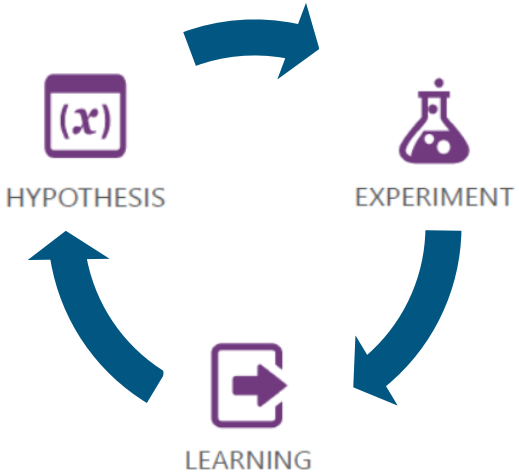
Clean Code and Technical Debt

Michael Kaufmann | MVP | Senior Manager

„THE NEXT BIG BLUE-COLLAR JOB IS CODING“

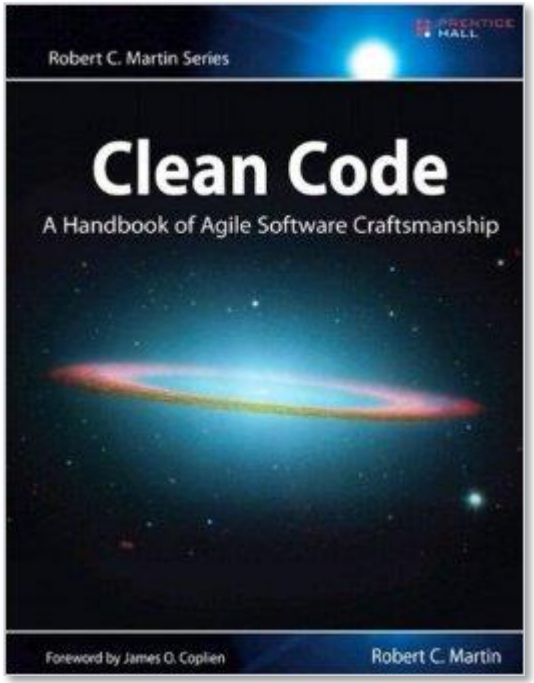
https://www.wired.com/2017/02/programming-is-the-new-blue-collar-job/

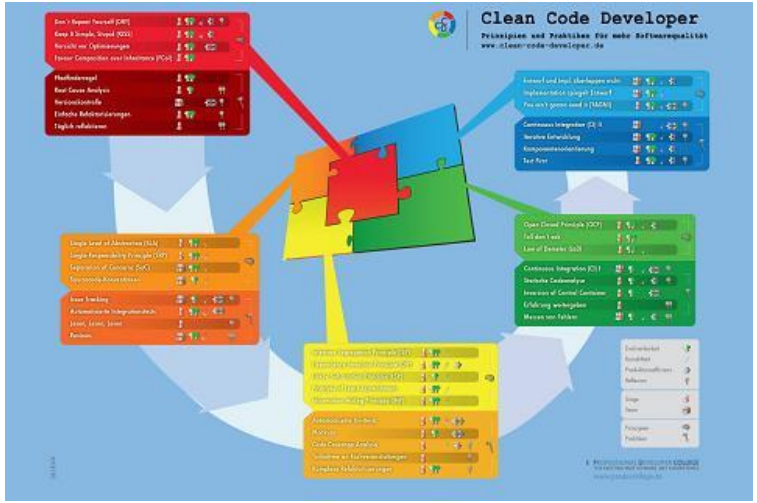
Programmierst du noch – oder experimentierst du schon?




Michael Kaufmann
Head of Development
Alegri
MVP
Microsoft®
Most Valuable
Professional
Microsoft
CERTIFIED
Solutions Developer
Application Lifecycle
Management
PRINCE2
PRACTITIONER
CPD
ITIL
FOUNDATION
CPD
redhat.
CERTIFIED
ENGINEER
Microsoft
CERTIFIED
Solutions Developer
Microsoft .NET
Blog: http://writeabout.net
Twitter: @mike_kaufmann

Was ist Clean Code?

