

Test-Driven C#

Rasmus Lystrøm
Associate Professor
ITU

The screenshot displays the Visual Studio Code interface with two open files: `ProgramTests.cs` and `Program.cs`. The `ProgramTests.cs` file contains a test class `ProgramTests` with a single test method `Main_given_no_args_p` using the `Fact` attribute and `Assert.Equal` to verify the output of `Program.Main`. The `Program.cs` file defines a `HelloWorld` namespace containing a `Program` class with a `Main` method that writes "Hello World!" to the console.

The bottom of the image shows the `TERMINAL` panel with the output of the command `dotnet test`. The output indicates that the test run was successful, with 1 test passed in 3.4618 seconds.

```
File Edit Selection View Go Debug Terminal Help
ProgramTests.cs x
HelloWorld.Tests > ProgramTests.cs > {} HelloWorld.Tests > HelloWorld
1 using System;
2 using System.IO;
3 using Xunit;
4
5 namespace HelloWorld.Tests
6 {
7     0 references | Run All Tests | Debug All Tests
8     public class ProgramTests
9     {
10         [Fact]
11         0 references | Run Test | Debug Test
12         public void Main_given_no_args_p
13         {
14             // Arrange
15             using var writer = new StringWriter();
16             Console.SetOut(writer);
17
18             // Act
19             Program.Main(new string[0]);
20
21             // Assert
22             var output = writer.GetStringBuilder().ToString();
23             Assert.Equal("Hello World!", output);
24         }
25     }
26 }
Program.cs x
HelloWorld > Program.cs > ...
1 using System;
2
3 namespace HelloWorld
4 {
5     1 reference
6     public class Program
7     {
8         1 reference
9         public static void Main(string[] args)
10         {
11             Console.WriteLine("Hello World!");
12         }
13 }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: pwsh
Loading personal and system profiles took 1008ms.
C:\HelloWorld> dotnet test
Test run for C:\HelloWorld\HelloWorld.Tests\bin\Debug\netcoreapp3.0\HelloWorld.Tests.dll (.NETCoreApp,Version=v3.0)
Microsoft (R) Test Execution Command Line Tool Version 16.3.0
Copyright (c) Microsoft Corporation. All rights reserved.

Starting test execution, please wait...

A total of 1 test files matched the specified pattern.

Test Run Successful.
Total tests: 1
Passed: 1
Total time: 3,4618 Seconds
C:\HelloWorld>
```

2013-: Principal Consultant @ Microsoft
DevOps, Cloud, Security

2014-: Associate Professor @ ITU
Object-Oriented Programming, C#, F#, .NET Core

M.Sc. IT, ITU (2012)
Thesis: Forecalc – Developing a core spreadsheet
implementation in F#

1996-2008, 2013-: Captain @ Danish Army (Reserve)
Battalion Chief of Staff,

Wife: Katrine
Children: Lærke (3), Laura (6), and Alma (13)

Origin: Aarhus
Current whereabouts: Vanløse, Copenhagen



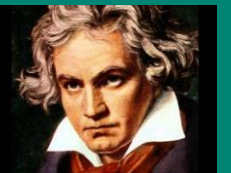
Hobbies

SLAYER



BRUTAL
ASSAULT

COPE~~N~~HELL



HELLEST

Agenda

- How to reach me
- *Boring stuff*
- Test-Driven Development
- Trunk Based Development
- Continuous Integration
- Show and tell

How to reach me

Do not write me an email!

Use Discord – the TA's or your fellow students might have the answer.

You may tag me.

You may DM me for personal questions.

Curriculum

Test-Driven C#

Generics

Lambdas and Linq

Data access (ORM)

Asynchronous and parallel

Web APIs

Design Patterns

Mobile and desktop apps

Web apps

Security

Cloud

Updated on LearnIT

“C# is intended to be a simple, modern, general-purpose, object-oriented programming language.”

ECMA-334 ISO/IEC 23270:2018(E)

C# language specification, 5th edition, December 2017

Why C#



Popular - [Stack Overflow Developer Survey 2021](#)



Ubiquitous



Open Source



Cross-platform



Industry / Enterprise



Tool for your toolbox



IRON



Why not C#?

It's just like Java:

Curly brackets and semicolons from C

Statically typed

Object-oriented

Single inheritance

Cross platform

Open source

Industry / enterprise

Enterprise

<https://github.com/joaomilho/Enterprise>

<https://github.com/EnterpriseQualityCoding/FizzBuzzEnterpriseEdition>



Build on what you know already

No basic stuff

Read the book

Do the exercises

Learn the advanced patterns and syntactic sugar

Idiomatic C[#]

Focus on clean, testable code

Stop the enterprise madness!



