# C#
# Manual Of Style

**Dino Esposito**

CTO Crionet Sports
dino.esposito@crionet.com

NEXT →

# Coding Practices

Style SOLID Techniques

## Backed by

Some common-sense

Some knowledge and pragmatism

Lots of personal idiosyncrasies
**(of a software person since 1992)**

# Style

Writing

Comments

Naming

## DISCLAIMER

Everything is highly subjective. But someone has to say it.
(And better if it’s an old-school guy ☺)

NEXT →

## Writing Rules

- **Accept suggestions from code assistant tools (i.e., R#)**
- One statement and declaration per line
  - Preferably render long LINQ statements fluently
- Add **one blank** line space between (related groups of) methods
- Be **standard** with indentation (tabs) and nesting
- XML comments to describe **at least** public elements of a class
- Be careful/wise with **#regions** (but don’t exclude their use)

## Comments

- **Open your heart ☺**
  - Software is full of silly things done for largely acceptable reasons
  - Make sure you explain weird choices and last-minute changes
  - If the code does things not completely intelligible, report your thoughts at the moment
- **Expect readers with some domain context**
  - But not too much (depending on members of the team and turnover)
  - Newbies are not the target of comments
  - Should you describe processes?
- **Be precise and concise (regardless)**

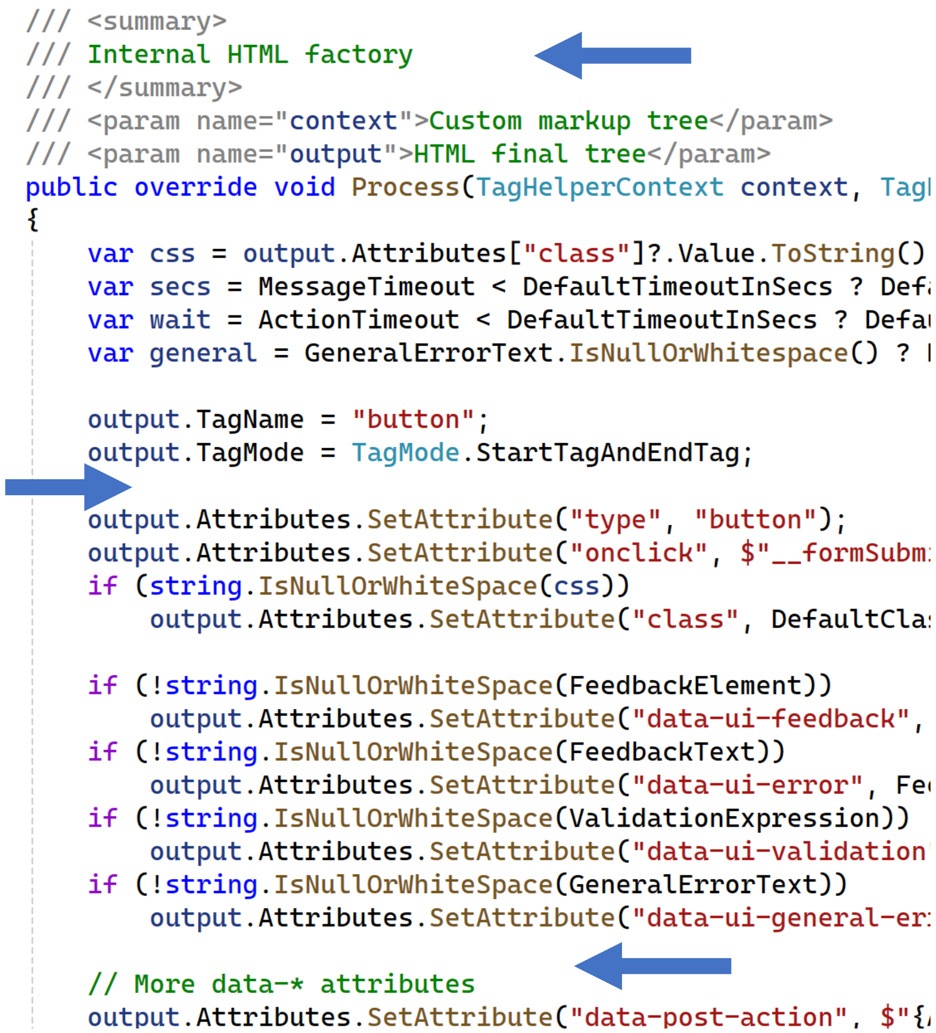
## Naming

- **C# is PascalCase**
  - Acronyms upper case unless followed by text: `ẄÔ`, `RsîŋʧẄÔ`, `RsîŋʧẄộĽîșʧ`
- **English only** **(NOTE: this could just be me)**
- **Ubiquitous Language rules from DDD**
- **Get a convention and be consistent**
  - Force team members to do the same

