

C# 6.0 UND VORSCHAU AUF C# 7

HINTERGRÜNDE, FEATURES, BEISPIELE

Vortrag am 18.07.2016 von Robert Walter
bei [INdotNET – Ingolstädter .NET User Group](#)

INHALT

- C# 6.0 – Seit wann und wo?
- C# 6.0 – Was genau?
- C# 6.0 – Das Beste!
- Und was ist mit VB14?
- C# 7 – Ein Blick in die Zukunft!

The background is a dark blue gradient. In the four corners, there are white, stylized circuit board traces. These traces consist of straight lines that turn at right angles, ending in small circles that represent components or connection points. The traces are more dense in the bottom-left and top-left corners and more sparse in the top-right and bottom-right corners.

C# 6.0 – SEIT WANN UND WO?

ALLGEMEINE VERFÜGBARKEIT VON C# 6.0

C# 6.0 ist schon bereits seit dem 20.07.2015 final verfügbar.

(Bzw. seit 2014 als Preview verfügbar.)

Wenn da nicht der Tail Call Bug gewesen wäre...



TAIL CALL BUG IN RYUJIT (.NET 4.6) (PROBLEM)

[Blog-Post from Nick Craver: Why you should wait on upgrading to .Net 4.6](#)

[GitHub Issue on dotnet/coreclr: Tail Call bug in RyuJIT - Incorrect parameters passed](#)

"The methods you call can get different parameter values than you passed in"

"Recommendations: Do not install .Net 4.6 in production."

Nick Craver

software developer and system administrator

Stack Exchange (home of the popular programming support site Stack Overflow)

Aussage vom 27.07.2015

TAIL CALL BUG IN RYUJIT (.NET 4.6) (LÖSUNG)

LÖSUNG UNTER .NET 4.6

Update (August 11th): A patch for this bug has been released by Microsoft:

We released an updated version of RyuJIT today, which resolves this advisory. The update was released as [Microsoft Security Bulletin MS15-092](#) and is available on Windows Update or via direct download as [KB3086251](#). The update resolves: [CoreCLR #1296](#), [CoreCLR #1299](#), and [VisualFSharp #536](#). Major thanks to the developers who reported these issues. Thanks to everyone for their patience.

Quelle: <http://nickcraver.com/blog/2015/07/27/why-you-should-wait-on-dotnet-46/>

LÖSUNG UNTER .NET 4.6.1

Bug auf alle Fälle mit .NET 4.6.1 gefixt.

Quelle:

<https://github.com/Microsoft/dotnet/blob/master/releases/net461/dotnet461-changes.md>

.NET 4.6.1 ist ebenfalls in Visual Studio 2015 integriert.

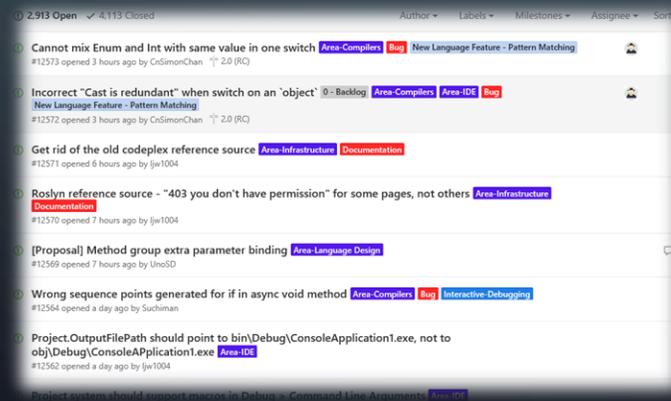
OFFENE DISKUSSION UM C# FEATURES

Beispiel: Öffentliche Diskussion um [Verzicht von null](#) ist 70 DIN A4 Seiten lang!