.NET – a brief introduction

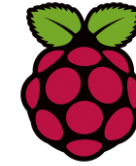
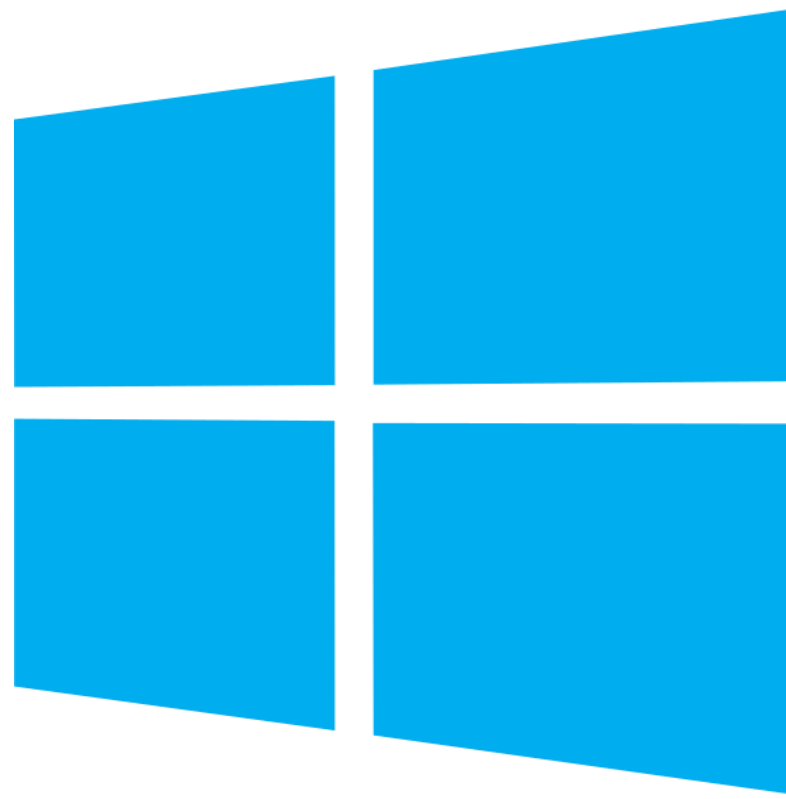
.NET

.NET is a free, cross-platform, open-source developer platform for building many different types of applications.