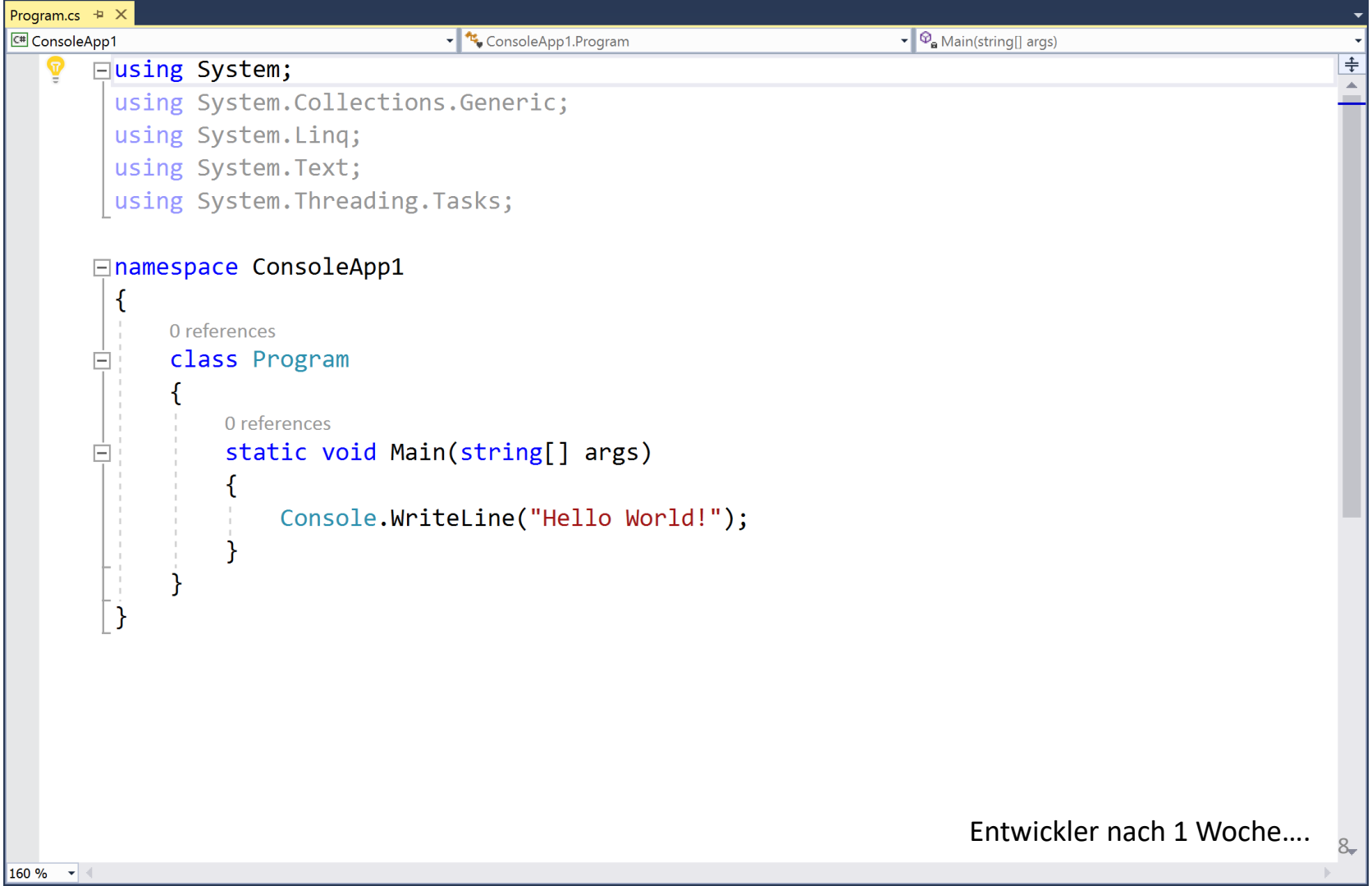




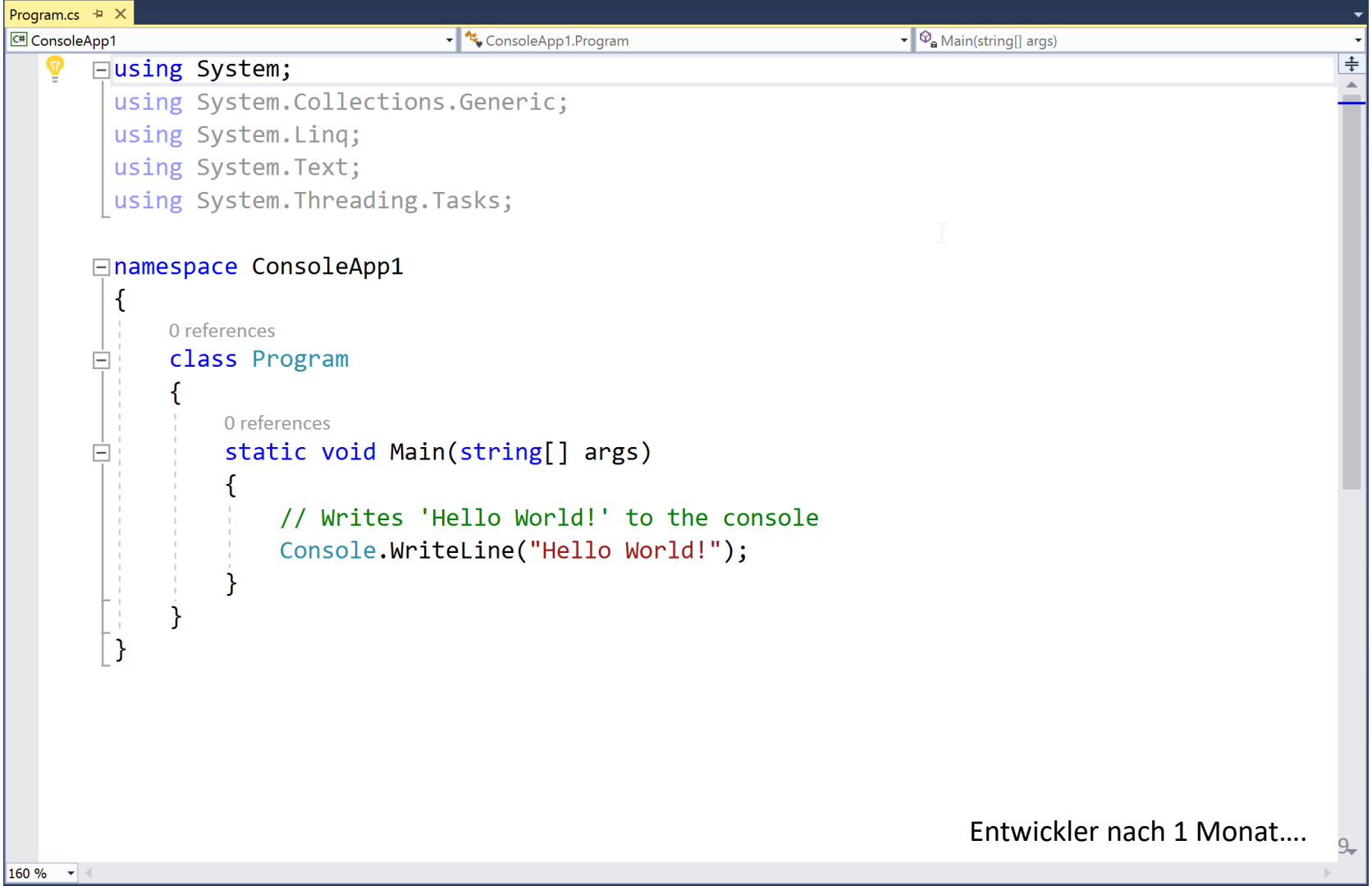
clean code developer

Grades

- **black** → commitment to improve
- **red** → basic principles and practices
- **orange** → automate things
- **yellow** → continous testing
- **green** → continous integration
- **blue** → continous deployment
- **white** → all of the above & restart with red



Entwickler nach 1 Woche….



Entwickler nach 1 Monat….

