



Revitalizing Legacy Code Approval Testing Unleashed

Erfahrungen aus der Legacy-App Portierung und Einblicke in die
von Verify

Patrick Drechsler



Patrick Drechsler

- Software Entwickler
- Beruflich: C#
- Interessen:
 - Software Crafting
 - Test-Driven Development
 - Funktionale Programmierung
 - Domain-Driven Design
- Background: Naturwissenschaftler
- Slides sind online: Siehe QR-Code





Legacy-Context

- Domain
- Tech-Stack(s)
- Architecture



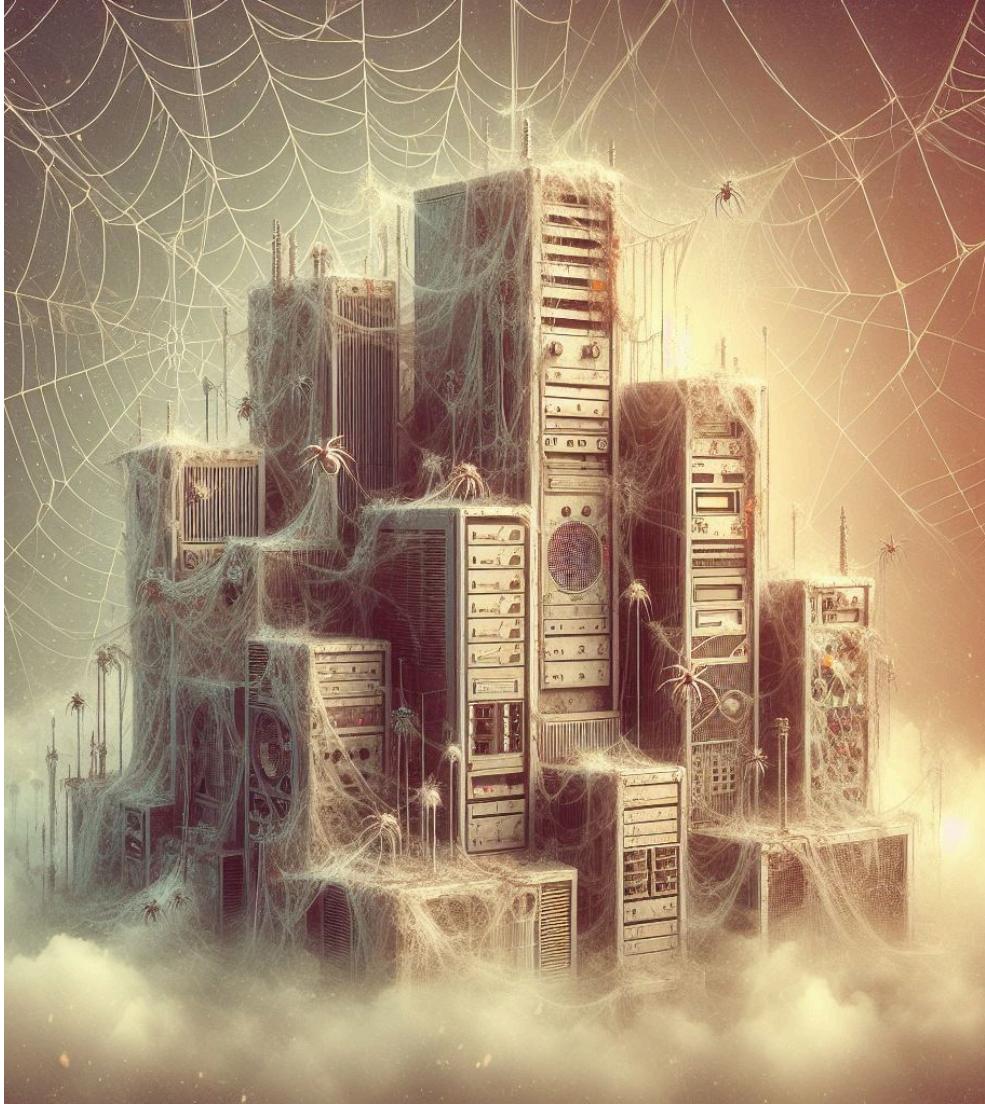
Domain

- expert system
- user: engineers in sales
- Special features:
 - SI units
 - floating point numbers
 - mathematics
 - not just rule of three
 - complex formulas (including integral calculus)
 - a lot of formulas ($x * 10^3$ LoC)
 - mathematics is core domain!
- perfect match for my background 😎