C# Language Design Discussions:

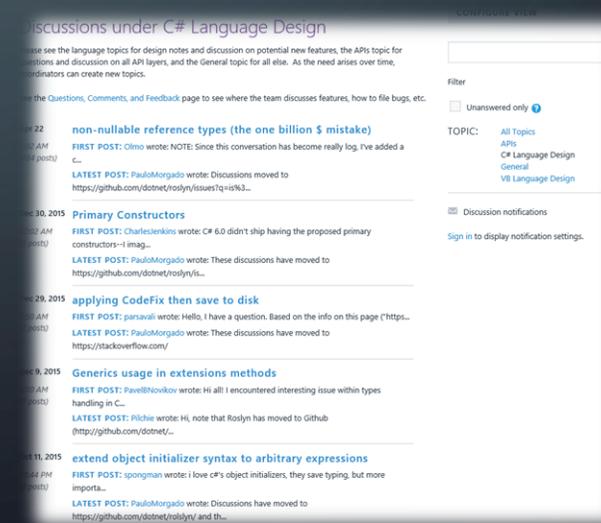
alt: <http://roslyn.codeplex.com/discussions>

neu: <https://github.com/dotnet/roslyn/issues>



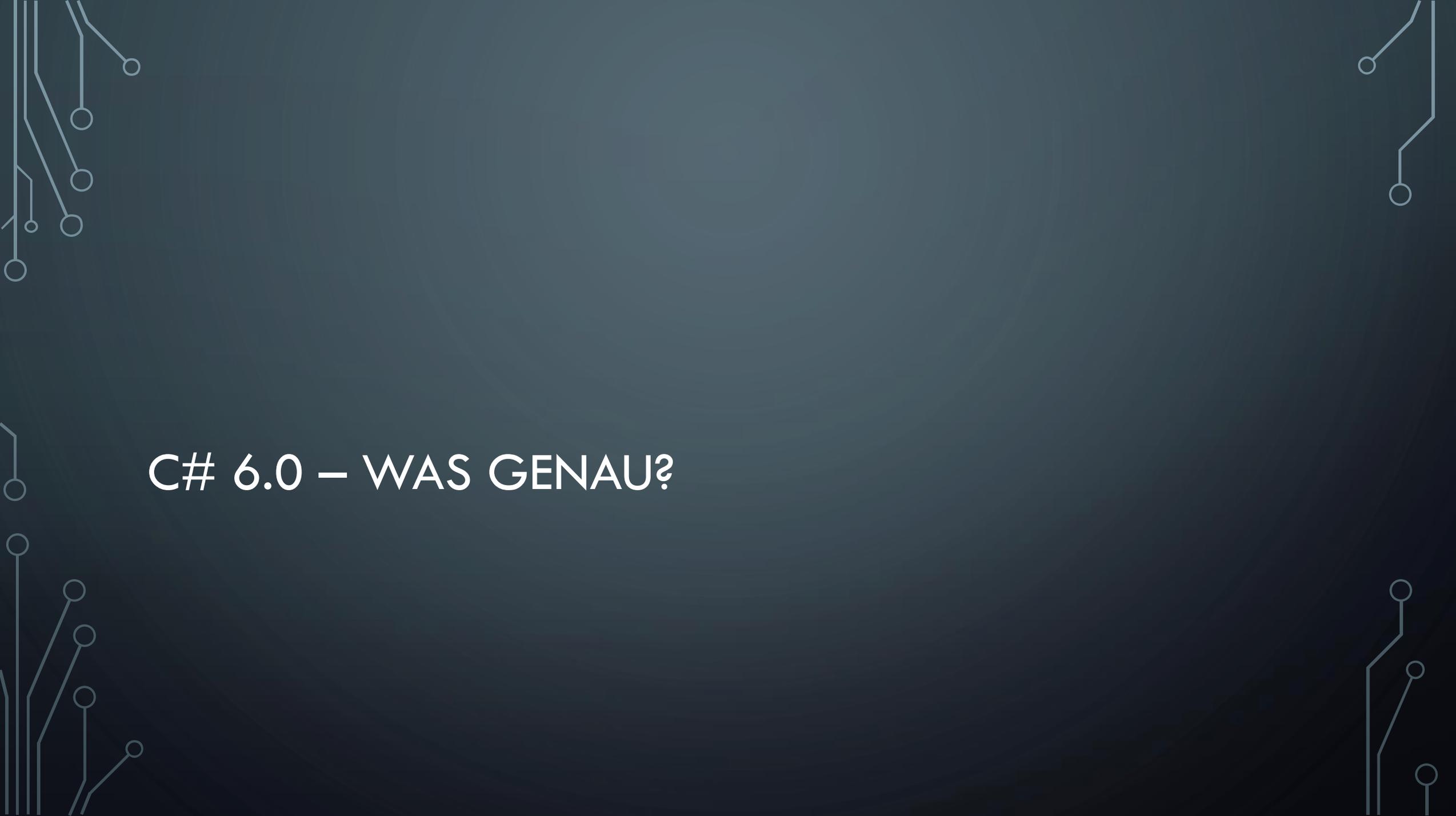
A screenshot of the GitHub Issues page for the 'dotnet/roslyn' repository, filtered to show 'Open' issues. The page displays a list of issues with their titles, labels, and authors. The issues listed include:

- Cannot mix Enum and Int with same value in one switch (Area: Compilers, Bug, New Language Feature - Pattern Matching)
- Incorrect "Cast is redundant" when switch on an 'object' (Area: Backlog, Area: Compilers, Area: IDE, Bug)
- Get rid of the old codeplex reference source (Area: Infrastructure, Documentation)
- Roslyn reference source - "403 you don't have permission" for some pages, not others (Area: Infrastructure, Documentation)
- [Proposal] Method group extra parameter binding (Area: Language Design)
- Wrong sequence points generated for if in async void method (Area: Compilers, Bug, Interactive: Debugging)
- Project.OutputFilePath should point to bin\Debug\ConsoleApplication1.exe, not to obj\Debug\ConsoleApplication1.exe (Area: IDE)
- Project system should support macros in Debug's Command Line Arguments (Area: IDE)



A screenshot of the 'Discussions under C# Language Design' page. The page shows a list of discussions with their titles, dates, and authors. The discussions listed include:

- non-nullable reference types (the one billion \$ mistake) (Nov 22, 2015)
- Primary Constructors (Nov 30, 2015)
- applying CodeFix then save to disk (Dec 29, 2015)
- Generics usage in extensions methods (Jan 9, 2016)
- extend object initializer syntax to arbitrary expressions (Jan 11, 2016)

The background is a dark blue gradient. In the four corners, there are decorative white line-art patterns resembling circuit traces or branching structures. These patterns consist of thin lines that branch out and terminate in small circles, creating a sense of connectivity and flow.

C# 6.0 – WAS GENAU?

C# 6.0 FEATURES

- **Null-conditional operators**
- **Auto-Property Initializers**
- **Getter-only Auto-Properties**
- **Nameof Expressions**
- **Expression-bodied Functions and Properties**
- **String Interpolation**
- **Using static**
- **Index Initializers**
- **Exception filters**
- **await in catch and finally Blocks**

NULL-CONDITIONAL OPERATORS

Ziel:

```
Console.WriteLine(  
    UserGroups.All[0].Events.First().Title  
);
```

Vorher:

```
if (UserGroups.All != null)  
{  
    if (UserGroups.All[0] != null && UserGroups.All[0].Events != null)  
    {  
        if (UserGroups.All[0].Events.FirstOrDefault() != null)  
        {  
            Console.WriteLine(UserGroups.All[0].Events.First().Title);  
        }  
    }  
}
```

C# 6.0:

```
Console.WriteLine(  
    UserGroups.All?[0].Events?.FirstOrDefault()?.Title  
);
```



AUTO-PROPERTY INITIALIZERS

Vorher:

```
public class UserGroup
{
    public string Name { get; set; }
    public List<Event> Events { get; set; }

    public UserGroup()
    {
        Events = new List<Event>();
    }
}
```

C# 6.0:

```
public class UserGroup
{
    public string Name { get; set; }
    public List<Event> Events { get; set; } = new List<Event>();
}
```

{ get; set; } = ...

GETTER-ONLY AUTO-PROPERTIES

Vorher:

```
public class UserGroup
{
    public string Name { get; set; }
    public List<Event> Events { get; set; } = new List<Event>();
    public string BestProgrammingLanguage { get; }

    public UserGroup()
    {
        BestProgrammingLanguage = "C#";
    }
}
```

C# 6.0:

```
public class UserGroup
{
    public string Name { get; set; }
    public List<Event> Events { get; set; } = new List<Event>();
    public string BestProgrammingLanguage { get; } = "C#";
}
```

{ get; } = ...

NAMEOF EXPRESSIONS

Vorher:

```
public void Invite(List<Attendee> attendees)
{
    if (attendees == null)
    {
        throw new ArgumentNullException("attendees");
    }
    // ...
}
```

C# 6.0:

```
public void Invite(List<Attendee> attendees)
{
    if (attendees == null)
    {
        throw new ArgumentNullException(nameof(attendees));
    }
    // ...
}
```

nameof(...)