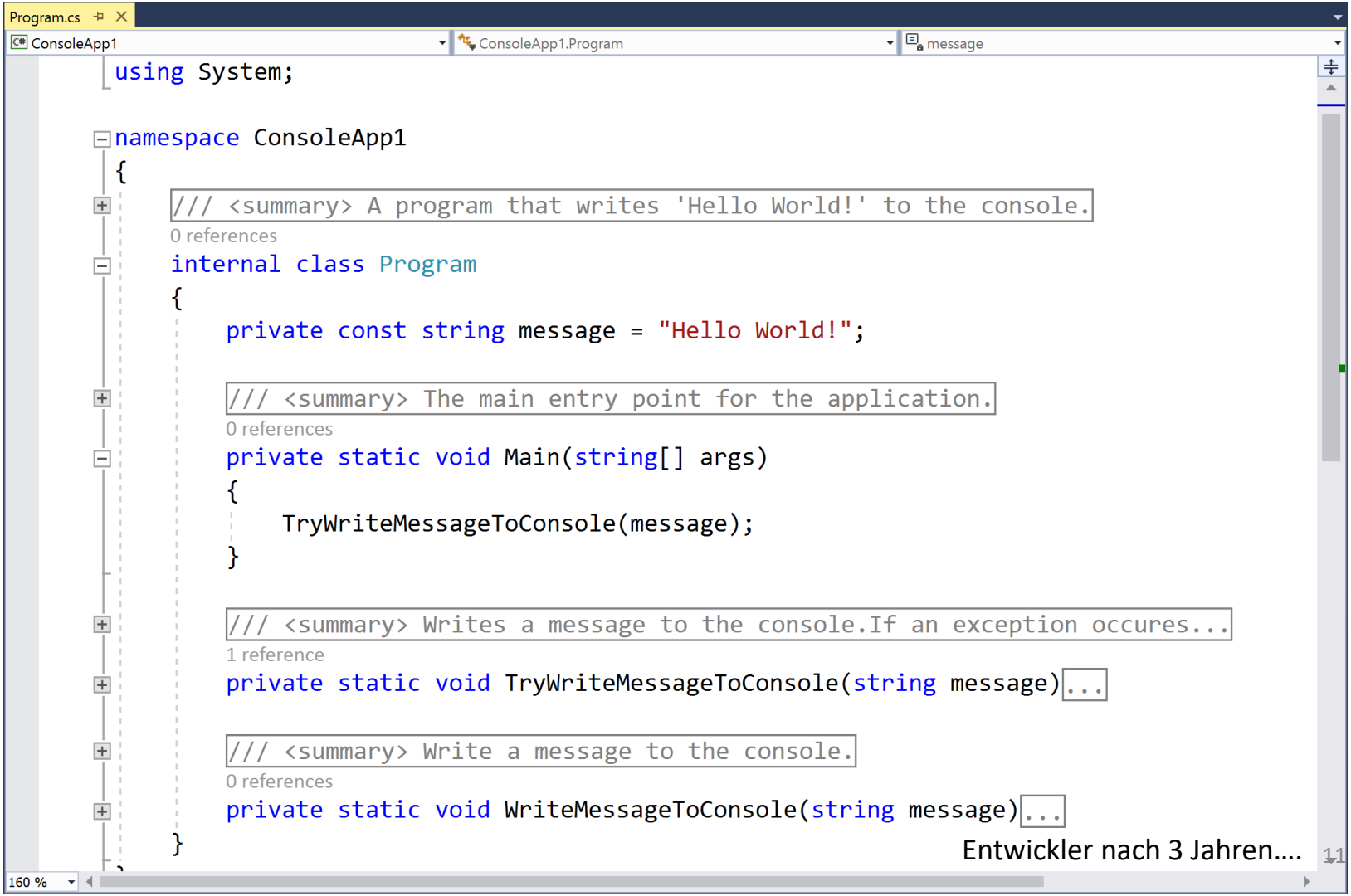
Program.cs

ConsoleApp1 | ConsoleApp1.Program | message

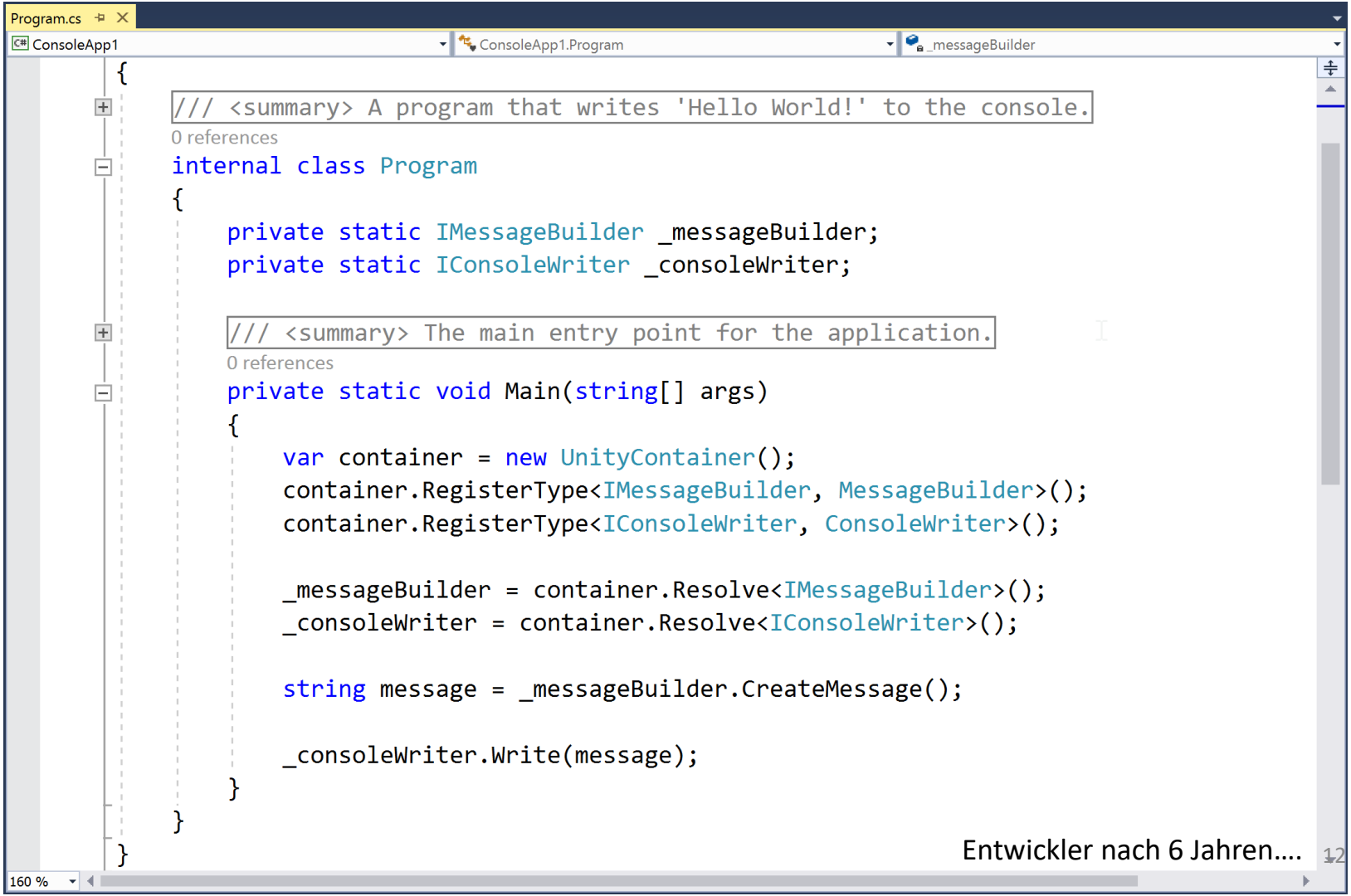
```
using System;

namespace ConsoleApp1
{
    /// <summary>
    /// A program that writes 'Hello World!' to the console.
    /// </summary>
    0 references
    internal class Program
    {
        // The message that is written to the console
        private const string message = "Hello World!";

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        /// <param name="args">Optional arguments that are passed in from the console.
        0 references
        private static void Main(string[] args)
        {
            // Writes the message to the console
            Console.WriteLine(message);
        }
    }
}
```

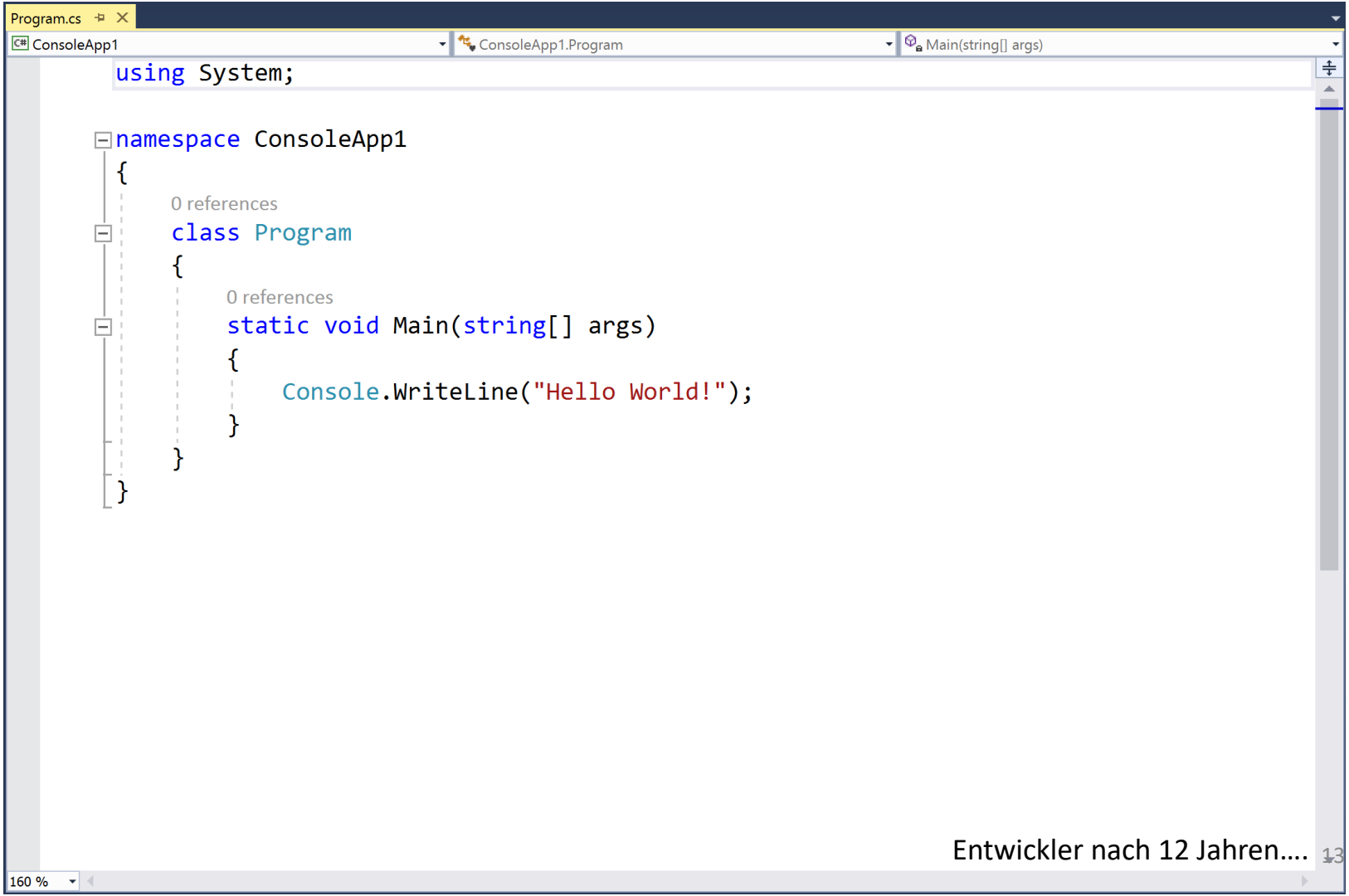
Entwickler nach 1 Jahr....