Tech-Stack(s)

- Main Stack: **LAMP**
 - Linux
 - Apache
 - MySQL
 - PHP
 - ✨ Angular 1 (!) ✨
- Other Stacks
 - C++
 - MATLAB



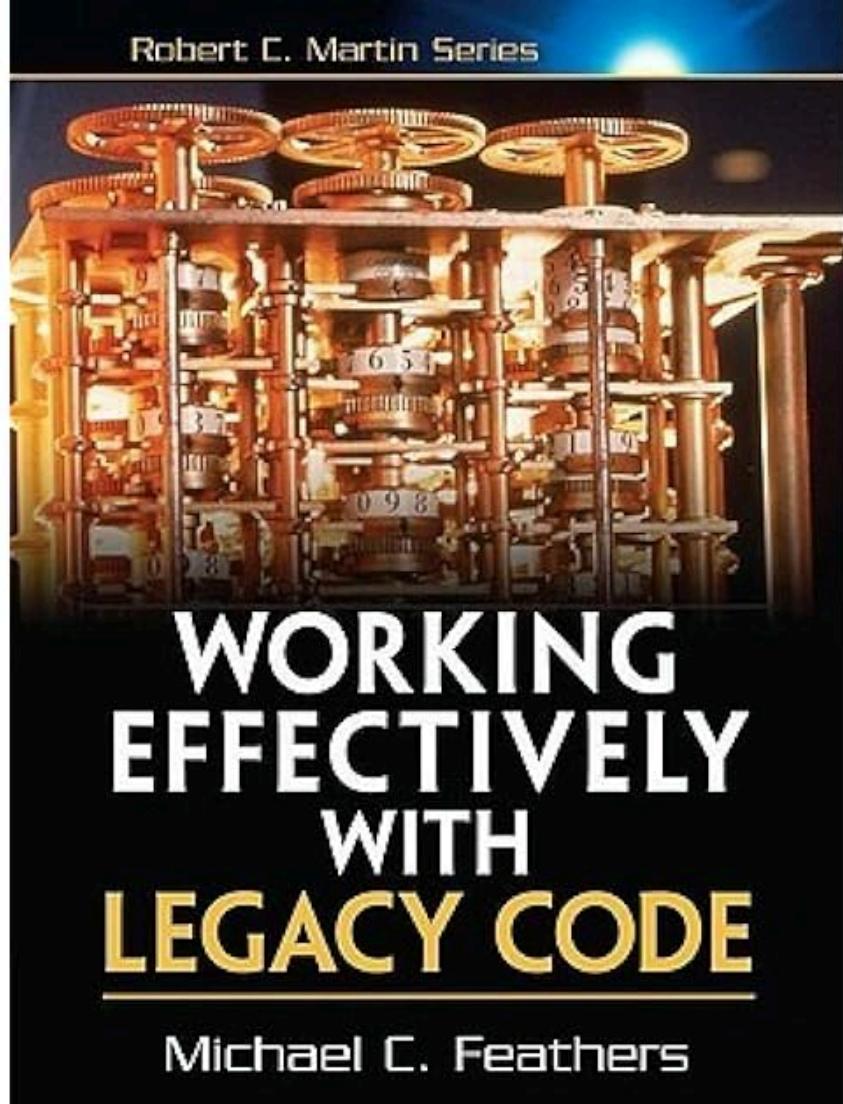


Architecture

- Frontend (FE): Angular 1
- Backend (BE): PHP
- Bonus: External system calls to FE & BE
 - Let's ignore this for now

Find a "Seam"

- What is a Seam? ➡ M. Feathers "Working Effectively with Legacy Code"
- "a seam is a place where you can alter behavior in your program without editing in that place"
- I used a "feature toggle" as seam