EXPRESSION-BODIED FUNCTIONS AND PROPERTIES

Vorher:

```
public DateTime CalculateBestTimeForUsingCSharp6()  
{  
    return DateTime.Now;  
}
```

C# 6.0:

```
public DateTime CalculateBestTimeForUsingCSharp6() => DateTime.Now;
```

() = > ...

STRING INTERPOLATION

Vorher:

```
int cSharpVersion = 6;  
string strAlt = string.Format("Hier geht es um C# {0}", cSharpVersion);
```

Änderungen gegenüber der Preview:

Syntaxänderung: Anstatt „\{...}“
jetzt \$“{...}“

C# 6.0:

```
int cSharpVersion = 6;  
string strNeu = $"Hier geht es um C# {cSharpVersion}";  
  
string strFormatted = $"Hier geht es um C# {cSharpVersion:0.0}";
```

\$“Text {i}“

USING STATIC

Vorher:

```
Console.WriteLine("Ohne using static System.Console");
```

C# 6.0:

```
using static System.Console;
```

```
WriteLine("Mit using static System.Console");
```

Änderungen gegenüber der Preview:

Schlüsselwort **static** zwingend
notwendig!

```
using static Namespace.Type;
```

INDEX INITIALIZERS

Vorher:

```
var dictAlt = new SortedDictionary<int, string>()  
{  
    { 2, "dot" },  
    { 3, "NET" },  
    { 1, "IN" },  
};
```

C# 6.0:

```
var dictNeu = new SortedDictionary<int, string>()  
{  
    [2] = "dot",  
    [3] = "NET",  
    [1] = "IN",  
};
```

{ [key] = value, ... }

EXCEPTION FILTERS

Vorher:

```
try
{
    var userGroupEvent = new Event() { Title = "C# 6.0" };
    userGroupEvent.Invite(null);
}
catch (ArgumentNullException ex)
{
    if (ex.ParamName == "attendees")
    {
        Console.Error.WriteLine("Ein null Parameter bedeutet natürlich ALLE .NET-Entwickler!");
        return;
    }
    Console.Error.WriteLine("Welche Argumente denn noch?");
}
```

C# 6.0:

```
try
{
    var userGroupEvent = new Event() { Title = "C# 6.0" };
    userGroupEvent.Invite(null);
}
catch (ArgumentNullException ex) when (ex.ParamName == "attendees")
{
    Console.Error.WriteLine("Ein null Parameter bedeutet natürlich ALLE .NET-Entwickler!");
}
catch (ArgumentNullException ex)
{
    Console.Error.WriteLine("Welche Argumente denn noch?");
}
```

Änderungen gegenüber der Preview:

Vorher Schlüsselwort *if* statt *when*

catch(...) when (...)

AWAIT IN CATCH AND FINALLY BLOCKS

Vorher:
(nicht möglich)

C# 6.0:

```
try
{
    //System.IO.File.ReadAllText(@"C:\Datei_existiert_nicht.txt"); // Achtung: Statische File-Methoden nicht asynchron!
    using (var streamReader = new System.IO.StreamReader(@"C:\Datei_existiert_nicht.txt"))
    {
        string text = await streamReader.ReadToEndAsync();
        Console.WriteLine(text);
    }
}
catch (Exception ex)
{
    await Console.Error.WriteLineAsync("Async Error:");
    await Console.Error.WriteLineAsync(ex.ToString());
}
finally
{
    await Task.Delay(TimeSpan.FromSeconds(1));
    await Console.Error.WriteLineAsync("1 Sekunde lang aufgeräumt!");
}
```

catch(...) { await ... }

VERSCHOBENE ODER ENTFERENTE C# 6.0 FEATURES

- **Primary constructor**

```
public class Customer(string firstName, string lastName) {..}
```