160 %



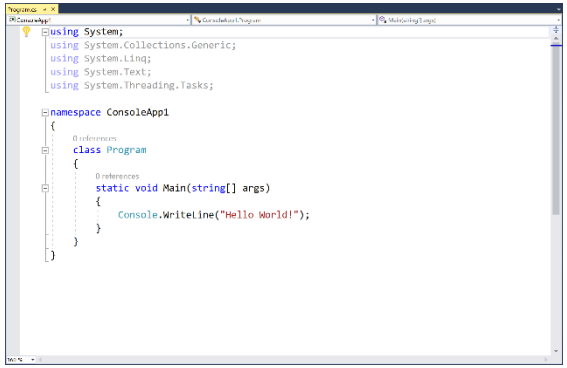
Entwickler nach 3 Jahren....


Program.cs
ConsoleApp1
ConsoleApp1.Program
_messageBuilder
{
/// <summary> A program that writes 'Hello World!' to the console.
0 references
internal class Program
{
private static IMessageBuilder _messageBuilder;
private static IConsoleWriter _consoleWriter;
/// <summary> The main entry point for the application.
0 references
private static void Main(string[] args)
{
var container = new UnityContainer();
container.RegisterType<IMessageBuilder, MessageBuilder>();
container.RegisterType<IConsoleWriter, ConsoleWriter>();
_messageBuilder = container.Resolve<IMessageBuilder>();
_consoleWriter = container.Resolve<IConsoleWriter>();
string message = _messageBuilder.CreateMessage();
_consoleWriter.Write(message);
}
}
}
160 %
Entwickler nach 6 Jahren....
12



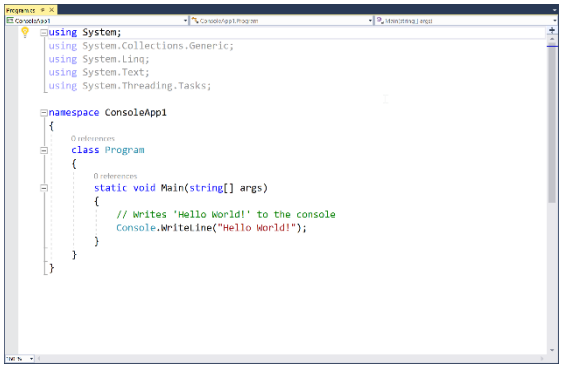
Entwickler nach 12 Jahren....

Entwickler nach 1 Woche….



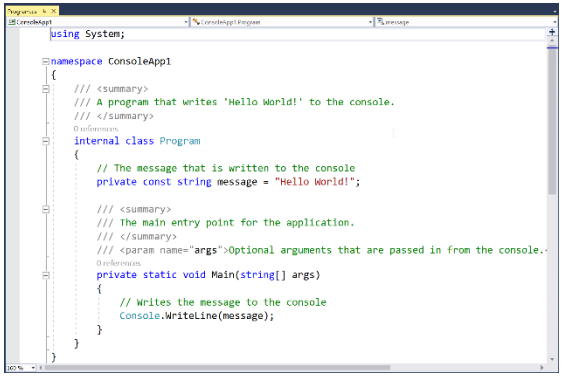
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Entwickler nach 1 Monat….



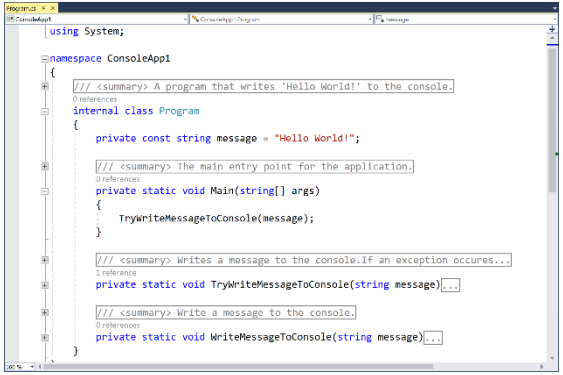
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Writes 'Hello World!' to the console
            Console.WriteLine("Hello World!");
        }
    }
}
```

Entwickler nach 1 Jahr….



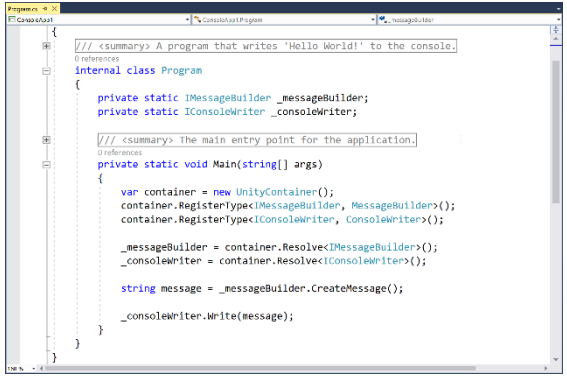
```
using System;

namespace ConsoleApp1
{
    /// <summary>
    /// A program that writes 'Hello World!' to the console.
    /// </summary>
    internal class Program
    {
        // The message that is written to the console
        private const string message = "Hello World!";

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        /// <param name="args">Optional arguments that are passed in from the console.
        private static void Main(string[] args)
        {
            // Writes the message to the console
            Console.WriteLine(message);
        }
    }
}
```

Entwickler nach 3 Jahren….



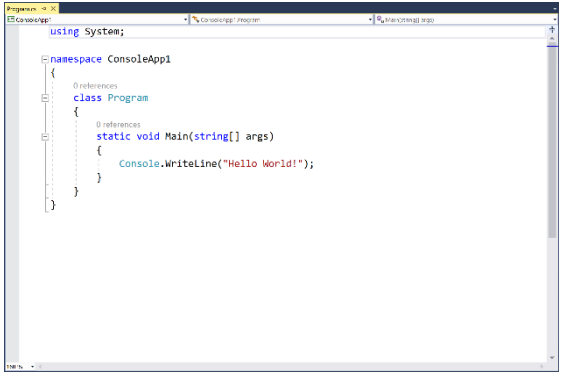
```
using System;

namespace ConsoleApp1
{
    /// <summary> A program that writes 'Hello World!' to the console.
    internal class Program
    {
        private const string message = "Hello World!";

        /// <summary> The main entry point for the application.
        private static void Main(string[] args)
        {
            TryWriteMessageToConsole(message);
        }

        /// <summary> Writes a message to the console.If an exception occures...
        private static void TryWriteMessageToConsole(string message)...