Find a "Seam" - Example

Redirecting a php request to a new **dotnet console** application

```
// Somewhere inside a PHP controller class...
//
// Seam which toggles between PHP and .NET
if ($this->useDotNet) { // 👍 Feature Toggle
    // C# calculation (new)
    return $this->calcDotNet("calculate", $request);
}
else {
    // PHP calculation (legacy)
    return new CalcWithPhp($request);
}

// ⚠️ Use the "Seam"
function calcDotNet($endpointName, Request $request)
{
    // ...
    $encodedJson = base64_encode($request->getContent());
    $result = shell_exec("".$dotnetProgramm." ".$endpointName." ".$encodedJson."");
    return $result;
}
```



Porting Code from PHP to C# (1/2)

- ▲ Pay attention to data structures (dynamic vs. static typing)

```
$foo[1] = 1;           // 🤔 is foo an int ?
$foo[2] = 2.3;         // 🤔 or a float ?
$foo[3] = "3";          // 🤔 or a string ?
// ...
// in some derived class (developed years later, by a different developer):
$foo["bar"] = [1, "2", 3.3]; // 🤔 or ???
```

- and floating point numbers (more on that later)...



Porting Code from PHP to C# (2/2)

- a lot of typing (no AI usage)
- Restructuring / Refactoring:
 - ⚠ Be extremely careful and conservative with restructuring
 - I disagreed with certain decisions in the old system (in my opinion way too much inheritance, followed by a lot of if/else in base classes)
 - 🤞 I mapped each product type to an independent type without inheritance (a lot of **code duplication**)
 - 🤞 I added "Stateless constructs": even more **code duplication**
- took 2-3 months
- no tests during that time 😭😭😭



Walking on water and **developing software** from a specification are easy if both are **frozen**. – Edward V. Berard



Testing the Ported Code

- Get hundreds of realistic example JSONs from the customer
- Run through the old system, save responses
- Run through the new system, compare responses with saved responses
- Rinse and repeat until the code coverage of the new .NET code is close to 100%
 - Extensive use of code coverage tools

How does that work in detail?



Definitions

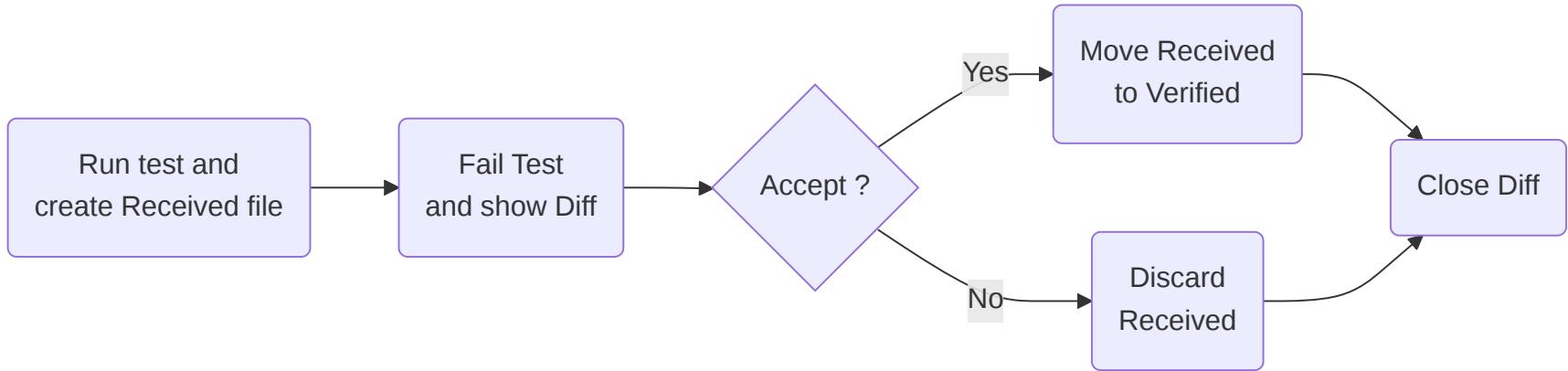
- Golden Master Test
- Approval Testing
- Snapshot Testing
- Verify
- Regression Test
- Acceptance Test
- Characterization Test

We'll stick with "Approval Testing" and "Verify" for now. And discuss the others later...



