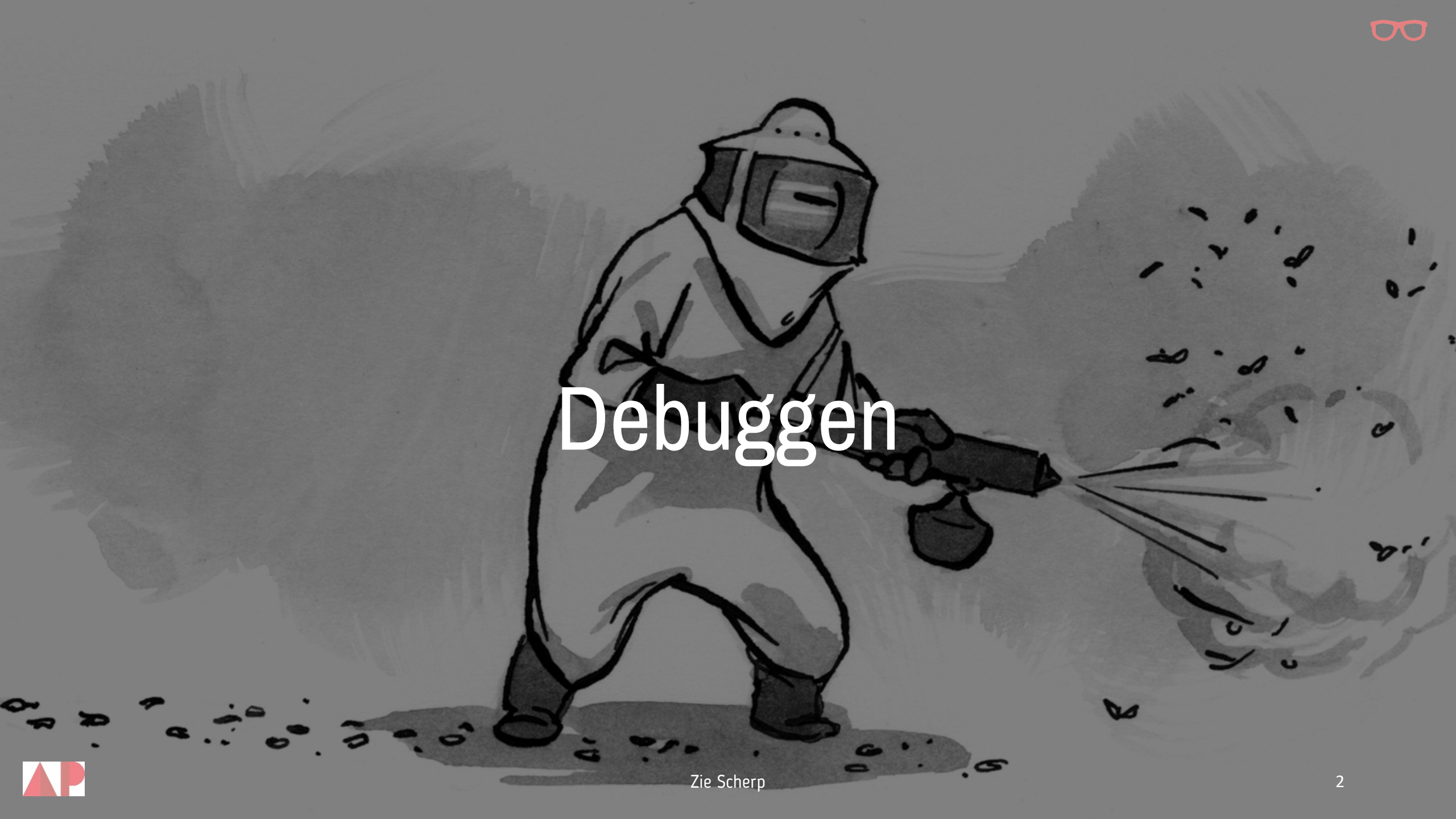
5. Debuggen

H4. Werken met data



Debuggen

verb.

Debugging

ˌdiɪˈbʌg-ing

1. Being the detective in a crime movie where you are also the murderer.

See also: Programmer and Genius

Zie Scherp

Inleiding

- Bug = fout in een programma
- Oplossen van bugs is voor omvangrijke programma's geen eenvoudige taak en is een discipline op zich

At compile-time (kunnen we reeds)

At link-time (volgend jaar)

At run-time (focus vandaag)

Breakpoint zetten

Breakpoint: uitvoer zal op dit punt, VOOR deze lijn, ‘pauzeren’ en naar Visual Code gaan
Meerdere breakpoints toegelaten

```
11            switch(oper)
12            {
13                case '+':
14                    return x + y;
15                case '-':
16                    return x - y;
17                case '*':
18                    return x * y;
19                case '/':
20                    return x / y;
21                default:
22                    return 0;
23            }
24        }
25    }
26 }
```