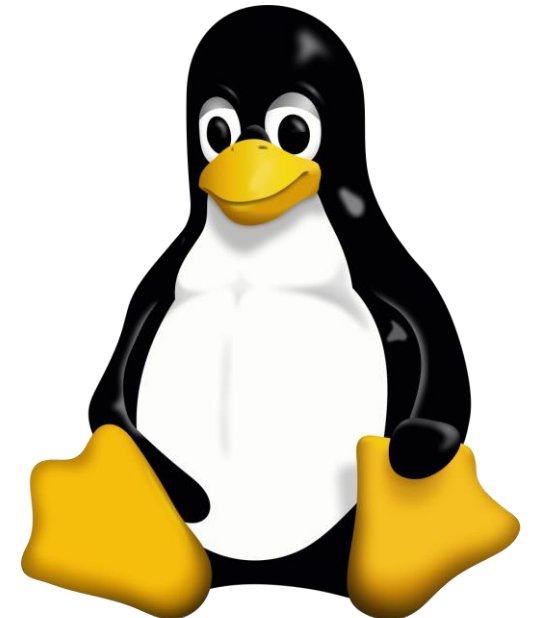
.NET Languages



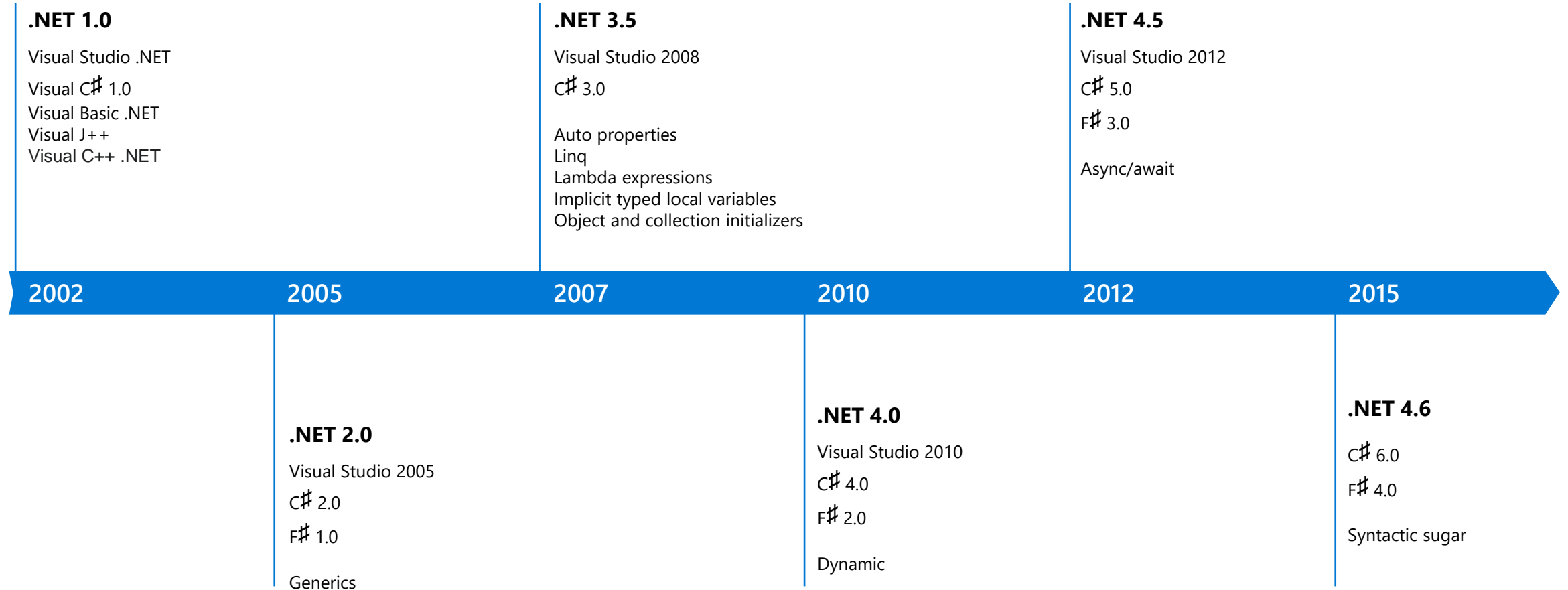
.NET Platforms



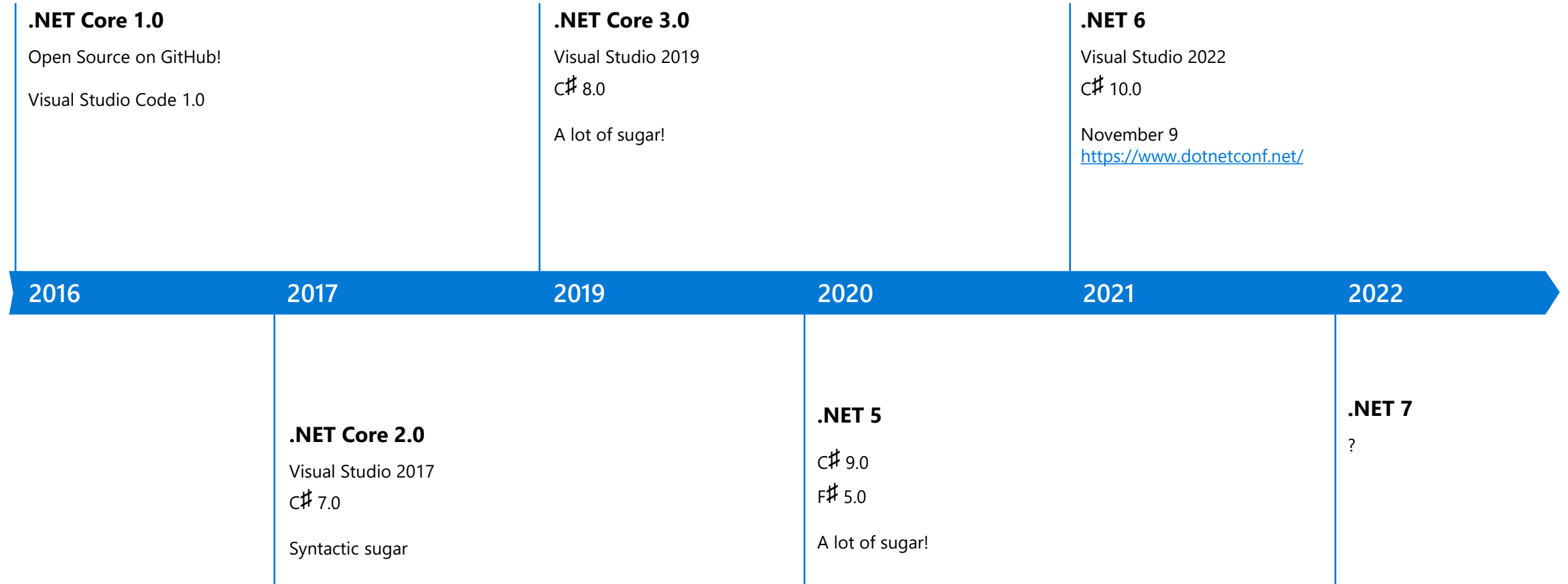
android



.NET Timeline



.NET Timeline II





Test-Driven Development

Test-Driven Development – What? – Why? – How?