Initial Workflow

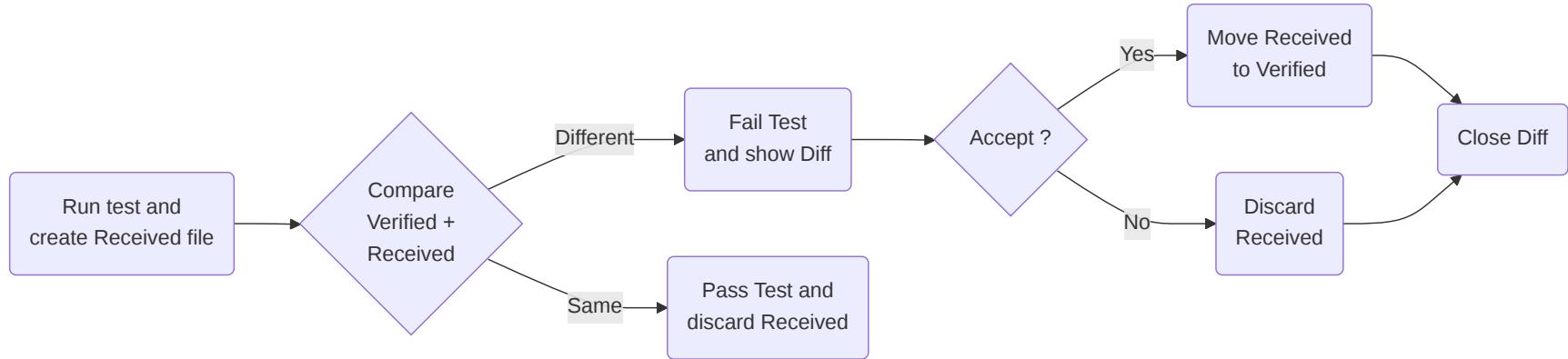
- No existing ``.verified.`` file.





Subsequent Workflow

- Existing `'.verified.'` file is compared with `'.received.'` file...





Hello World Example

```
public record Person(string FirstName, string LastName, int Age);

// ⚠ Fact must return Task!
[Fact]
public Task VerifyPersonTest()
{
    var homer = new Person("Homer", "Simpson", 39);
    return Verify(homer); // 👍 don't forget to "return" the Task
}
```

Verified text file:

```
{
    FirstName: Homer,
    LastName: Simpson,
    Age: 39
}
```



Where is the Arrange-Act-Assert (GWT)?

```
public record PersonRequest(string FirstName, string LastName, int Age);
public record PersonResponse(string FirstName, string LastName, int Age)
{
    public static PersonResponse FromRequest(PersonRequest request) =>
        new (request.FirstName, request.LastName, request.Age);
}
```

```
[Fact]
public Task PersonResponse_FromRequest()
{
    // Arrange / Given
    PersonRequest homer = new ("Homer", "Simpson", 39);

    // Act / When
    var actual = PersonResponse.FromRequest(homer);

    // Assert / Then
    return Verify(actual);
}
```

Verified text file:

```
{
    FirstName: Homer,
    LastName: Simpson,
    Age: 39
}
```

AAA vs GWT is the same mindset



Verify - Randomness

No problem 🤞 "Scrubbers" to the rescue

- GUIDs (by default)
- TimeStamps (by default)



Randomness - Default Scrubbers

```
public record Person(  
    string FirstName,  
    string LastName,  
    int Age,  
    Guid Id,           // 👈  
    DateTime? AquiredAt, // 👈  
    DateTime CreatedAt, // 👈  
    DateTime? UpdatedAt); // 👈
```

[Fact]

```
public Task PersonTest()  
{  
    var now = DateTime.Now;  
    var anotherDate = DateTime.Now;  
    var homer = new Person(  
        "Homer",  
        "Simpson",  
        39,  
        Guid.NewGuid(), // 👈  
        anotherDate,    // 👈  
        now,            // 👈  
        now);          // 👈
```

```
{  
    FirstName: Homer,  
    LastName: Simpson,  
    Age: 39,  
    Id: Guid_1,           // 👈  
    AquiredAt: DateTime_1, // 👈 "1"  
    CreatedAt: DateTime_2, // 👈 "2"  
    UpdatedAt: DateTime_2 // 👈 "2"  
}
```