- **Binary literals**

```
int nineteen = 0b10011;
```

- **Digit separators**

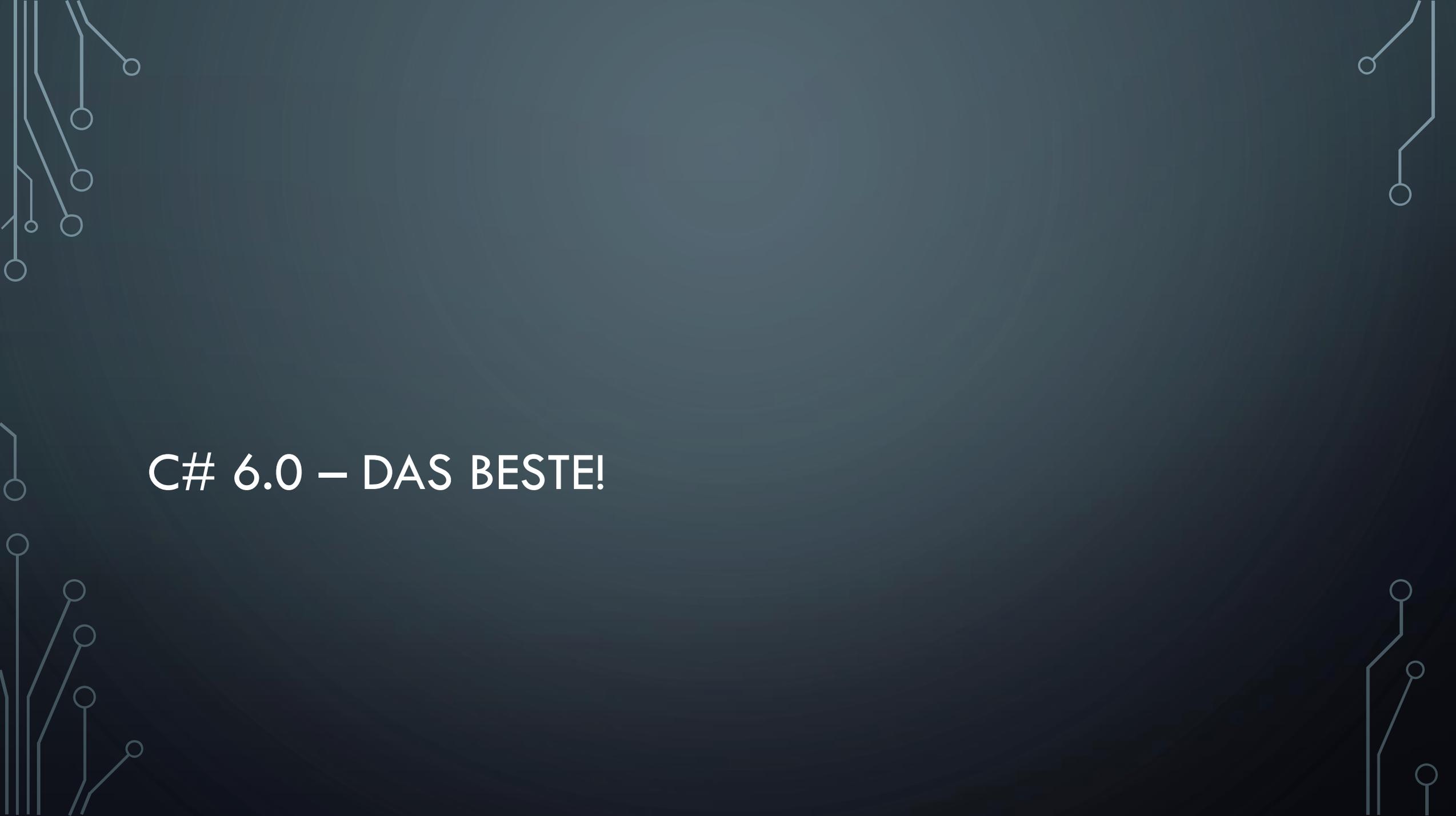
```
int dec = 33_554_432; int bin = 0b1001_1010_0001_0100;
```

- **Params IEnumerable**

```
void Method(params Arguments<int> args) { ... }
```

- **Inline declaration for out params**

```
int.TryParse(input, out var result)
```

The background is a dark blue gradient. In the four corners, there are white, stylized circuit board traces. These traces consist of straight lines that turn at right angles, ending in small circles that represent components or connection points. The traces are more densely packed in the corners and become sparser towards the center.