NEXT →

## Even minor things count

Painful if not done consistently across the repo and commits



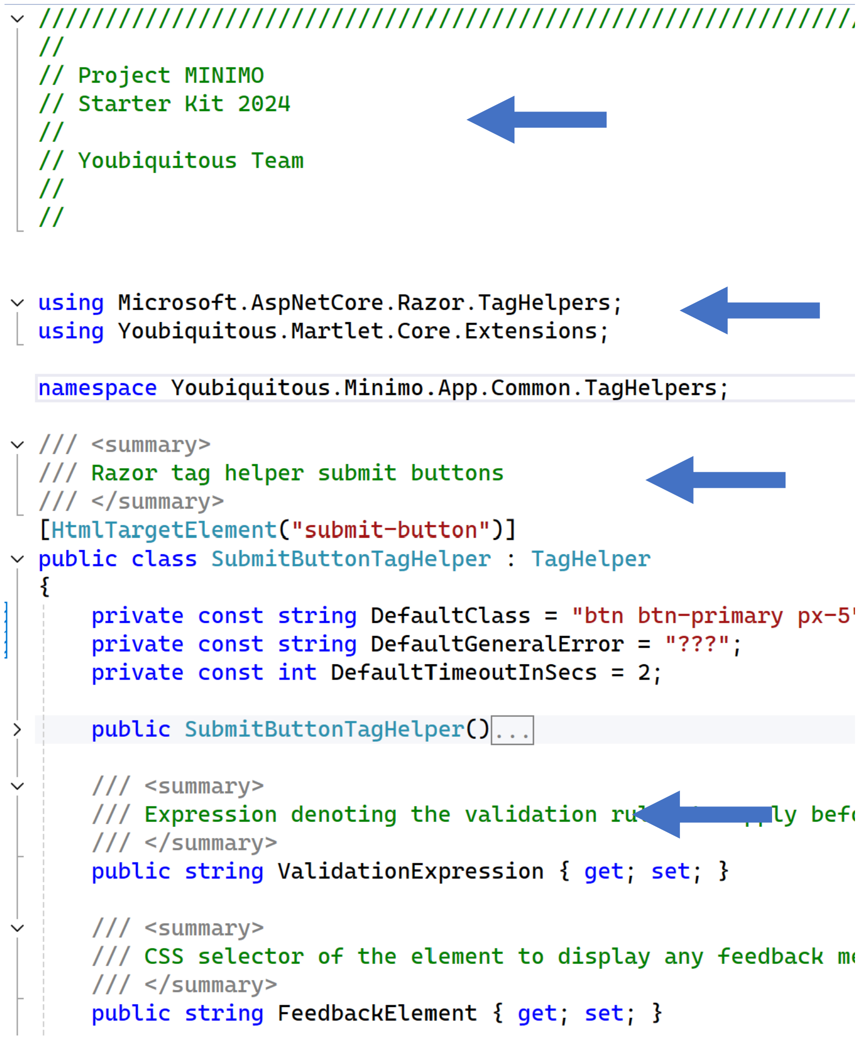
```
/// <summary>
/// Internal HTML factory
/// </summary>
/// <param name="context">Custom markup tree</param>
/// <param name="output">HTML final tree</param>
public override void Process(TagHelperContext context, Tag
{
    var css = output.Attributes["class"]?.Value.ToString()
    var secs = MessageTimeout < DefaultTimeoutInSecs ? Def
    var wait = ActionTimeout < DefaultTimeoutInSecs ? Defa
    var general = GeneralErrorText.IsNullOrWhitespace() ? 

    output.TagName = "button";
    output.TagMode = TagMode.StartTagAndEndTag;

    output.Attributes.SetAttribute("type", "button");
    output.Attributes.SetAttribute("onclick", $"__formSubm
    if (string.IsNullOrWhiteSpace(css))
        output.Attributes.SetAttribute("class", DefaultCla

    if (!string.IsNullOrWhiteSpace(FeedbackElement))
        output.Attributes.SetAttribute("data-ui-feedback",
    if (!string.IsNullOrWhiteSpace(FeedbackText))
        output.Attributes.SetAttribute("data-ui-error", Fe
    if (!string.IsNullOrWhiteSpace(ValidationExpression))
        output.Attributes.SetAttribute("data-ui-validation
    if (!string.IsNullOrWhiteSpace(GeneralErrorText))
        output.Attributes.SetAttribute("data-ui-general-er

    // More data-* attributes
    output.Attributes.SetAttribute("data-post-action", $"{
```



```
////////////////////////////////////////////////////////////////
//
// Project MINIMO
// Starter Kit 2024
//
// Youbiquitous Team
//
//


using Microsoft.AspNetCore.Razor.TagHelpers;
using Youbiquitous.Martlet.Core.Extensions;

namespace Youbiquitous.Minimo.App.Common.TagHelpers;

/// <summary>
/// Razor tag helper submit buttons
/// </summary>
[HtmlTargetElement("submit-button")]
public class SubmitButtonTagHelper : TagHelper
{
    private const string DefaultClass = "btn btn-primary px-5
    private const string DefaultGeneralError = "???";
    private const int DefaultTimeoutInSecs = 2;

    public SubmitButtonTagHelper()...

    /// <summary>
    /// Expression denoting the validation ru      ly bef
    /// </summary>
    public string ValidationExpression { get; set; }

    /// <summary>
    /// CSS selector of the element to display any feedback m
    /// </summary>
    public string FeedbackElement { get; set; }
```