Verify - Custom Scrubbers

<https://github.com/VerifyTests/Verify/blob/main/docs/scrubbers.md>

- Example when generating SVGs using `Plotly.NET`: Scrub all lines containing ``#clip`` followed by a word character
- ``ScrubLinesWithReplace`` and friends

```
// F#
let settings = VerifySettings ()
settings.ScrubLinesWithReplace (fun line →
    System.Text.RegularExpressions.Regex.Replace(line, "#clip\\w+", "#clipSCRUBBED"))

// C# (unverified)
var settings = new VerifySettings();
settings.ScrubLinesWithReplace(line =>
    System.Text.RegularExpressions.Regex.Replace(line, "#clip\\w+", "#clipSCRUBBED"));
```



Verify: Diff-Tooling for Devs

- Windows: [DiffEngineTray](#)
- [Resharper test runner](#)
- [Rider test runner](#)
- Terminal via `dotnet verify` :
[Verify.Terminal](#)
- 1st class support for all major IDEs
- 1st class integration for most common
[diff tools](#)

Very cool: customizable to your needs!



Verify - Setup

- We can define the output folder, file extensions, etc.
- Example ``ModuleInitializer`` for global setup, defining the output folder:

```
using System.Runtime.CompilerServices;

public static class ModuleInitializer
{
    [ModuleInitializer]
    public static void Initialize()
    {
        // To prevent cluttering the main folder, we will collect all verified snapshots in a dedicated folder.
        // For details, see: https://github.com/VerifyTests/Verify/blob/main/docs/naming.md#derivepathinfo
        DerivePathInfo(
            _, projectDirectory, type, method) => new(
                directory: Path.Combine(projectDirectory, "VerifiedData"),
                typeName: type.Name,
                methodName: method.Name));
    }
}
```



Verify - CI

- works out of the box
- No need to install anything on the CI server
- Customizable to your needs





Verify - JSON/XML

- JSON/XML are first-class citizens:
 - Instead of verifying a string, you can verify a JSON/XML object:
 - ``VerifyJson`` instead of ``Verify``
 - ``VerifyXml`` instead of ``Verify``
 - These customized methods will fail fast if the input is not valid JSON/XML



Verify - JSON/XML: Example

```
string InvalidJson = """{ "FirstName": "Homer" """;  
  
[Fact]  
public Task Invalid_json_demo1() => Verify(InvalidJson); // 👍 Verify, vs...  
  
[Fact]  
public Task Invalid_json_demo2() => VerifyJson(InvalidJson); // 👍 VerifyJson 😎
```

Error message from second test:

```
Argon.JsonReaderException: Unexpected end of content while loading JObject
```

It does not try to compare the invalid JSON with the verified JSON.

So, if you know that you are working with JSON/XML, use ``VerifyJson` / `VerifyXml`` !



Verify - Simplify Reading Test Data

```
[Fact]
public Task Reading_from_file()
{
    var fileContent = GetFileContent("valid-demo1.xml");
    return VerifyXml(fileContent);
}

// This saves us the hassle from having to deal with file paths in the tests.
// No more marking the file as "Copy if newer" or "Copy always".
private static string GetFileContent(string filename)
{
    const string sampleFolder = "SampleData";
    var relative = CurrentFile.Relative(Path.Combine(sampleFolder, filename));
    var fileContent = File.ReadAllText(relative);
    return fileContent;
}
```



Verify - Floating Point Numbers

- Floating point numbers are always a joy 
- Especially when working with different programming languages and platforms
- .NET will produce different results depending on the platform (Windows, Linux, macOS)
 - especially when doing real math: [MathNet.Numerics](#)
 - probably a niche case, but be aware of it
 - CI and/or target platform might differ from your dev machine 
 - possible solutions... 