        /// <summary> Write a message to the console.
        private static void WriteMessageToConsole(string message)...
    }
```

Entwickler nach 6 Jahren….



```
{
    /// <summary> A program that writes 'Hello World!' to the console.
    internal class Program
    {
        private static IMessageBuilder _messageBuilder;
        private static IConsoleWriter _consoleWriter;

        /// <summary> The main entry point for the application.
        private static void Main(string[] args)
        {
            var container = new UnityContainer();
            container.RegisterType<IMessageBuilder, MessageBuilder>();
            container.RegisterType<IConsoleWriter, ConsoleWriter>();

            _messageBuilder = container.Resolve<IMessageBuilder>();
            _consoleWriter = container.Resolve<IConsoleWriter>();

            string message = _messageBuilder.CreateMessage();

            _consoleWriter.Write(message);
        }
    }
}
```

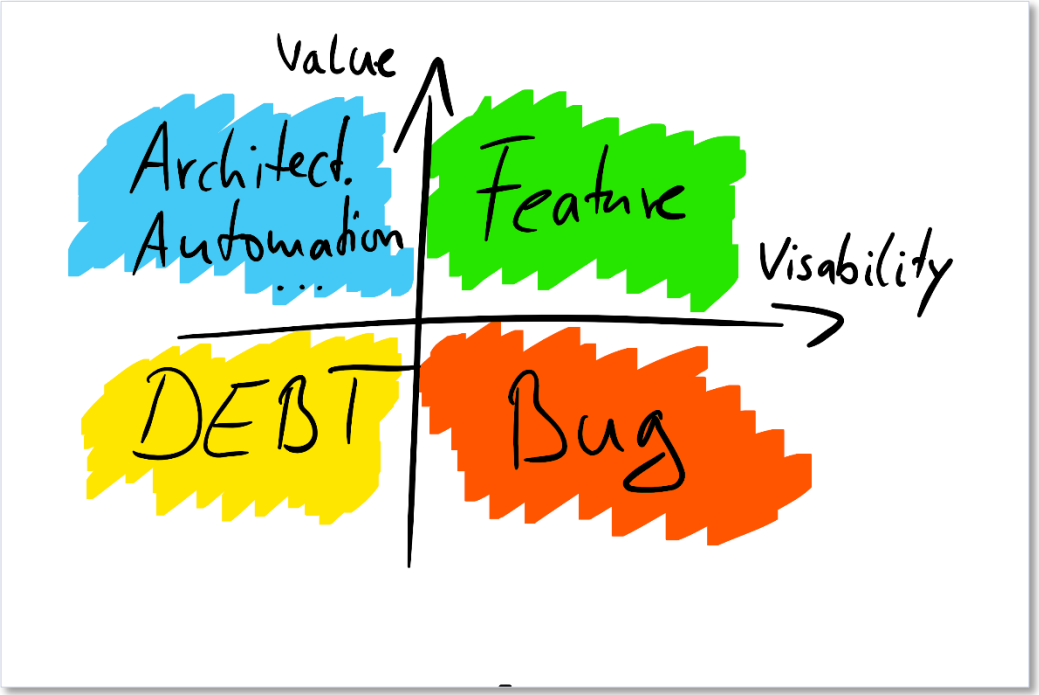
Entwickler nach 12 Jahren….

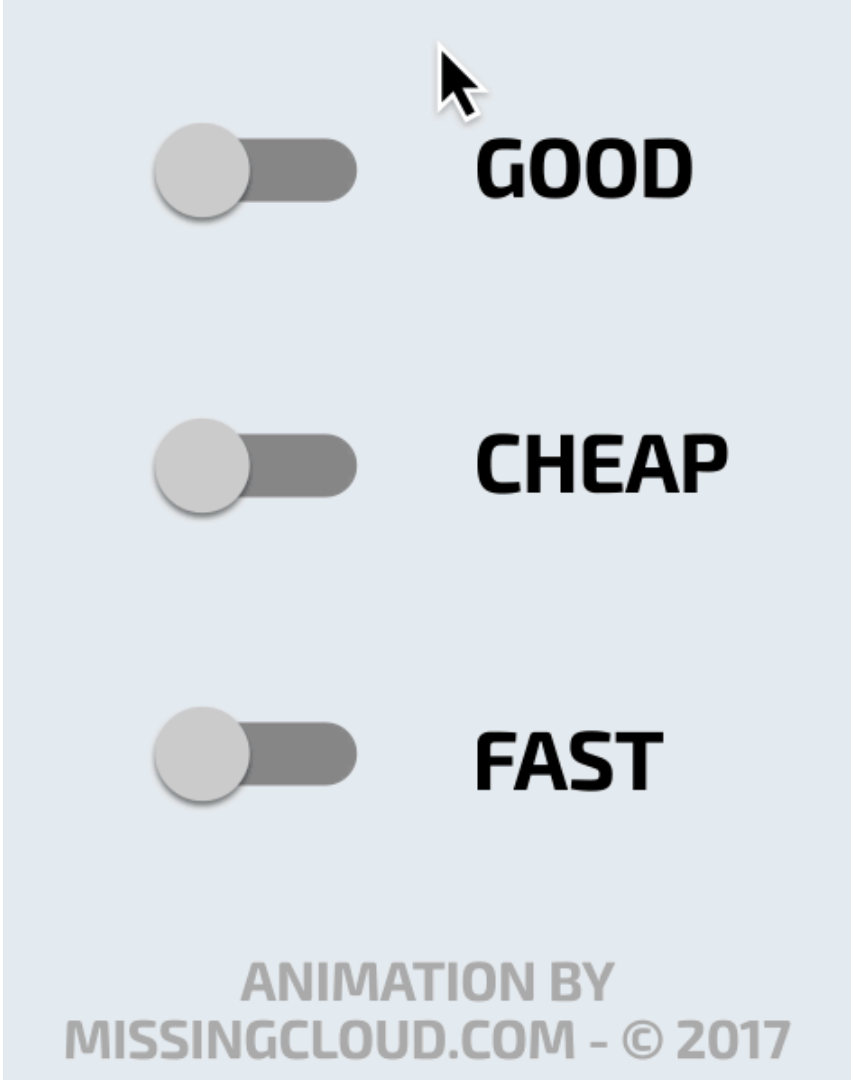


