

Curso de **Buenas Prácticas y Código Limpio en C#**

Miguel Teheran

Prerrequisitos

Prerrequisitos

- Conocimiento en **C# y .NET**
- Conocimiento de **Git**
- **Visual Code o Visual Studio**
- **.NET 6 o superior**

Terminología

Buenas prácticas o Best practices

Buenas prácticas

- Estándares comprobados y verificados
- Resuelven desafíos de escenarios comunes
- Brindan guías fáciles de aprender y comprender
- Permiten tener una estructura similar para múltiples proyectos

Código limpio o Clean Code

Código limpio

- Un código que sigue las buenas prácticas
- Código fácil de entender y analizar
- Código fácil de mantener
- Código fácil de actualizar
- Código fácil de escalar

Cómo logramos **código limpio**

- Mantener bajo acoplamiento
- Utilizar sintaxis simple y moderna
- Evitar incorporar muchas librerías de terceros
- Distribución de responsabilidades
- Creación de componentes pequeños

Deuda técnica o Technical debt

Deuda técnica

- Se refiere a los problemas técnicos que puede tener un software que son adquiridos durante la fase de desarrollo y que deben ser solucionados en el futuro
- Puede estar presente en cualquier aspecto de una aplicación como: Seguridad, rendimiento, escalabilidad, etc...
- A veces puede resultar más costosa que el mismo desarrollo

Refactoring

Refactoring

- Proceso de cambios sobre un código para el mejoramiento en cualquier aspecto (rendimiento, seguridad, legibilidad, soporte, etc...)
- Lo más importante es hacer los cambios manteniendo la lógica de negocio sin afectaciones

Refactoring

- Debe ser recurrente
- Disminuye la deuda técnica a futuro

Nombramiento / Naming

Variables



```
//Mal:  
int d;
```

```
//Bien:  
int daySinceModification;
```


Método



//Mal:

```
public List<Users> getUsers()
```

//Bien:

```
public List<Users> GetActiveUsers()
```

Classes



```
//Mal:  
public class ClassUser2
```

```
//Bien:  
public class User
```

Code Smells

Transmite la
sensación de que
**algo en el código
está mal**

Ejercicio

**Analiza la siguiente
palabra: Excelente**

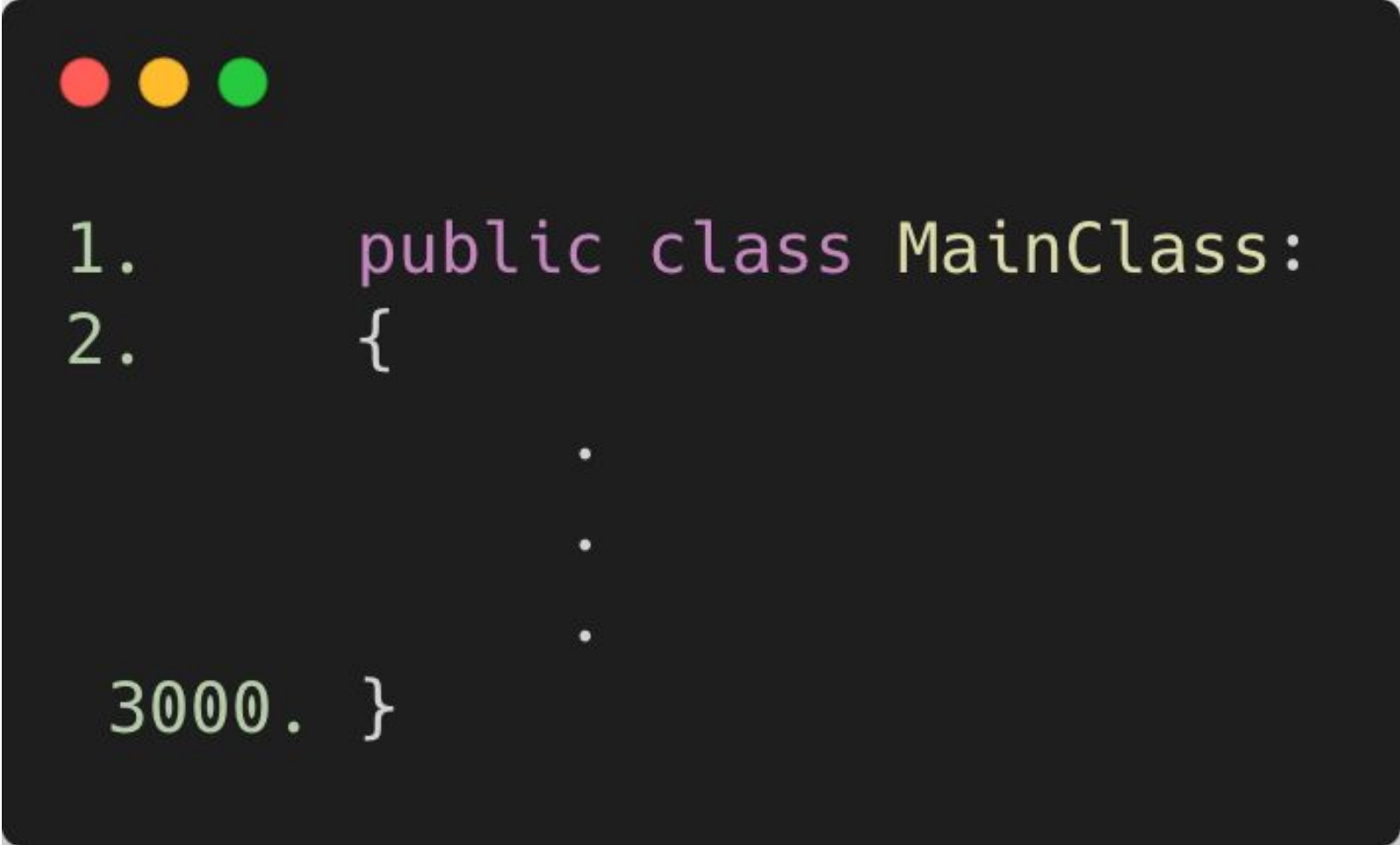
Nombramiento



```
//Mal:  
int d;
```

```
//Bien:  
int daySinceModification;
```