# SOLID

Writing

Comments

Naming

NEXT →

## SOLID at a (pragmatic) glance

### SRP
Do just one thing—the boundary of which is up to you and your expertise/sensitivity

### OCP
Think the class to be extensible, via generics or behavior providers (Strategy pattern)

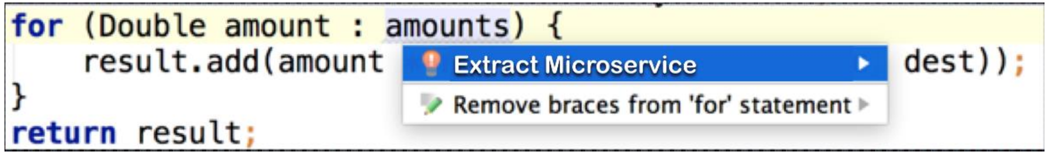
### LSP
Use inheritance widely? Then, every derived class should be usable wherever base class is accepted

### ISP
Use a lot of abstractions? Then, no client should be forced to implement an interface it doesn't use

### DIP
Code to an interface rather than to an implementation

## **SOLID** at a (**even more** pragmatic) glance

- **Remember the mantra** "Every class as a service"**?**
  - A provocative statement…



```
for (Double amount : amounts) {
    result.add(amount                                    dest));
}
return result;
```

- **Patterns won't save the world**
  - Tool, but not a magic wand
- **Abstractions and DI only if you need to replace pieces of behavior**
  - Over-engineering is a risk (IF statements still supported)
  - In the end, you use implementations not interfaces

NEXT →

## SOLID Like Personal Hygiene

- **Health/Technical Debt analogy**
  - Personal hygiene prevents health issues, while SOLID prevents code rot and technical debt
- **Both require regular, disciplined practices**
  - Consistent application over time to maintain good habits
- **Neglect leads to long-term problems**
  - Poor hygiene causes infections; poor SOLID leads to fragile and unmanageable codebases
- **Not always immediately obvious**
  - Effects of good hygiene and clean code aren't always instantly visible, but their benefits accumulate over time