Verify - Floating Point Numbers

Verify offers different strategies:

- Custom rounding

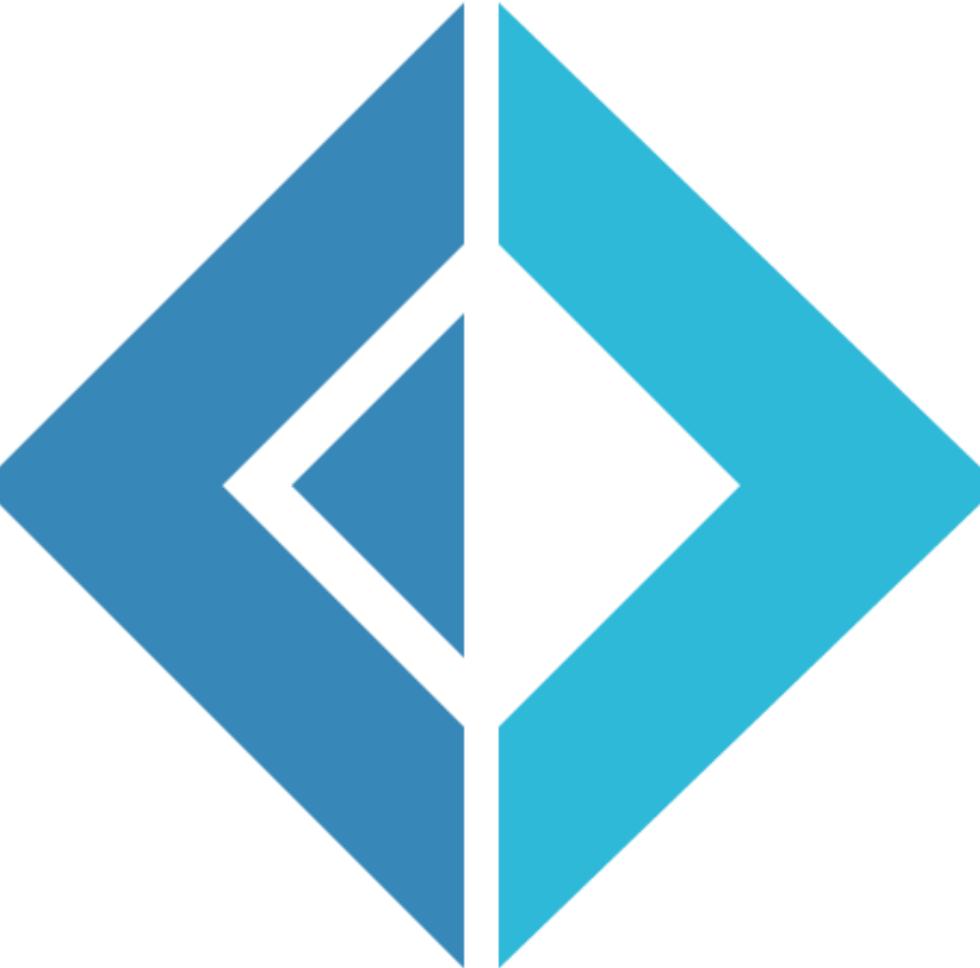
```
VerifierSettings.AddExtraSettings(x => x.FloatPrecision = 8);
```

- Custom tests for each platform (if above fails)

```
// ...
settings.UniqueForOSPlatform()
// ...
```

Drawback:

- works on Linux dev machine, CI pipeline, target platform
- fails on Windows dev machine, until windows dev commits 😞



Verify - F# Support

works out of the box 😎



Verify - Many Extensions

[https://github.com/VerifyTests/Verify?
tab=readme-ov-file#extensions](https://github.com/VerifyTests/Verify?tab=readme-ov-file#extensions)

- Images
- PDFs
- Databases
- HTTP
- Desktop-UI
- Infrastructure
- and many more...