```
using System;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

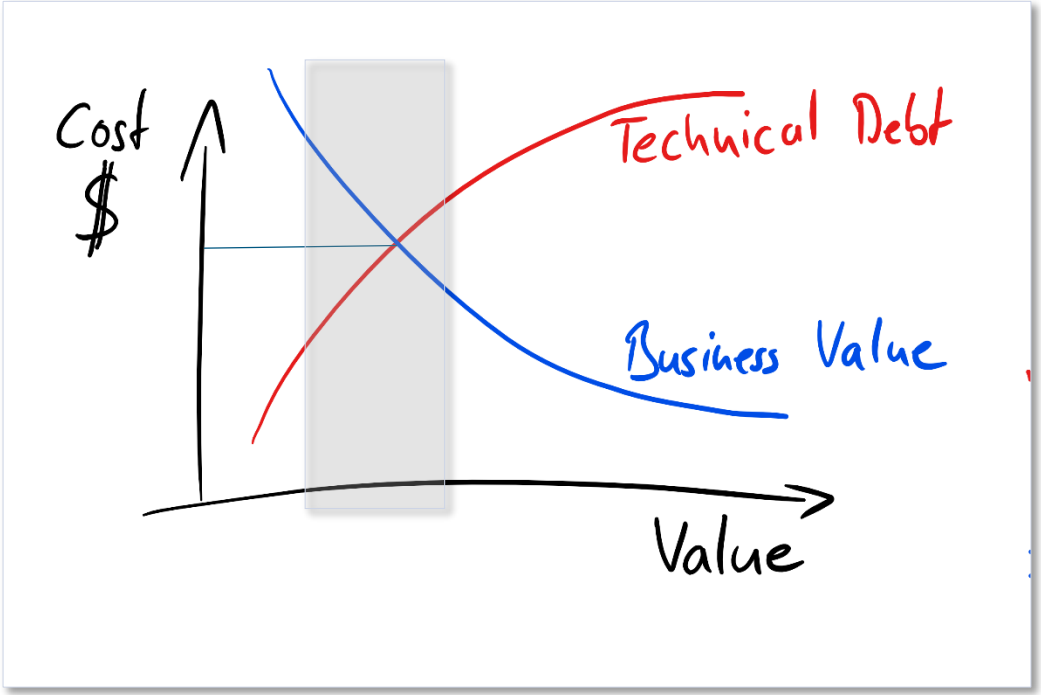
“Clean Code ist Teamsport!”

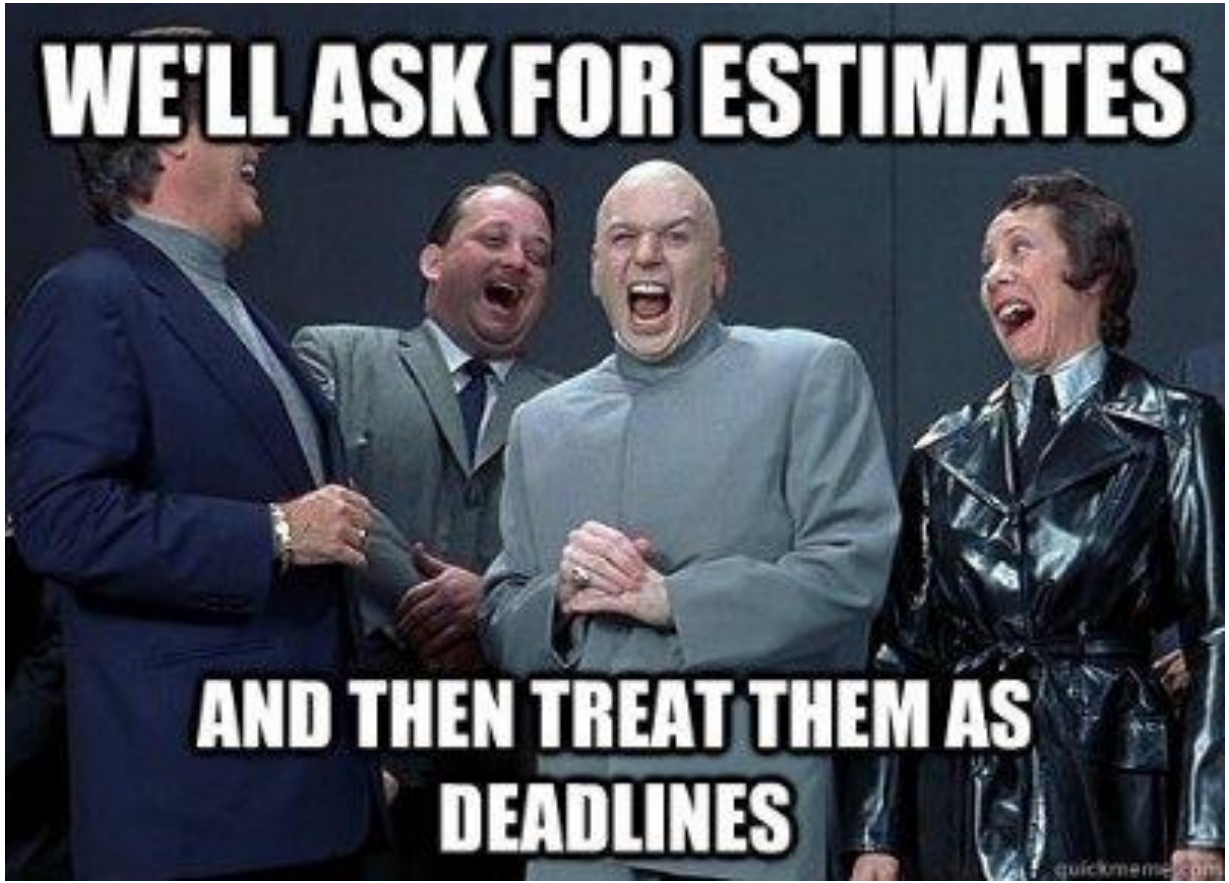
Was ist „Technical Debt“?