# C# TECHNIQUES

Partial Classes

Extensions

Sugar

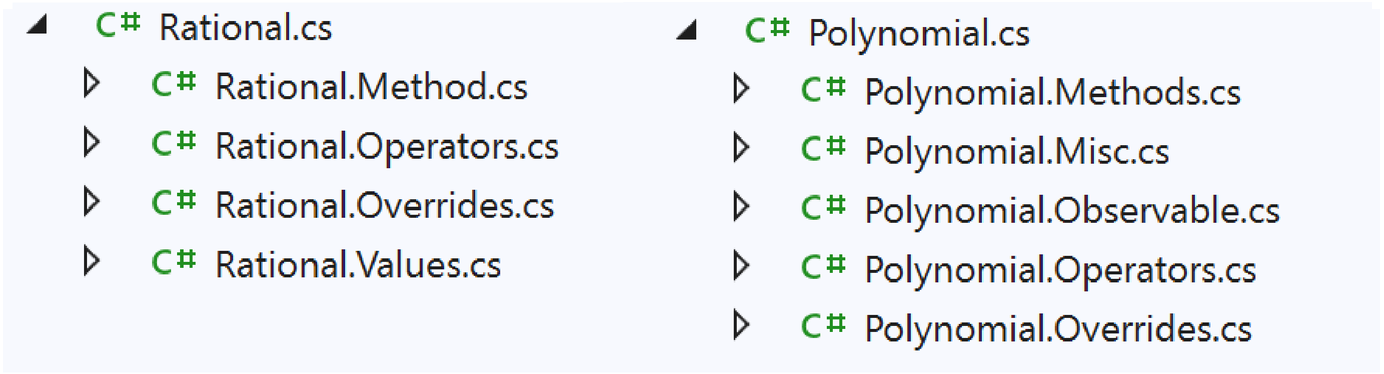
NEXT →

1

PARTIAL CLASSES

More coming in C#13

## **Single class** definition but split across multiple files


Rational.cs
Rational.Method.cs
Rational.Operators.cs
Rational.Overrides.cs
Rational.Values.cs

Polynomial.cs
Polynomial.Methods.cs
Polynomial.Misc.cs
Polynomial.Observable.cs
Polynomial.Operators.cs
Polynomial.Overrides.cs

✓ Different aspects of a class, such as data members, methods, or event handlers, can be placed in separate files, making it easier to manage and maintain code.

✓ Multiple developers can work on different parts of a class simultaneously without conflicting with each other, promoting parallel development.

✓ Enhance code readability by allowing developers to focus on specific sections of a class at a time, making it easier to understand and navigate the codebase.

Partial methods: definition in one file, implementation in another

NEXT →

2

MORE COMPACT CODE

## ✓ Early Return

```
public string BuildReport(int year)
{
    if (year < 2021)
        return null;

    // Proceed
    // ...
```

**Preconditions here**

## ✓ IF pollution

- ❑ Invert the Boolean condition
- ❑ Use switch construct
- ❑ Merge multiple IF statements

## ✓ Pattern Matching

```
if (doc != null &&
    doc.YearOfRelease >= 2015 &&
    doc.YearOfRelease < 2023 &&
    doc.YearOfRelease != 2020)
{
    // Do some work
}
```

```
if (doc is
      { YearOfRelease: >=  2015 and
                       <   2023 and
                       not 2020 })
{
    // Do some work
}
```

NEXT →

3

METHODS
OVER PROPERTIES

```
user.PasswordResetToken = Guid.NewGuid();
user.PasswordResetRequested = DateTime.UtcNow;
```

The method includes just the two lines above that set properties. You simply shifted from a data-centric vision to a behavior-centric perspective.

```
user.RequestPasswordReset();
```

**Where is readability**? In the name of the action.

**Where is maintainability**? You can possibly change the way password reset is implemented by just rewriting the method.

4

MAGIC CONSTANTS

```
public class Surface
{
    private static readonly Surface[] _all = { Clay, Grass, Hard };
    private Surface(string name)
    {
        Name = name;
    }

    // Public readable name
    public string Name { get; private set; }

    // Enum values
    public static readonly Surface Clay = new("Clay");
    public static readonly Surface Grass = new("Grass");
    public static readonly Surface Hard = new("Hard");

    // Behavior
    public static Surface Parse(string name)
    { /* ... */ }

    public IEnumerable<Surface> All()
    {
        return _all;
    }
}
```