Bekijk je deze slides in pdf? De animated gifs in deze en volgende slides kan je hier in beweging zien
https://tutorials.visualstudio.com/vs-get-started/debugging

Debugger starten

DotNetCalculator - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Test Analyze Window Help

Debug | Any CPU | DotNetCalculator

Calculator.cs | Program.cs

DotNetCalculator | DotNetCalculator.Program | Main(string[] args)

```
7   static void Main(string[] args)
8   {
9
10      Console.WriteLine("Calculator Console Application");
11      while (true)
12      {
13          Console.WriteLine();
14          Console.WriteLine("Please enter the first number");
15          double x = Convert.ToDouble(Console.ReadLine());
16
17          Console.WriteLine("Please enter an operand (+ - / *)");
18          char oper = Convert.ToChar(Console.ReadLine());
19
20          Console.WriteLine("Please enter the second number");
21          double y = Convert.ToDouble(Console.ReadLine());
22
23          Calculator c = new Calculator();
24          double result = c.Calculate(x, oper, y);
25
```

100 %

Output

Show output from: Debug

```
'dotnet.exe' (CoreCLR: DefaultDomain): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\2.0.4\System.Private.CoreLib
'dotnet.exe' (CoreCLR: clrhost): Loaded 'C:\Users\jefgao\Source\Repos\DotNetCalculator\DotNetCalculator\bin\Debug\netcoreapp2.0\
'dotnet.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\2.0.4\System.Runtime.dll'. Skipped
'dotnet.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\2.0.4\System.Console.dll'. Skipped
'dotnet.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\2.0.4\System.Runtime.Extensions.dl
'dotnet.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\2.0.4\System.Threading.dll'. Skipp
The program '[184832] dotnet.exe' has exited with code -1073741510 (0xc000013a).
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'DotNetCalculator' (1 project)
  - DotNetCalculator
    - Dependencies
    - Calculator.cs
    - Program.cs

Solution Explorer | Team Explorer

Properties

Ready | Ln 19 | Col 17 | Ch 17 | INS | Add to Source Control

Zie Schema

Watch (autos) venster observeren

```
17          Console.WriteLine("Please enter an operand (+ - / *)");
18          char oper = Convert.ToChar(Console.ReadLine());
19                  oper 43 '+'
20          Console.WriteLine("Please enter the second number");
21          double y = Convert.ToDouble(Console.ReadLine());
22
23          Calculator c = new Calculator();
24          double result = c.Calculate(x, oper, y);   ≤ 5,324ms elapsed
25
26          Console.WriteLine("The result is {0}", result);
27      }
28  }
```

100 %

Autos

Name	Value	Type
c	{DotNetCalculator.Calculator}	DotNetCalculato
oper	43 '+'	char
result	0	double
x	5	double
y	10	double

Zie Scherp

Autos | Locals | Watch 1

Doorheen code ‘steppen’

```
17            Console.WriteLine("Please enter an operand (+ - / *)
18            char oper = Convert.ToChar(Console.ReadLine());
19
20            Console.WriteLine("Please enter the second number");
21            double y = Convert.ToDouble(Console.ReadLine());
22
23            Calculator c = new Calculator();
24            double result = c.Calculate(x, oper, y);   ≤ 1ms elapsed
25
26            Console.WriteLine("The result is {0}", result);
27        }
28    }
29 }
30 }
```

Command	Toolbar icon	Description
Continue		Resumes code execution until the next breakpoint is hit
Step Into		Runs the next statement. If the current line contains a function call, **Step Into** steps into the most deeply nested function. If you use **Step Into** on a line of code like `Func1(Func2())`, the debugger steps into the function `Func2`. If the current line doesn’t contain a function call, **Step Into** runs the code then suspends execution at the next line of code.
Step Over		Runs the next statement without stepping into functions or methods. If the current line contains a function call, **Step Over** runs the code then suspends execution at the first line of code after the called function returns.
Step Out		Advances the debugger all the way through the current fuction. **Step Out** continues running code and suspends execution when the current function returns.

*Hint: **Step Into** and **Step Over** are your primary tools for stepping through code line-by-line. Use **Step Into** when you want to debug a method called by the current line of code. Otherwise, use **Step Over**.*

6 Stages of Debugging

1. This can't happen.
2. That doesn't happen on my machine.
3. That shouldn't happen.
4. Why does that happen?
5. Oh, I see.
6. How did that ever work?

Demo: de debugger gebruiken

- Debugger starten en stoppen
- Breakpoints
- Single Stepping

- Watch window, Locals, Autos, Immediate, Call Stack