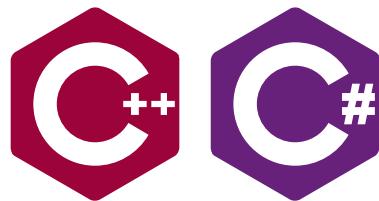
- [Verify.Avalonia](#): Verification of [Avalonia UIs](#).
- [Verify.Blazor](#): Verification of [Blazor Component](#) via [bunit](#) or via raw Blazor rendering.
- [Verify.Brighter](#): Verification of [Brighter](#) bits.
- [Verify.CommunityToolkit.Mvvm](#): Verification of [CommunityToolkit.Mvvm](#).
- [Verify.Cosmos](#): Verification of [Azure CosmosDB](#).
- Bunsen-Burner support: [BunsenBurner.Verify.NUnit](#) / [BunsenBurner.Verify.Xunit](#).
- [Verify.DiffPlex](#): Comparison of text via [DiffPlex](#).
- [Verify.DocNet](#): Verification of pdfs via [DocNet](#).
- [Verify.EntityFramework](#): Verification of EntityFramework bits.
- [Verify.FakeItEasy](#): Verification of [FakeItEasy](#) bits.
- [Verify.Flurl](#): Verification of [Flurl](#) bits.
- [Verify.HeadlessBrowsers](#): Verification of Web UIs using [Playwright](#), [Puppeteer Sharp](#), or [Selenium](#).
- [Verify.Http](#): Verification of Http bits.
- [Verify.ICSharpCode.Decompiler](#): Comparison of assemblies and types via [ICSharpCode.Decompiler](#).
- [Verify.ImageHash](#): Comparison of images via [ImageHash](#).
- [Verify.ImageSharp.Compare](#): Verification and comparison of images via [Codeuctivity.ImageSharp.Compare](#).
- [Verify.ImageMagick](#): Verification and comparison of images via [Magick.NET](#).
- [Verify.ImageSharp](#): Verification of images via [ImageSharp](#).
- [Verify.MailMessage](#): Verification of [MailMessage](#) and related types.
- [Verify.MassTransit](#): Support for MassTransit test helpers.
- [Verify.MicrosoftLogging](#): Verify MicrosoftLogging.
- [Verify.MongoDB](#): Verification of [MongoDB](#) bits.
- [Verify.Mog](#): Verification of [Mog](#) bits.
- [Verify.NodaTime](#): Support for [NodaTime](#).
- [Verify.NewtonsoftJson](#): Support for converting [Newtonsoft.Json](#) types (JObject and JArray).
- [Verify.NServiceBus](#): Verify NServiceBus Test Contexts.
- [Verify.NSubstitute](#): Support for [NSubstitute](#) types.
- [Verify.Nupkg](#): Verification of [NuGet.nupkg](#) files.
- [Verify.PdfPig](#): Verification of pdfs via [PdfPig](#).
- [Verify.Phash](#): Comparison of images via [Phash](#).
- [Verify.Quibble](#): Comparison of objects via [Quibble](#).
- [Verify.QuestPDF](#): Verification of pdf documents via [QuestPDF](#).
- [Verify.RavenDb](#): Verification of [RavenDb](#) bits.
- [Verify.SendGrid](#): Verification of [SendGrid](#).
- [Verify.Serilog](#): Verification of [Serilog](#) bits.
- [Verify.SqlServer](#): Verification of [SqlServer](#) bits.
- [Verify.SourceGenerators](#): Verification of C# Source Generators.



Verify - For all the languages!

Similar libraries exist for most programming languages.

Overview: <https://github.com/approvals>





Definitions - Revisited

- Synonyms:
 - Golden Master Test
 - Approval Test
 - Snapshot Test
 - Verification Test / Verify
- NOT a synonym for:
 - **▲ Regression Test**
 - a test which is run to ensure that the code still works after a change
 - **▲ Acceptance Test**
 - a test which is run to ensure that the code works as expected
 - **▲ Characterization Test** (Martin Fowler)
 - a test which is run to understand the behavior of the code



Verify - Maintainer

- Simon Cropp
- <https://github.com/SimonCropp>
- Simon reacts to issues and PRs very quickly





Summary

- Verify is a great tool when dealing with legacy code
- You must find a "seam" in the legacy code to inject the new code
- Once you found it, you can start porting the code and writing tests
- Try to keep the "testing the seam" loop as short as possible
- As always: tests should be automated (especially in CI)...
- You might still have to deal with the final output format (encoding, floating point numbers, etc.)



Thank You!

- patrick.drechsler@mathema.de
- <https://github.com/drapтик>
- <https://draptik.github.io/talks/>
- <https://floss.social/@drechsler>
- <https://www.linkedin.com/in/patrick-drechsler-drapтик/>

Slides

- QR Code or
- <https://draptik.github.io/2024-09-seneca-approval-testing/>
- sample code: [https://github.com/drapтика/2024-approval-testing/tree/seneca24/code/](https://github.com/drapтик/2024-approval-testing/tree/seneca24/code/)

Image sources: pixabay.com and perchance.org/ai-photo-generator