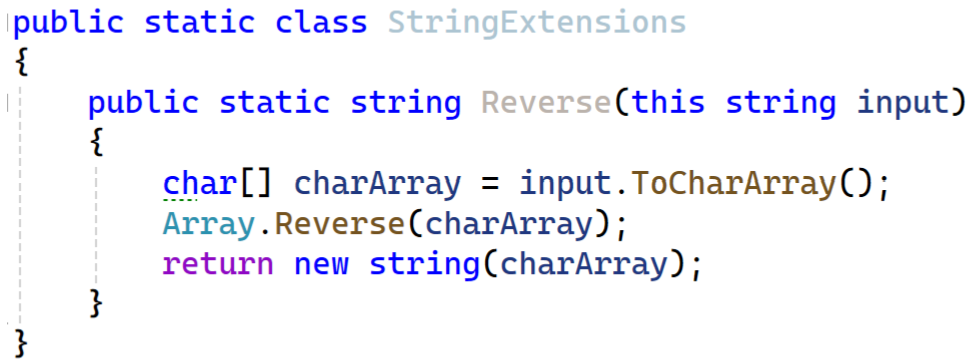
EXT →

5

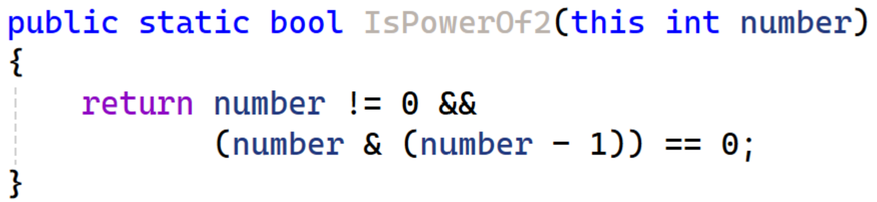
EXTENSION METHODS

New methods on existing types to extend the functionality without having to inherit or create wrapper classes.

**Make the code more convenient and readable**



```
public static class StringExtensions
{
    public static string Reverse(this string input)
    {
        char[] charArray = input.ToCharArray();
        Array.Reverse(charArray);
        return new string(charArray);
    }
}
```



```
public static bool IsPowerOf2(this int number)
{
    return number != 0 &&
           (number & (number - 1)) == 0;
}
```

NEXT →

# Extension Methods Extensions in C# 14

- **An extension type builds on an underlying type**
  - Normal C# types, yours or from external libraries
  - Might want use an extension if you can't change the code of the underlying type
- **Syntactic sugar**
  - Implemented as static methods that receive an instance as a parameter
  - Compiler accepts a "magic" syntax that make it look like a true method of the type
- **Two kinds of extension types: implicit and explicit extensions**
  - **Implicit** apply to all occurrences of the underlying type (same as today)
  - **Explicit** apply only to instances of the underlying type converted to the extension type
  - Explicit extension types may include methods **and properties**

C# 14

```
řụčľîç çľǻṣṣ Rêṣṣộŋ

    řụčľîç ṣʧsîŋĝ GîṣṣʧŅǻṇê   ĝêʧ  ṣêʧ
    řụčľîç ṣʧsîŋĝ ĽǻṣʧŅǻṇê   ĝêʧ  ṣêʧ
    řụčľîç DǻʧêŢîṇê Bîsʧḥ   ĝêʧ  ṣêʧ
```

```
řụčľîç îṇřľîçîʧ êỵʧêŋṣîộŋ RêṣṣộŋÉỵʧêŋṣîộŋ ǧộs Rêṣṣộŋ

      Éỵʧêŋṣîộŋ řsộřêsʧỳ
    řụčľîç îŋʧ Aĝê    DǻʧêŢîṇê ÛʧçŅộx Ÿêǻs   Bîsʧḥ Ÿêǻs
```

NEXT →

```
řụčľîç çľắṣṣ Rêṣṣộŋ

    řụčľîç ṣʧsîŋĝ GîsṣʧŅắṇê   ĝêʧ  ṣêʧ
    řụčľîç ṣʧsîŋĝ ĽắṣʧŅắṇê   ĝêʧ  ṣêʧ
    řụčľîç DắʧêŢîṇê Bîsʧḥ   ĝêʧ  ṣêʧ
```