Métodos y clases grandes



```
1.      public class MainClass:
2.      {
           .
           .
           .
3000.  }
```

Demasiados parámetros



```
public double Calculation(int, number1, int number2, string title,  
                           Mode mode, int decimals, string[] params )
```


Números mágicos



```
if (variable == 1)
{
    ShowMenuAdd( );
}
else if (variable == 2)
{
    ShowMenuDos( );
}
else if (variable == 3)
{
    ShowMenuTres( );
}
```

Principio DRY

**Don't Repeat
Yourself**

Principio KISS

**Keep It
Simple Stupid**

**Keep It Short
and Simple**

Uso correcto de try catch

Try catch es para excepciones

- Son escenarios excepcionales o inusuales en el código
- Se implementa donde se desee devolver un valor específico después de realizar el control
- El rendimiento se ve afectado al capturar la excepción

Rendimiento de un método con try catch














Seleccionar C:\Personal\Udemy\TryCatchBenchmarking\ConsoleApp2\bin\Release\netcoreapp3.0\TryCatchBenchmarking.exe

Runtime=.NET Core 3.0 InvocationCount=1 IterationCount=5
LaunchCount=1 UnrollFactor=1 WarmupCount=2

	Method	number1	number2	Mean	Error	StdDev	Medi
an							
ns	GetDivision	0	0	340.00 ns	1,430.45 ns	371.48 ns	200.0000
ns	GetDivisionDivisionException	0	0	107,120.00 ns	225,769.50 ns	58,631.62 ns	81,800.0000
ns	GetDivision	0	1	420.00 ns	1,834.62 ns	476.45 ns	300.0000
ns	GetDivisionDivisionException	0	1	560.00 ns	1,898.18 ns	492.95 ns	500.0000
ns	GetDivision	0	2	140.00 ns	583.98 ns	151.66 ns	100.0000
ns	GetDivisionDivisionException	0	2	40.00 ns	251.03 ns	65.19 ns	0.0000
ns	GetDivision	1	0	220.00 ns	740.69 ns	192.35 ns	200.0000
ns	GetDivisionDivisionException	1	0	151,020.00 ns	349,252.07 ns	90,699.65 ns	120,700.0000
ns	GetDivision	1	1	110.00 ns	738.18 ns	191.70 ns	50.0000
ns	GetDivisionDivisionException	1	1	500.00 ns	816.84 ns	212.13 ns	500.0000