GOOD
CHEAP
FAST
ANIMATION BY
MISSINGCLOUD.COM - © 2017

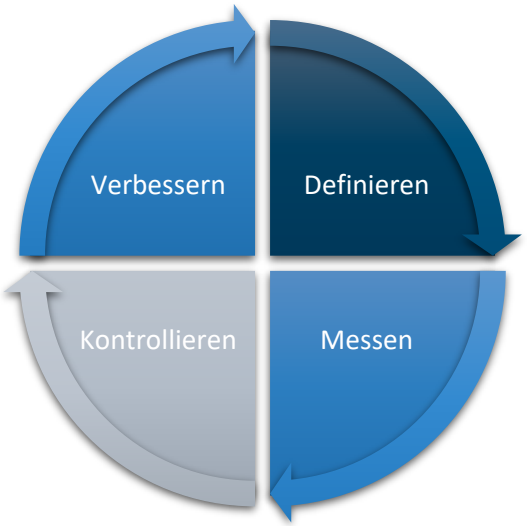
Umgang mit technischer Schuld




WE'LL ASK FOR ESTIMATES
AND THEN TREAT THEM AS DEADLINES

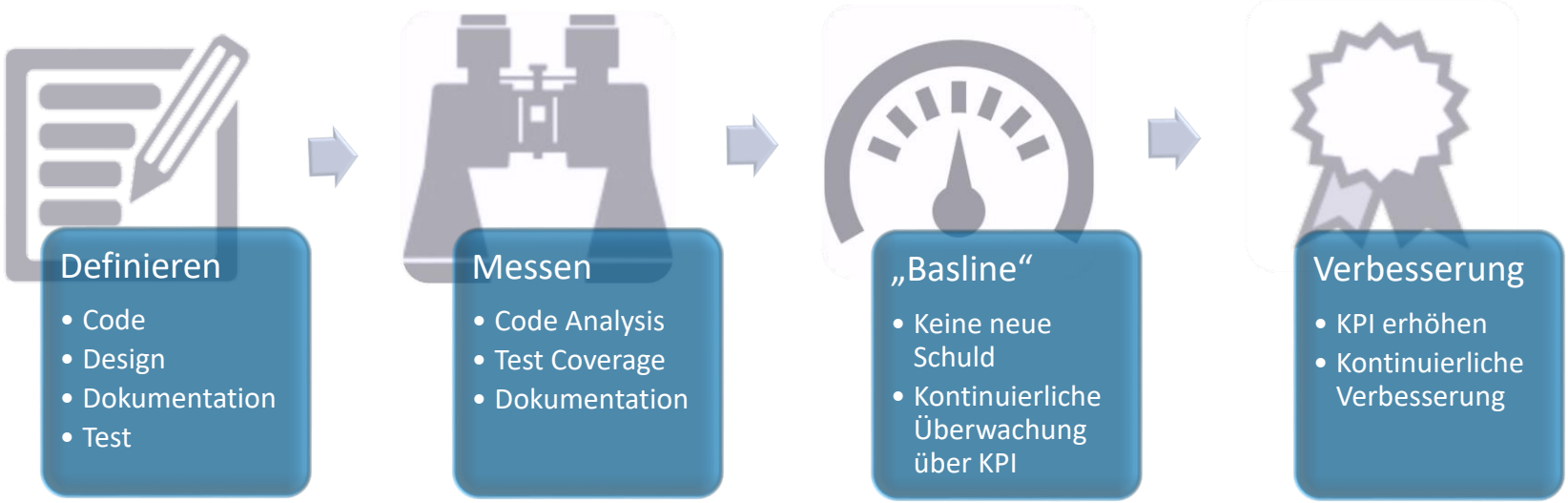
Umgang mit technischer Schuld

Management von Technical Debt

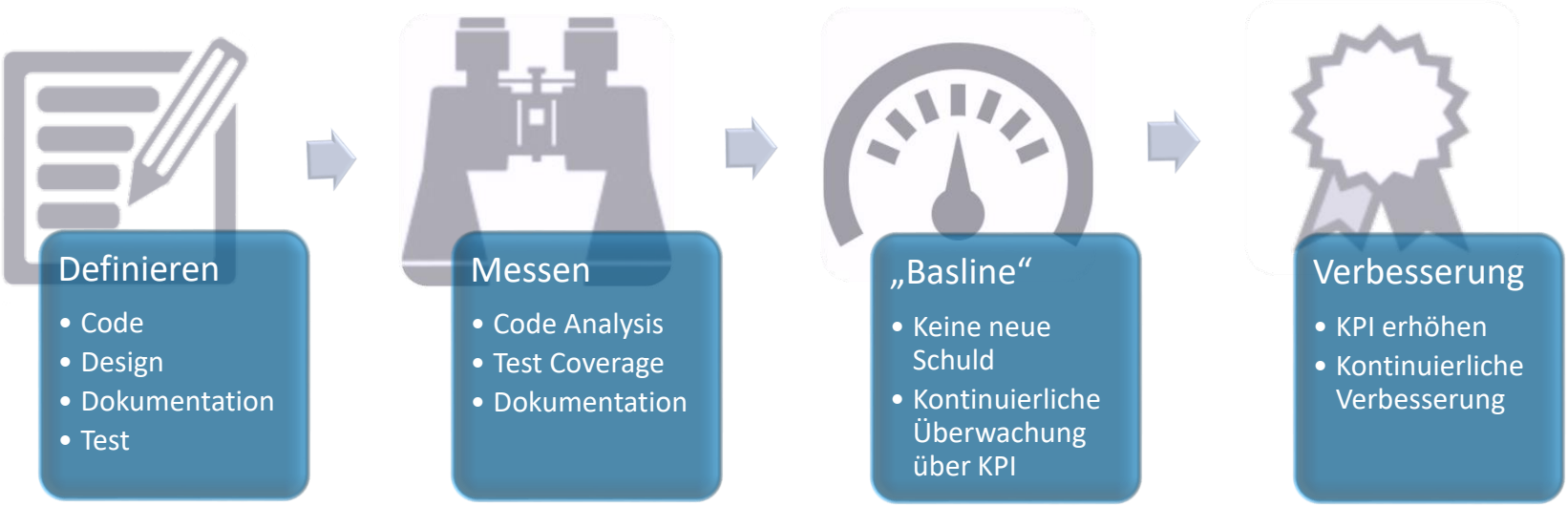


- Management Kreislauf
  - Definieren
  - Messen
  - Kontrollieren
  - Verbessern