WHAT?

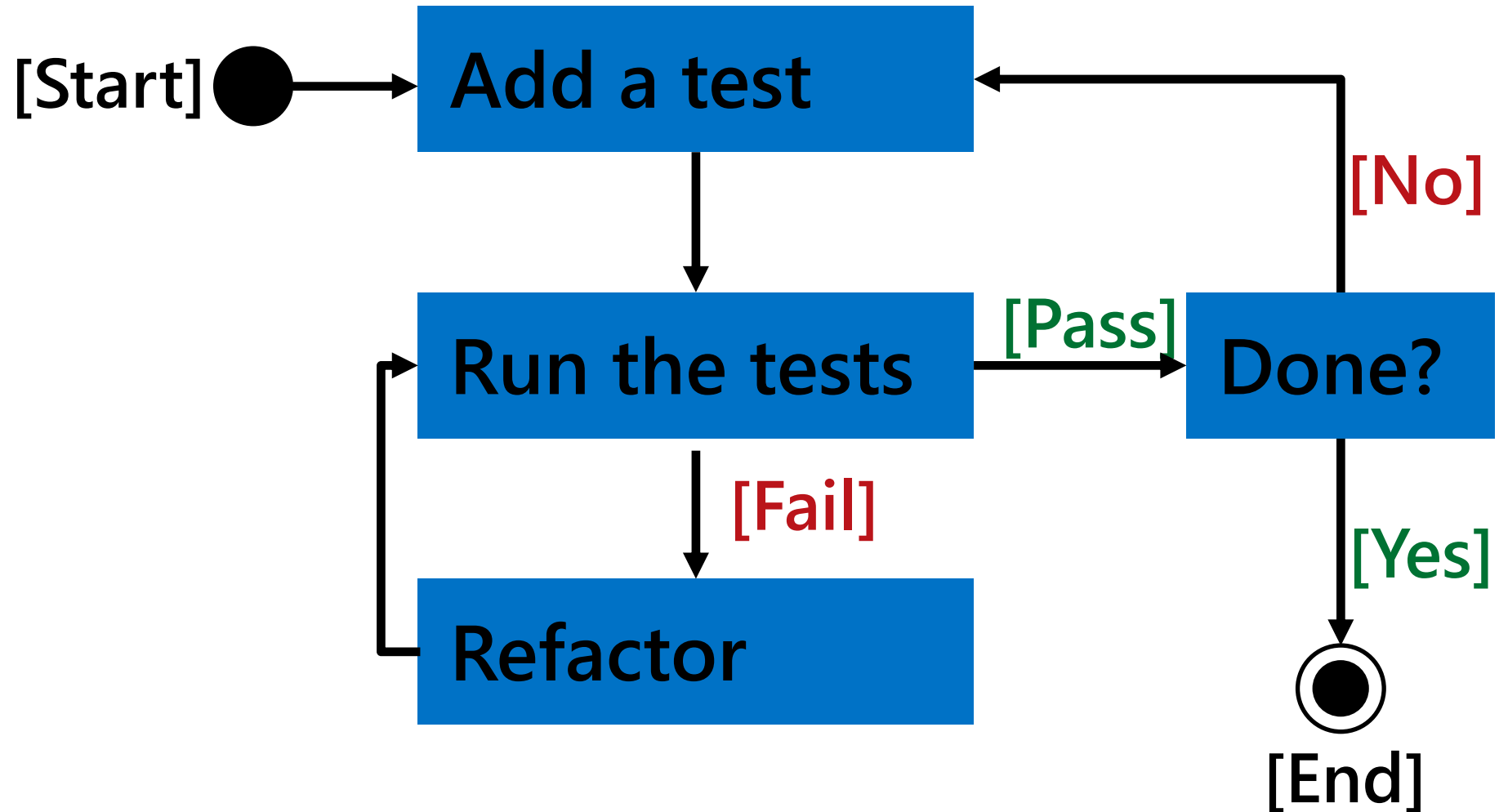


WHY?



HOW?

Red – Green - Refactor



The image is a close-up photograph of a tree trunk's cross-section. It features prominent concentric growth rings in shades of light brown and tan. A small, dark, irregular hole or knot is visible in the center-left area. The text "Trunk Based Development" is overlaid on the right side in a white, sans-serif font.