Explicit extensions let you give **extra features** to specific instances of a type

```
řụčľîç êyřľîçîʧ êyʧêŋṣîộŋ RêṣṣộŋÉyʧêŋṣîộŋ ğộs Rêṣṣộŋ

      Éyʧêŋṣîộŋ řsộřêsʧỳ
   řụčľîç îŋʧ Aĝê    DắʧêŢîṇê ÛʧçŅộx Ÿêắs   Bîsʧḥ Ÿêắs


     Ûṣắĝê
 ŵắs řêṣṣộŋ   ŋêx Rêṣṣộŋ
 RêṣṣộŋÉyʧêŋṣîộŋ ṣřêçîắľ   řêṣṣộŋ      Aĝê ộŋľỳ ắŵắîľắčľê ʧộ person
 Cộŋṣộľê ŴsîʧêĽîŋê ṣřêçîắľ Aĝê
```

NEXT →

**It’s all** (or most) **about readability**

## What do I do for a living?

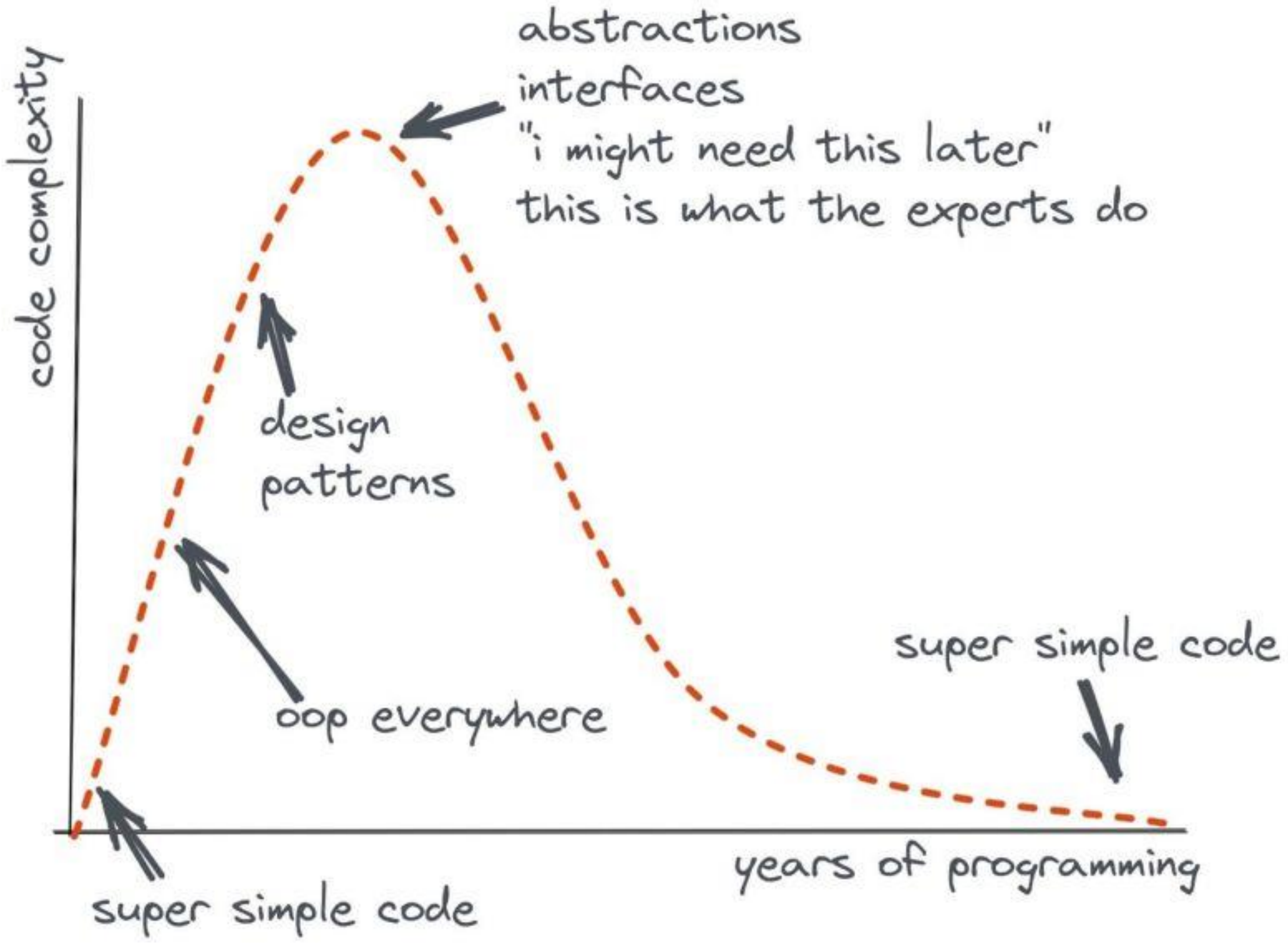
Ensure daily operations  across a few sport governing bodies