Evolución de C#

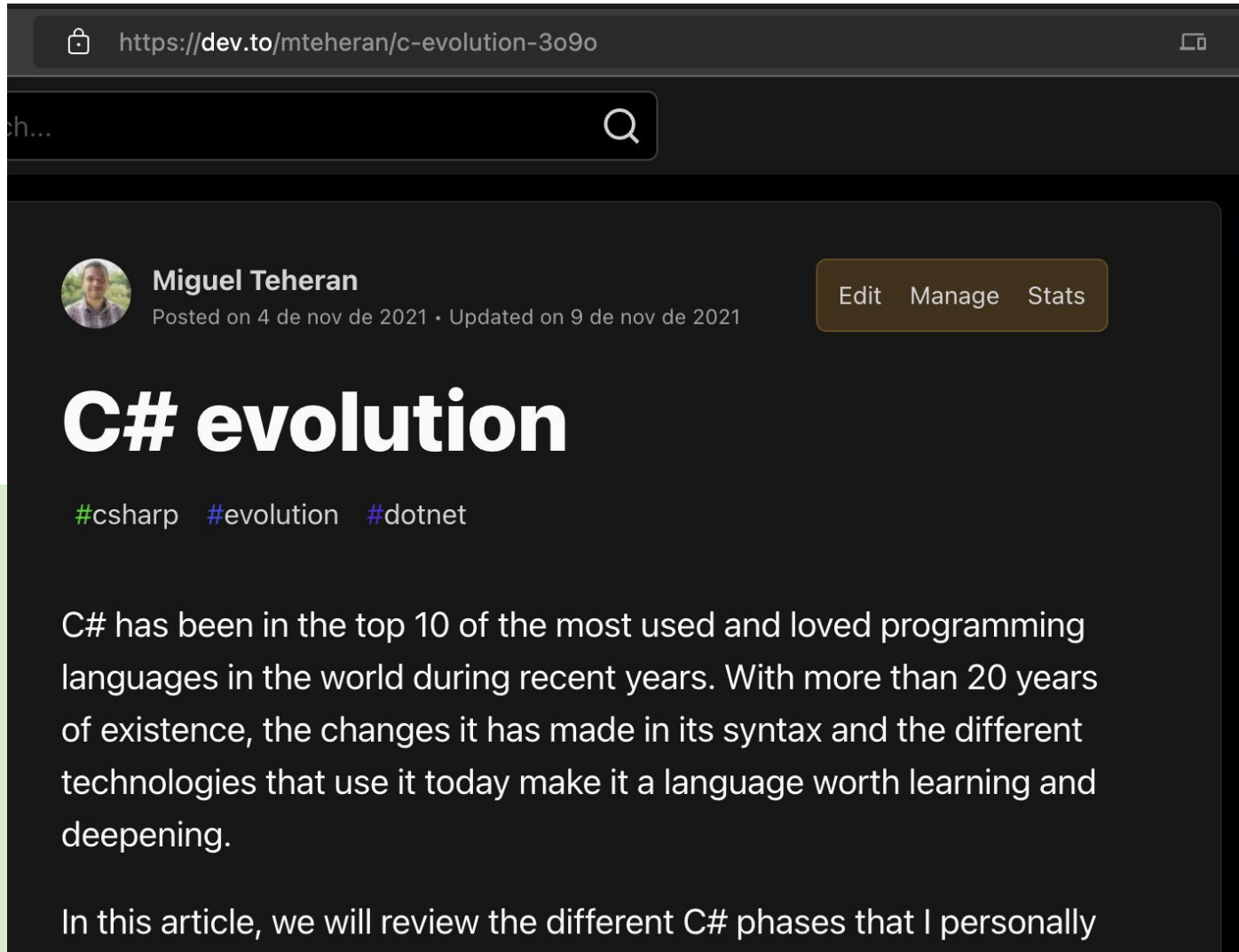
Primera Etapa

Version	Language specification			Date	.NET	Visual Studio
	Ecma	ISO/IEC	Microsoft			
C# 1.0	ECMA-334:2003, December 2002 	ISO/IEC 23270:2003, April 2003 	January 2002 	January 2002	.NET Framework 1.0	Visual Studio .NET 2002
C# 1.1 C# 1.2			October 2003 	April 2003	.NET Framework 1.1	Visual Studio .NET 2003
C# 2.0 ^[43]			ECMA-334:2006, June 2006 	ISO/IEC 23270:2006, September 2006 	September 2005  ^[c]	November 2005
C# 3.0 ^[44]	None		August 2007 	November 2007	.NET Framework 2.0 (Except LINQ) ^[45] .NET Framework 3.0 (Except LINQ) ^[45] .NET Framework 3.5	Visual Studio 2008
C# 4.0 ^[46]			April 2010		April 2010	
C# 5.0 ^[47]	ECMA-334:2017, December 2017 	ISO/IEC 23270:2018, December 2018 	June 2013 	August 2012	.NET Framework 4.5	Visual Studio 2012 Visual Studio 2013
C# 6.0 ^[48]	ECMA-334:2022, June 2022 	None	Draft 	July 2015	.NET Framework 4.6 .NET Core 1.0 .NET Core 1.1	Visual Studio 2015