Trunk Based Development

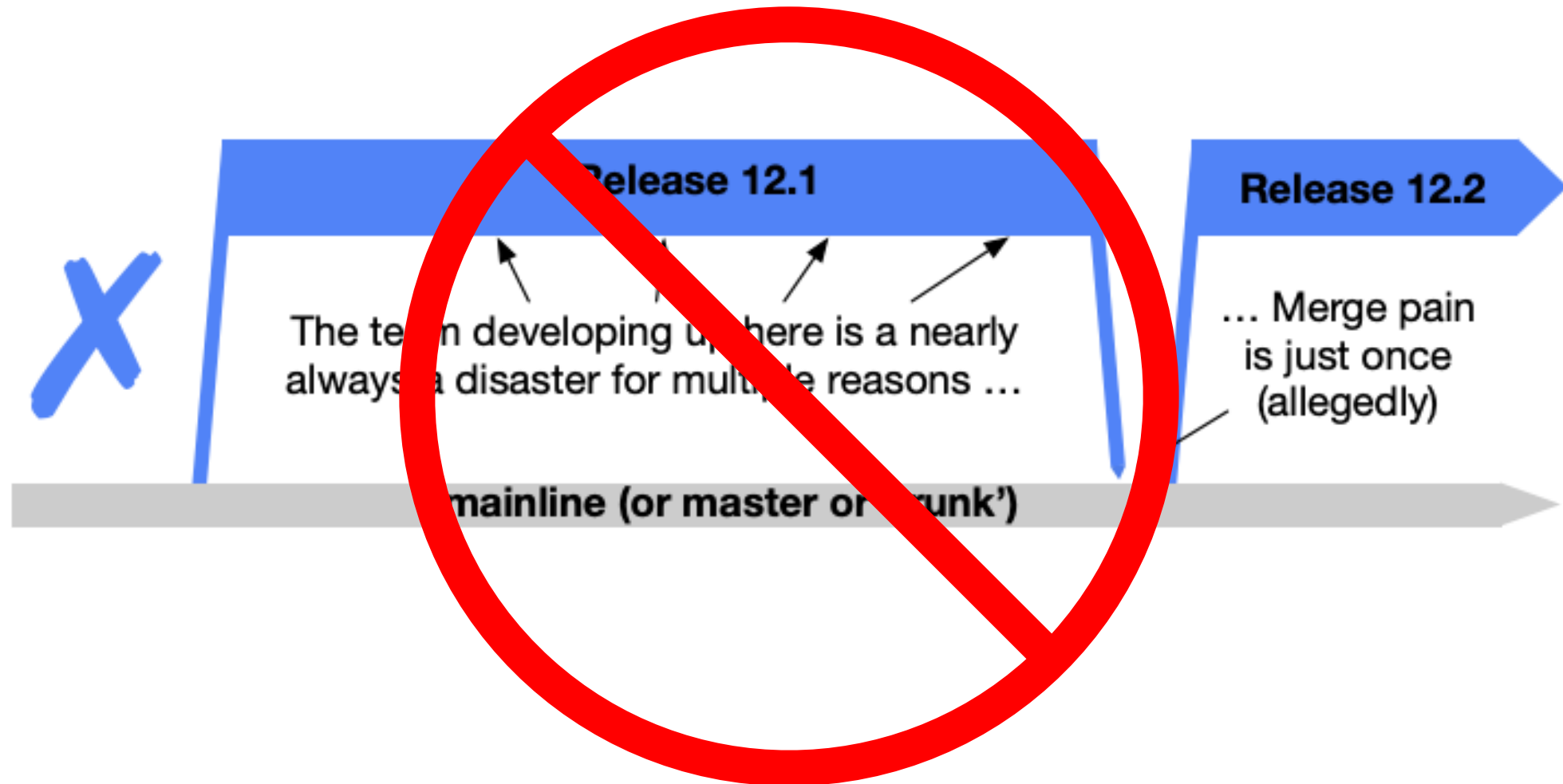
Trunk Based Development – What?

A source-control branching model, where developers collaborate on code in a single branch called 'trunk' *, resist any pressure to create other long-lived development branches by employing documented techniques. They therefore avoid merge hell, do not break the build, and live happily ever after.