Ensure proper data/stats worldwide distribution

Ensure 24x7 proper betting

# Coding for Simplicity

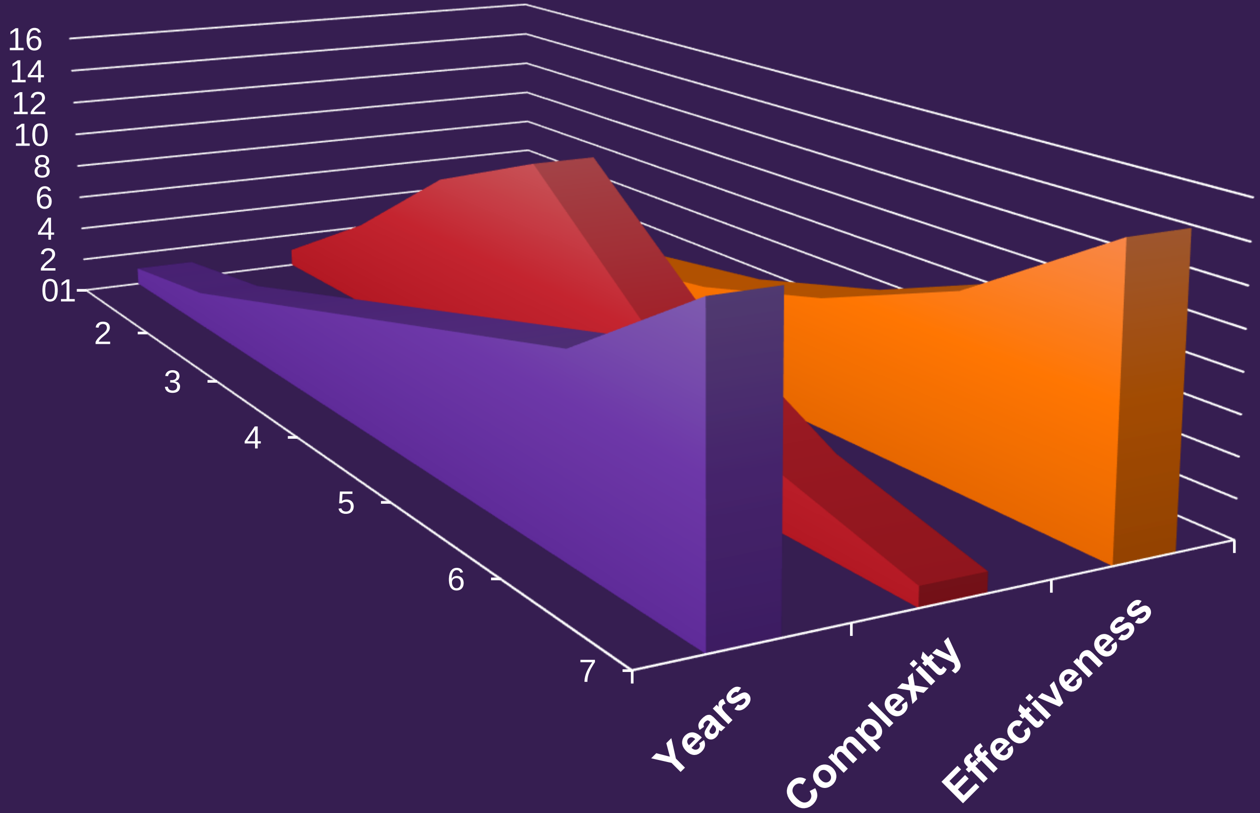
Focus on **actual facts** rather than catching up with all new language features and engineering practices



simple code
as written by a good, seasoned developer

**is different from**

simple code
as written by a junior, young developer


16
14
12
10
8
6
4
2
0
1
2
3
4
5
6
7
Years
Complexity
Effectiveness

**Humphrey’s Law**

The user of the software won’t know what she wants until she sees the software.

**Wegner’s Lemma**

An interactive system can never be fully specified nor can it ever be fully tested.

## One proven way of doing things is more than enough

https://github.com/youbiquitous/project-renoir/