Segunda Etapa

	December 2017 	December 2018 				Visual Studio 2013
C# 6.0 ^[48]	ECMA-334:2022, June 2022 	None	Draft 	July 2015	.NET Framework 4.6 .NET Core 1.0 .NET Core 1.1	Visual Studio 2015
C# 7.0 ^{[49][50]}	None		Specification proposal 	March 2017	.NET Framework 4.7	Visual Studio 2017 version 15.0
C# 7.1 ^[51]			Specification proposal 	August 2017	.NET Core 2.0	Visual Studio 2017 version 15.3 ^[52]
C# 7.2 ^[53]			Specification proposal 	November 2017		Visual Studio 2017 version 15.5 ^[54]
C# 7.3 ^[55]			Specification proposal 	May 2018	.NET Core 2.1 .NET Core 2.2 .NET Framework 4.8	Visual Studio 2017 version 15.7 ^[54]
C# 8.0 ^[56]			Specification proposal 	September 2019	.NET Core 3.0 .NET Core 3.1	Visual Studio 2019 version 16.3 ^[57]
C# 9.0 ^[58]			Specification proposal 	November 2020	.NET 5.0	Visual Studio 2019 version 16.8 ^[57]
C# 10.0 ^[59]			Specification proposal 	November 2021	.NET 6.0 .NET 6.0.1	Visual Studio 2022 version 17.0 ^[60]
C# 11.0 ^[61]	(preview)	August 2022				


Artículo recomendado



The image is a screenshot of a web browser displaying an article on the Dev.to platform. The browser's address bar shows the URL <https://dev.to/mteheran/c-evolution-3o9o>. Below the address bar is a search bar with a magnifying glass icon. The article header features a circular profile picture of Miguel Teheran, his name, and the text "Posted on 4 de nov de 2021 • Updated on 9 de nov de 2021". To the right of the header are three buttons: "Edit", "Manage", and "Stats". The article title "C# evolution" is displayed in a large, bold, white font. Below the title are three hashtags: #csharp, #evolution, and #dotnet. The main body of the article begins with a paragraph: "C# has been in the top 10 of the most used and loved programming languages in the world during recent years. With more than 20 years of existence, the changes it has made in its syntax and the different technologies that use it today make it a language worth learning and deepening." Below this paragraph, the text "In this article, we will review the different C# phases that I personally" is visible.

<https://dev.to/mteheran/c-evolution-3o9o>

ch...

 **Miguel Teheran**
Posted on 4 de nov de 2021 • Updated on 9 de nov de 2021

Edit Manage Stats

C# evolution

#csharp #evolution #dotnet

C# has been in the top 10 of the most used and loved programming languages in the world during recent years. With more than 20 years of existence, the changes it has made in its syntax and the different technologies that use it today make it a language worth learning and deepening.

In this article, we will review the different C# phases that I personally

C# 6 - Auto Property Initializer



```
public string MyVar { get; set; } = "Hello World";
```

C# 6 - String Interpolation




```
string MyVar = "Hello World";  
Console.WriteLine($"The message is: {MyVar}")
```


C# 6 - Expression - Bodied Function



```
public int Sum2Numbers(int x, int y) => (x + y);
```


C# 6 - null conditional operator




```
DateTime? datetime = new DateTime();

//using ? asking if the value is null
var YearOfDate = datetime?.Year;

Console.WriteLine(YearOfDate);

var YearOfDateConditional = datetime?.Year ?? 0;
```

C# 7 - Local functions



```
static void Main(string[] args)
{
    int Sum(int x, int y)
    {
        return x + y;
    }

    Console.WriteLine(Sum(10, 20));
    Console.ReadKey();
}
```

C# 7 - Digital separator



```
public const long BillionsAndBillions = 100_000_000_000;
```

C# 7 - Tuples




```
var named = (first: "one", second: "two");
```

```
//method returning a tuple
```

```
(bool bresult, double dcalculation) = Tuples.CalculationTuple();
```

C# 8 - Switch as expression



```
//switch as expression
int intValueForSwitch = 0;

var state = (intValueForSwitch) switch
{
    (0) => "Zero",
    (1) => "One",
    (2) => "Two",
    _   => "NO valid number"
};
```

C# 9 - Top - level statement



The image shows a C# code editor window with two files. The top file is a standard C# program with a namespace and a class. The bottom file is a top-level statement program. Both files have a red, yellow, and green window control icon at the top left.

```
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}

using System;

Console.WriteLine("Hello World!");
```

C# 10 - Global using



```
Console.WriteLine("Hello World!");
```

Comentarios

Malos comentarios

What a typical code comment looks like

```
// The User class.  
public class User {  
    ...  
}
```



Malos comentarios

**DON'T COMMENT BAD
CODE - REWRITE IT.**

- Brian Kernighan

atlaz.io

Recomendaciones para comentar

- No abuses de los comentarios
- Si los comentarios son obvios, no los pongas
- Escribe código fácil de entender como filosofía
- No comentes la trazabilidad de cambios
- Mantén los comentarios cortos y claros
- No dejes código comentado

Cierre del curso

Recomendaciones finales

- Lee sobre buenas prácticas
- Realiza code review entre colegas, así como entre compañeros y compañeras de trabajo
- Realiza refactoring luego de implementar tu código

Recomendaciones finales

- Utiliza herramientas que te ayuden a limpiar tu código
- Implementa estándares y prácticas comunes en tu empresa o proyecto