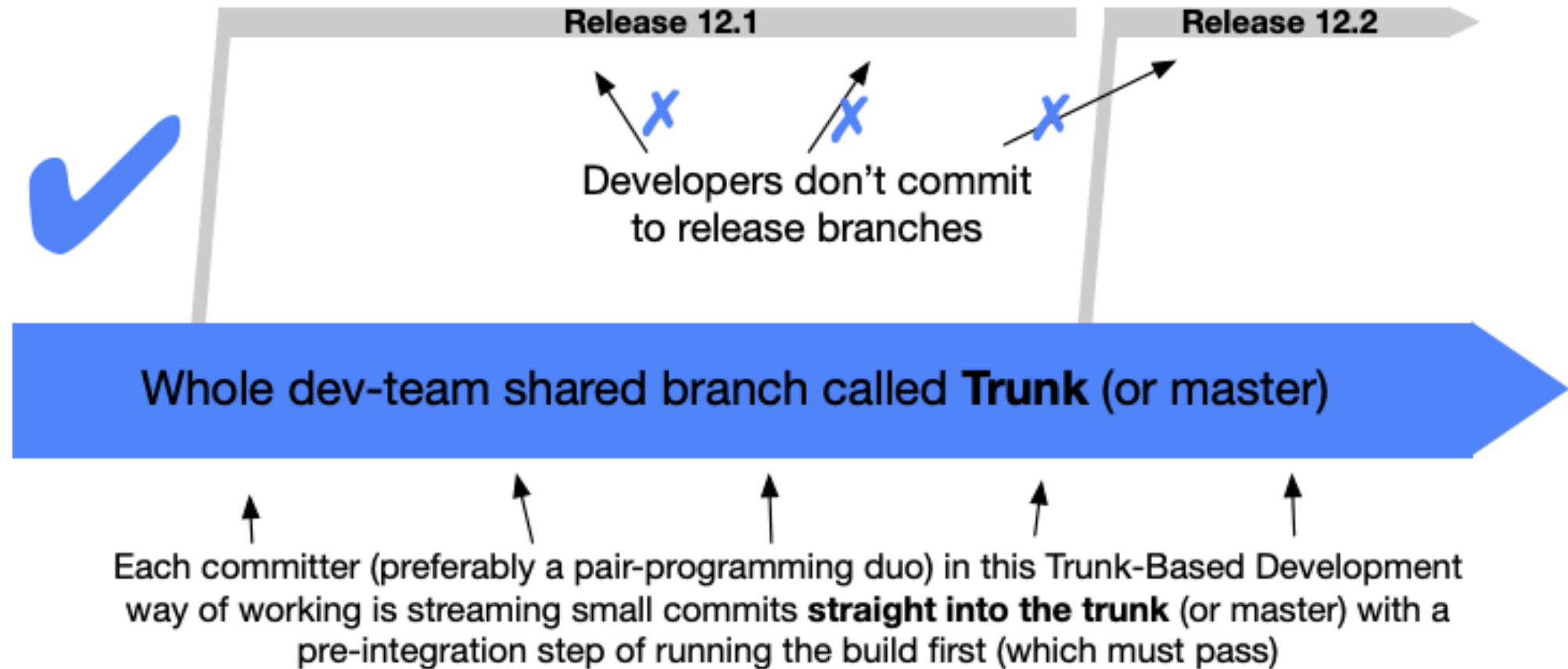
master/main, in Git nomenclature

Source: <https://trunkbaseddevelopment.com/>

Shared branches off mainline/master/trunk are bad at any release cadence



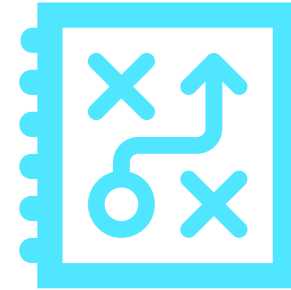
Trunk Based Development – How?