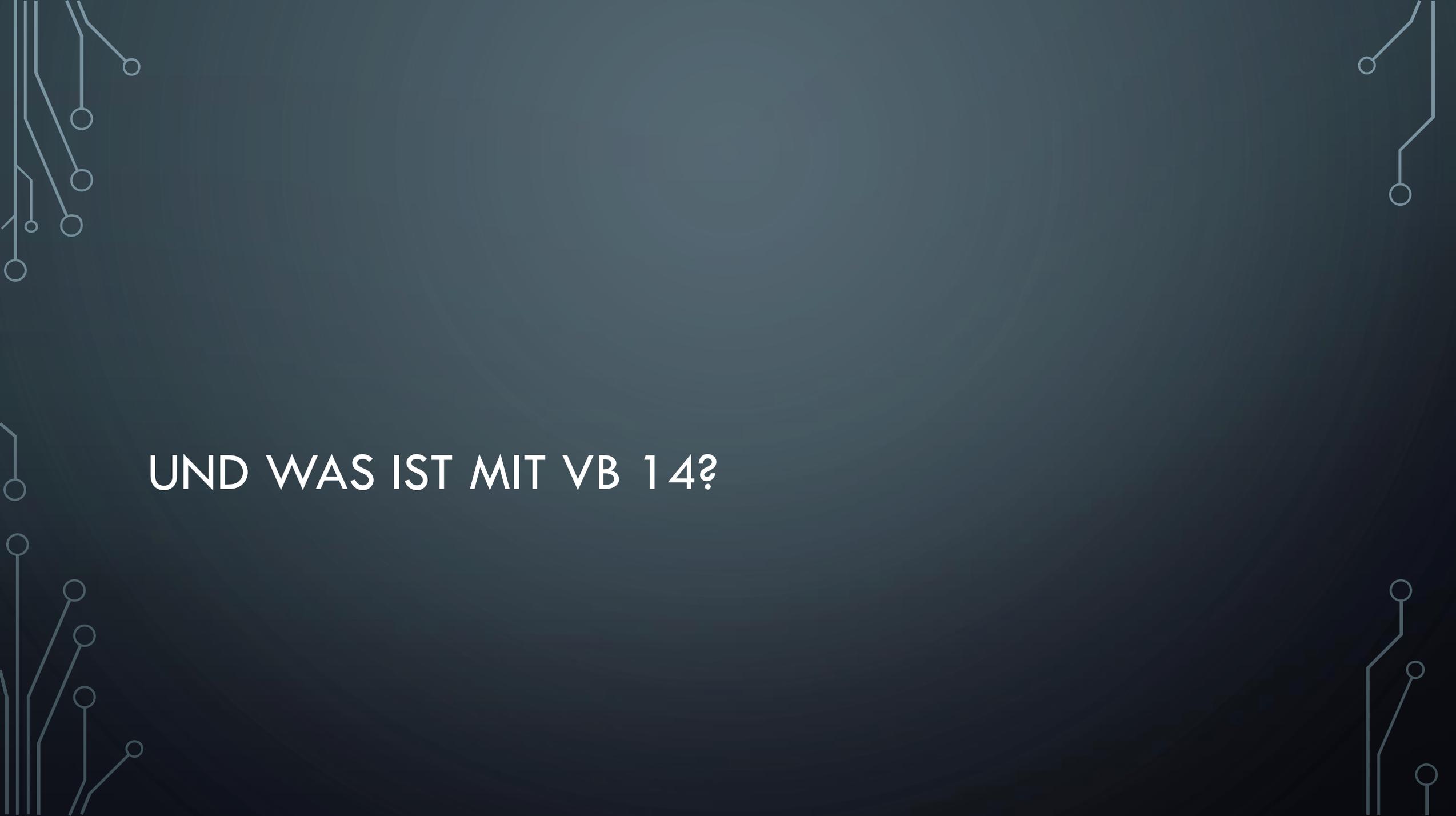
C# 6.0 – DAS BESTE!

C# 6.0 – TOP 3

1. Null-conditional operators `? .`

2. String Interpolation `$“Text {i}“`

3. Auto-Property Initializers `{ get; set; } = ...`

The image features a dark blue background with white, stylized circuit board traces in the corners. These traces form various geometric shapes and paths, some ending in small circles, resembling a network or data flow diagram. The central text is in a clean, white, sans-serif font.

UND WAS IST MIT VB 14?

VB 14 – EINE VOLLSTÄNDIGE LISTE

<https://github.com/dotnet/roslyn/wiki/New-Language-Features-in-VB-14>

Null-conditional operators

This new operator is a convenient shorthand for the many occasions when you have to check for null:

```
Dim x = customer.Address?.Country

' is shorthand for
Dim _temp = customer.Address
Dim x = If(_temp IsNot Nothing, _temp.Address.Country, Nothing)
```

You can also use it in a sequence and you can mix with the regular `.` operator, e.g. `a?.b.c?.d`. It reads left-to-right. Any null value before a `?.` will just stop the sequence short, and any null value before a `.` will raise a `NullReferenceException` as usual.

