Xpirit

Deepdive Unit Testing in .NET

Day 2

Agenda Day 2

- Check-in
- Recap
- Anti Corruption Layer
- Separate Unit & Integration Testing
- Test Doubles
- Code Coverage vs Test Quality
- Legacy
- .NET Core
- Checkout

Format

- Slides
- Demos
- Hands-On-Labs

→ TDD katas

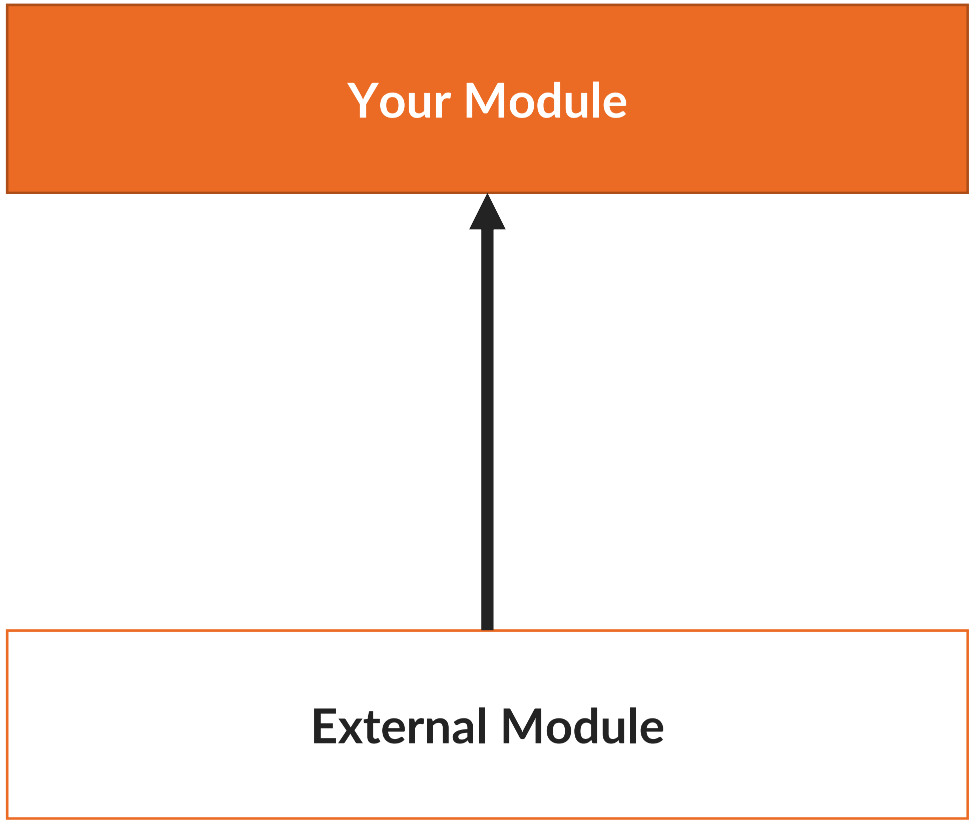
→ Pair programming

→ Ask for help

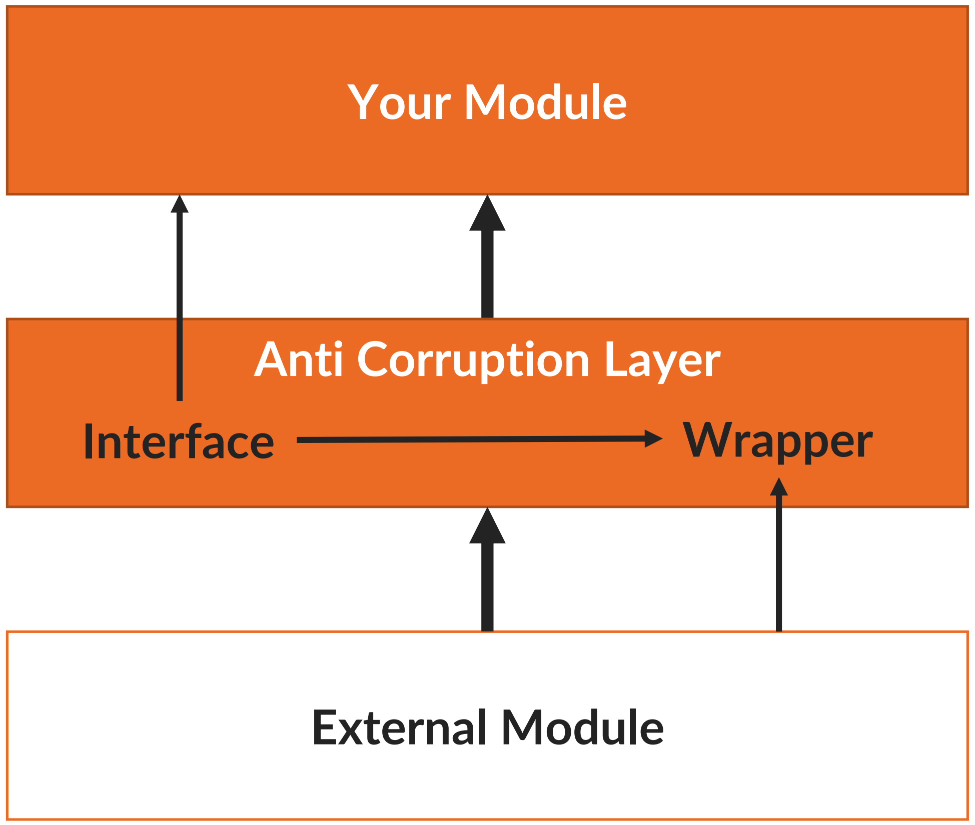
→ Central review afterwards

Anti Corruption Layer

Untestable code due to dependency on external module