Branching strategy vs. tactics

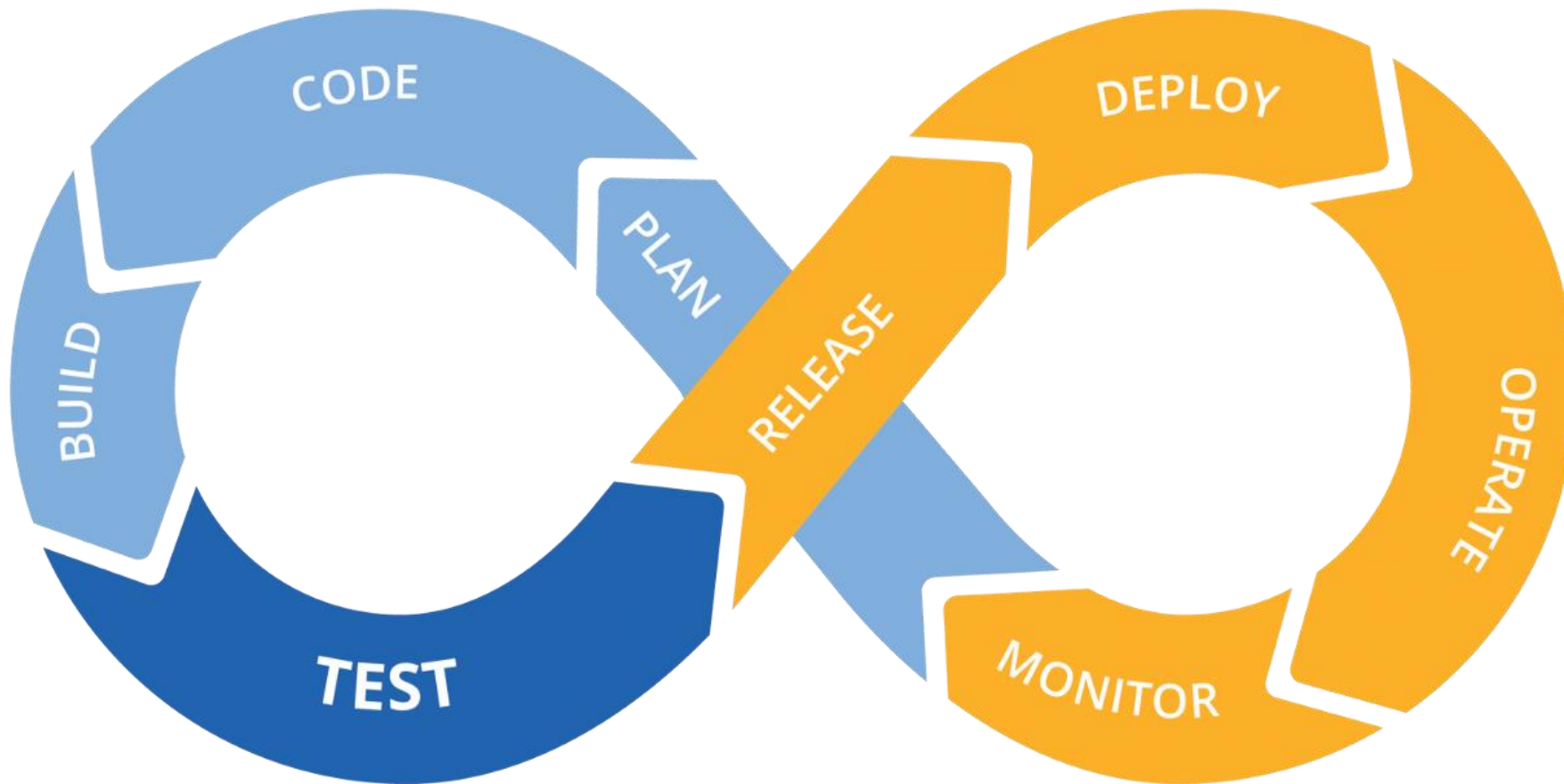


Short-lived < 1 day



No *strategy*!

Continuous Integration



This Photo by Unknown Author is licensed under [CC BY-SA-NC](#)

Continuous Integration – What?

Continuous Integration (CI) is the process of automating the build and testing of code every time a team member commits changes to version control.

Source: <https://docs.microsoft.com/en-us/devops/develop/what-is-continuous-integration>