Umgang mit technischer Schuld

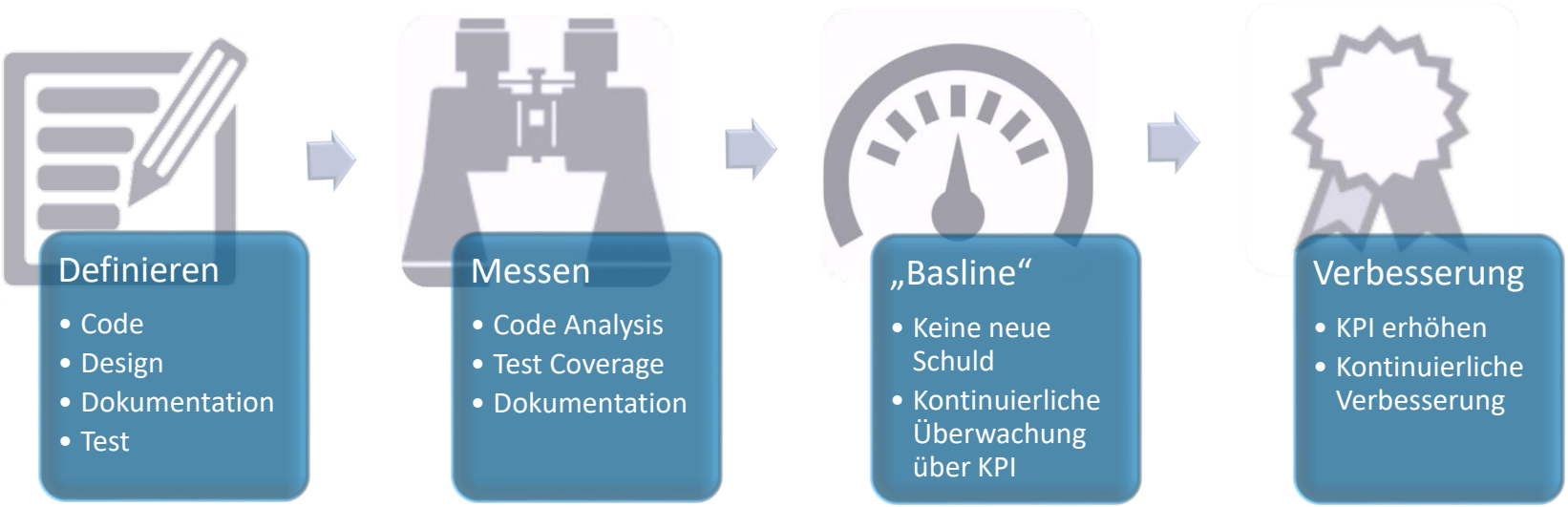


Umgang mit technischer Schuld



Code / Design: Continuous Integration / Delivery

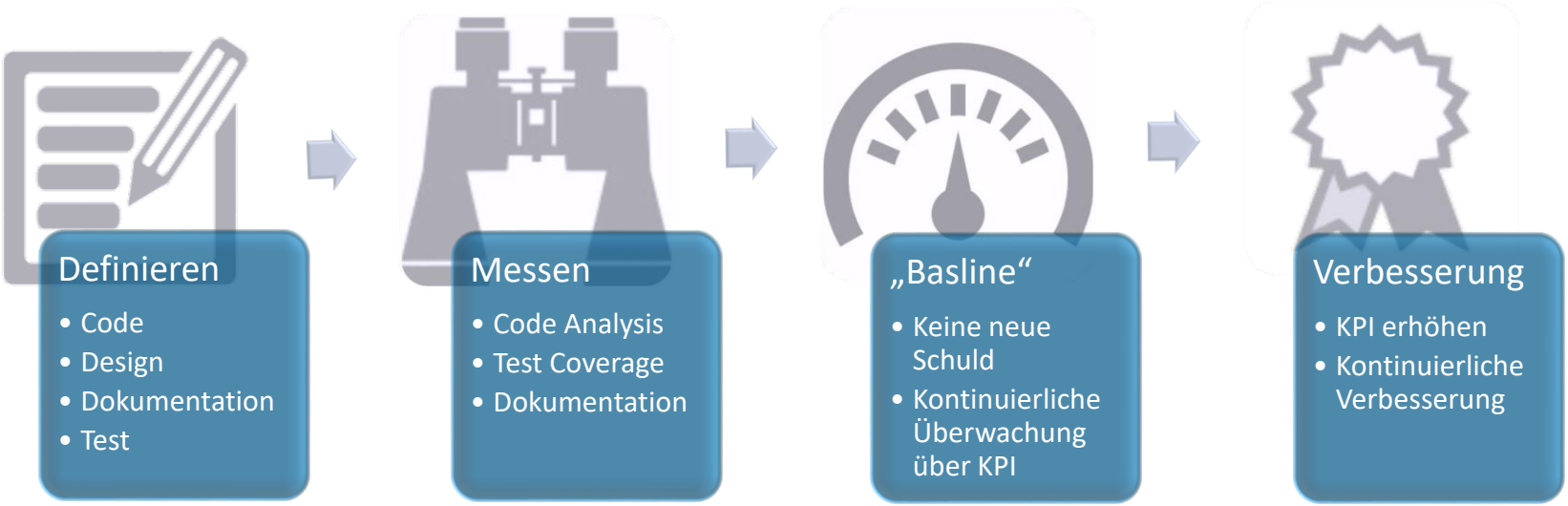
Umgang mit technischer Schuld



Code / Design: Metriken

- LoC
- Cyclomatic Complexity
- Depth of Inheritance
- Class Coupling

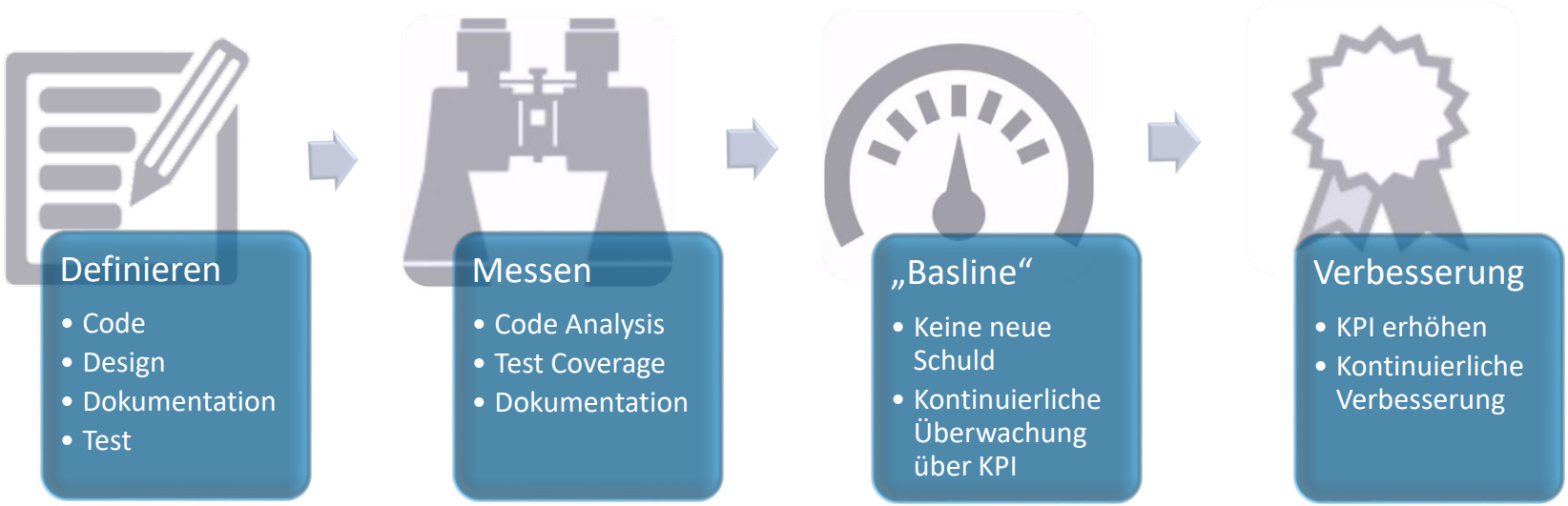
Umgang mit technischer Schuld



Code / Design: Statische Codeanalyse

- Roslyn
- Resharper
- SonarQube

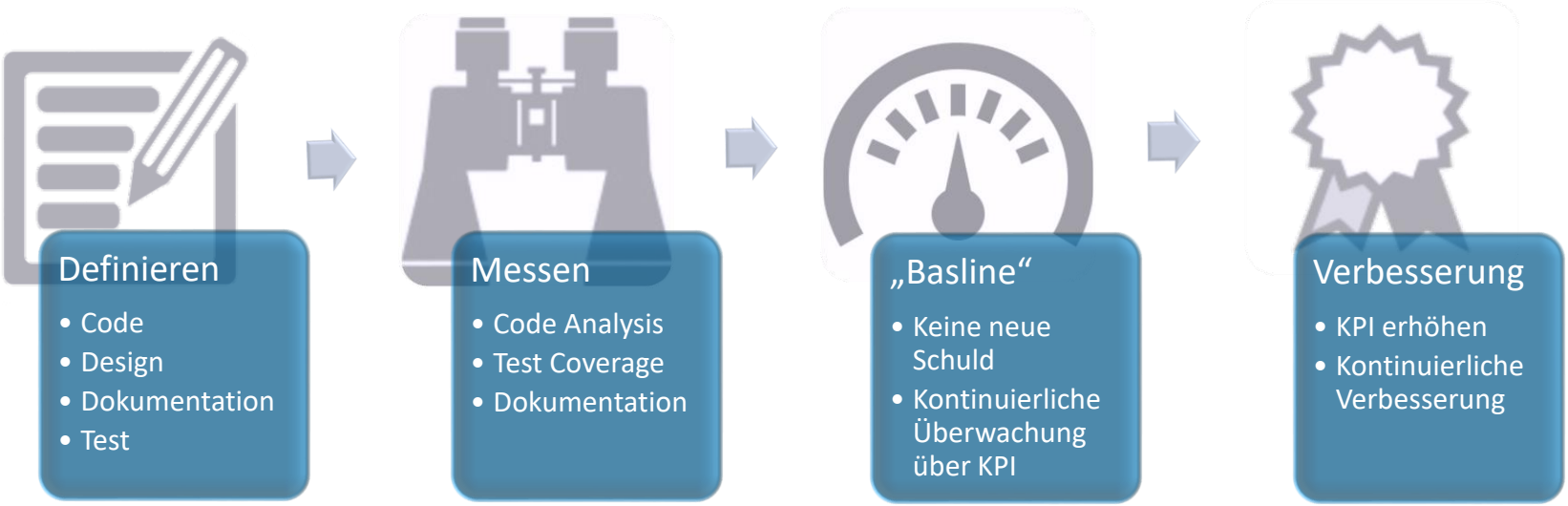
Umgang mit technischer Schuld



Dokumentation:

- So viel wie nötig, so wenig wie möglich
- Automatisierung

Umgang mit technischer Schuld



Test:Code Coverage

- Unit Tests
- Integration Tests
- System Tests

Demo

- https://sonarcloud.io/projects

Demo

- SonarLint

“Clean Code
ist die Abwesenheit von
technischer Schuld“

Michael Kaufmann