internet and will be available for download to public
- Your custom extension will execute on C# command line builds and inside Visual Studio

Demo

# Getting involved

- Automate an often-repeated manual action into a code refactoring
- Automate a common coding mistake or style pattern into a diagnostic and code fix
- Enable these for your own projects and publish it for others to use in their projects

# Getting involved

- We are *fully open source* and completely open to community contributions
- ***Repos:***
  - Roslyn: <https://github.com/dotnet/roslyn>
  - Analyzers: <https://github.com/dotnet/roslyn-analyzers>
- Good first *issues:*
  - Roslyn: [query](#)
  - Analyzers: [query](#)

# Getting involved

- Good first *features*:
  - Roslyn: [query](#)
  - Analyzers: [query](#)
- Create tutorials for others
- Contribute to documentation: <https://github.com/dotnet/docs>
- Live Chat/Discussions/Questions: <https://gitter.im/dotnet/roslyn>

# References

- Learn C#: <https://dotnet.microsoft.com/learn/csharp>
- Lightbulbs and Quick actions:  
<https://docs.microsoft.com/visualstudio/ide/quick-actions>
- Code analysis: <https://docs.microsoft.com/visualstudio/code-quality/code-analysis-for-managed-code-overview>



# References

- Learn about Roslyn SDK and APIs: <https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/>
- Roslyn documentation: <https://github.com/dotnet/roslyn/tree/main/docs>

# Books

- C# books:
  - **C# Programming Language:** <https://www.amazon.com/Programming-Language-Covering-Portable-Documents-ebook-dp-B004BSFKXY/dp/B004BSFKXY>
  - **C# 9 and .NET 5:** <https://www.amazon.com/NET-Cross-Platform-Development-intelligent-Framework-ebook/dp/B08KQK22LJ>
- Roslyn books:
  - **Roslyn Cookbook:** <https://www.amazon.com/Roslyn-Cookbook-Manish-Vasani/dp/1787286835>
  - **Code generation with Roslyn:** <https://www.oreilly.com/library/view/code-generation-with/9781484222119/>

Q&A

Thank you!