For a string value like `customer.Address?.Name`, if it stops short then the result is a null value typed

VERGLEICH ZWISCHEN C# 6.0 UND VB14

(1/2)

| Feature | Example | C# 6.0 | VB 14 |
|--|--|--------|--------|
| Auto-property initializers | <code>public int X { get; set; } = x;</code> | Added | Exists |
| Getter-only auto-properties | <code>public int Y { get; } = y;</code> | Added | Added |
| Ctor assignment to getter-only autoprops | <code>Y = 15</code> | Added | Added |
| Parameterless struct ctors | <code>Structure S : Sub New() : End Sub : End Structure</code> | Added | Added |
| Using static members | <code>using System.Console; ... Write(4);</code> | Added | Exists |
| Dictionary initializer | <code>new JObject { ["x"] = 3, ["y"] = 7 }</code> | Added | No |
| Await in catch/finally | <code>try ... catch { await ... } finally { await ... }</code> | Added | No |
| Exception filters | <code>catch(E e) if (e.Count > 5) { ... }</code> | Added | Exists |
| Partial modules | <code>Partial Module M1</code> | N/A | Added |
| Partial interfaces | <code>Partial Interface I1</code> | Exists | Added |
| Multiline string literals | <code>"Hello<newline>World"</code> | Exists | Added |
| Year-first date literals | <code>Dim d = #2014-04-03#</code> | N/A | Added |
| Line continuation comments | <code>Dim addr = From c in Customers ' comment</code> | N/A | Added |
| ... | ... | ... | ... |

VERGLEICH ZWISCHEN C# 6.0 UND VB14

(2/2)

| Feature | Example | C# 6.0 | VB 14 |
|--|--|--------|---------|
| ... | ... | ... | ... |
| TypeOf IsNot | If TypeOf x IsNot Customer Then ... | N/A | Added |
| Expression-bodied members | public double Dist => Sqrt(X * X + Y * Y); | Added | No |
| Null propagation | customer?.Orders?[5]?.\$price | Added | Added |
| String interpolation | (\$"{p.First} {p.Last} is {p.Age} years old.") | Added* | Planned |
| nameof operator | string s = nameof(Console.Write); | Added* | Planned |
| #pragma | #Disable Warning BC40008 | Added | Added |
| Smart name resolution | | N/A | Added |
| ReadWrite props can implement ReadOnly | | Exists | Added |
| #region inside methods | | Exists | Added |
| Overloads inferred from Overrides | | N/A | Added |
| CObj in attributes | | Exists | Added |
| CRef and parameter name | | Exists | Added |
| Extension Add in collection initializers | | Added | Exists |
| Improved overload resolution | | Added | N/A |

The image features a dark blue background with white decorative circuit-like lines in the corners. These lines consist of straight segments connected by small circles, resembling a stylized PCB or network diagram. The lines are positioned in the top-left, top-right, bottom-left, and bottom-right corners, framing the central text.

C# 7 – EIN BLICK IN DIE ZUKUNFT!

VORSCHAU AUF C# 7

- **Local Functions**

```
void Method() { int localFunction(int a, int b) => a + b; }
```

- **Tuples**

```
(int x, int y) t1 = (23, 45); (int a, int b) t2 = t1;
```

- **Pattern Matching**

```
var c = new Customer(Name: "Mayer");  
SpecialCustomer sc = c with { Discount = 0.2 };
```

```
if(x is SpecialCustomer && x.Discount > 0.15) { ... }
```

- **ref locals and ref returns**

```
public static ref int Max(ref int first, ref int second, ref int third)  
{  
    ref int max = first > second ? ref first : ref second;  
    return max > third ? ref max : ref third;  
}
```

- ...

VORSCHAU AUF C# 7

- ...

- **Binary Literals**

```
int nineteen = 0b10011;
```

- **Digit Separators**

```
int dec = 33_554_432; int bin = 0b1001_1010_0001_0100;
```

- **async Main**

```
async Task Main();  
async Task<int> Main();  
async Task Main(string[]);  
async Task<int> Main(string[]);
```

- **out var**

```
int.TryParse(input, out var result)
```