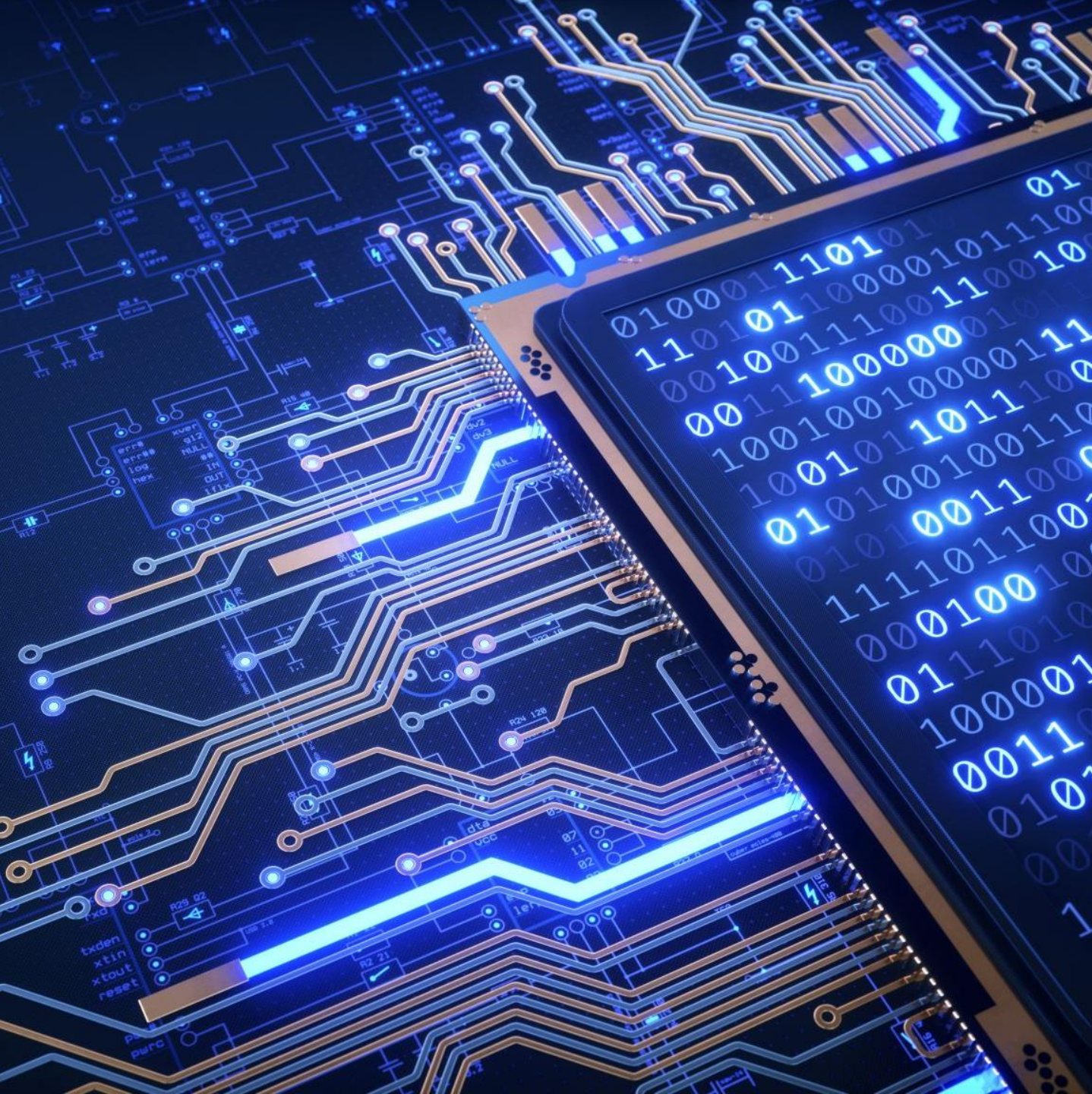
Demo

Test-Driven C[#] using Continuous Integration and Trunk Based Development



Thank you

Appendix



Create a C# console app with a test library

```
mkdir MyApp
```

```
cd MyApp
```

```
dotnet new console -o MyApp
```

```
dotnet new xunit -o MyApp.Tests
```

```
dotnet new sln
```

```
dotnet sln add MyApp
```

```
dotnet sln add MyApp.Tests
```

```
dotnet add MyApp.Tests
```

```
reference MyApp
```



Build, test, and run your app

```
dotnet build
```

```
dotnet test
```

```
dotnet run --project MyApp
```

Naming conventions

Composed names

`currentLayout, CurrentLayout`

Variables and fields

`vehicle, leftElement`

Private fields

`_vehicle, _leftElement`

Methods

`CurrentVehicle(), Size()`

Properties

`Pi, Name, Size`

Classes

`MyClass, List<T>`

Interfaces

`IOException, IObservable`

The C# class

```
using System;

namespace Namespace
{
    public class Class
    {
        private string _field;

        protected DateTime _inheritableField;

        public string Property { get => _field; } // Getter

        public int AutoProperty { get; set; }

        public Class() // Constructor
        {
        }
    }
}
```

The C# class (methods)

```
public object InstanceMethod(int parameter)
{
    return null;
}
```

```
public virtual object OverridableInstanceMethod(bool parameter)
{
    return null;
}
```

```
public static void StaticMethod()
{
}
```

```
private void PrivateInstanceMethod()
{
}
```

The C# class (events and delegates)

```
public event EventHandler Event;

protected virtual void OnEvent(EventArgs e)
{
    EventHandler handler = Event;
    handler?.Invoke(this, e);
}

}

public delegate void MyEventHandler(object sender, EventArgs e);
}
```

Built-in types

```
bool boolean; // true || false
char character; // 'a', 'b', 'c', '1', '2', '3'
```

```
// Floating point numeric types
decimal precisionFloatingPoint;
double floatingPoint64bit;
float floatingPoint32Bit;
```

```
// Integral numeric types
```

```
byte integer8bit;
int integer32bit;
long integer64bit;
short integer16bit;
```

```
sbyte signedByte;
uint unsignedInteger32bit;
ulong unsignedInteger64bit;
ushort unsignedInteger16bit;
```

```
// Reference types
```

```
object _object;
string _string;
dynamic dynamic;
```

Basic Unit Test

```
public class Ticker
{
    public int Counter { get; private set; }
    public void Increment() => Counter++;
}

public class TickerTests
{
    [Fact]
    public void Increment_when_called_increases_Counter_by_1()
    {
        // Arrange
        var sut = new Ticker();

        // Act
        sut.Increment();

        // Assert
        Assert.Equal(1, sut.Counter);
    }
}
```