Testable code due to Anti Corruption Layer



Xpirit

Dealing with System.IO

- **Don’t depend directly on System.IO implementation!**

  → Directory.GetFiles(…)

  → File.Exist(…)

- **Implement an ACL or use a existing solution such as WrapThat.System and use IDirectory and IFile interfaces in your code.**

Xpirit

Hands-on-Labs

- **Perform TDD Kata 6 – Using game state (10 min timebox)**
  - → Pair programming
  - → Ask for help
  - → Central review afterwards

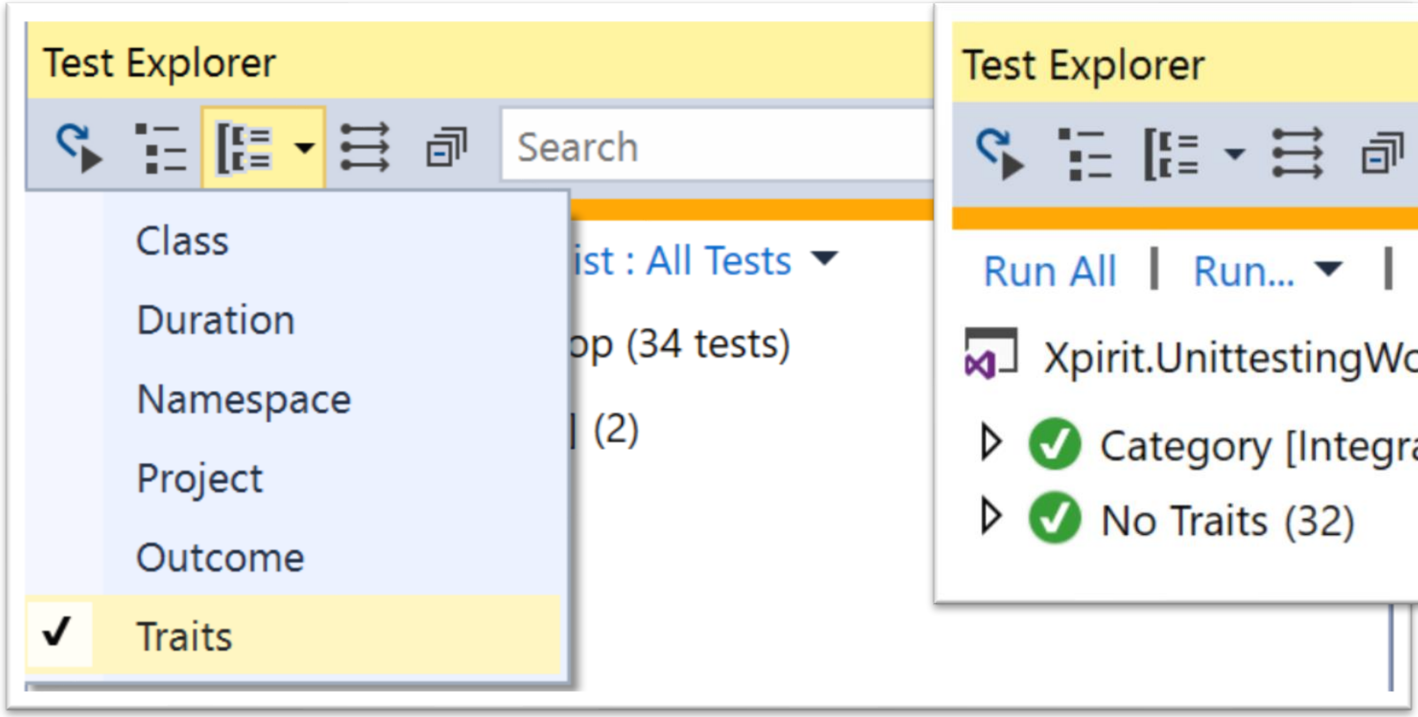
Hands-on-Labs

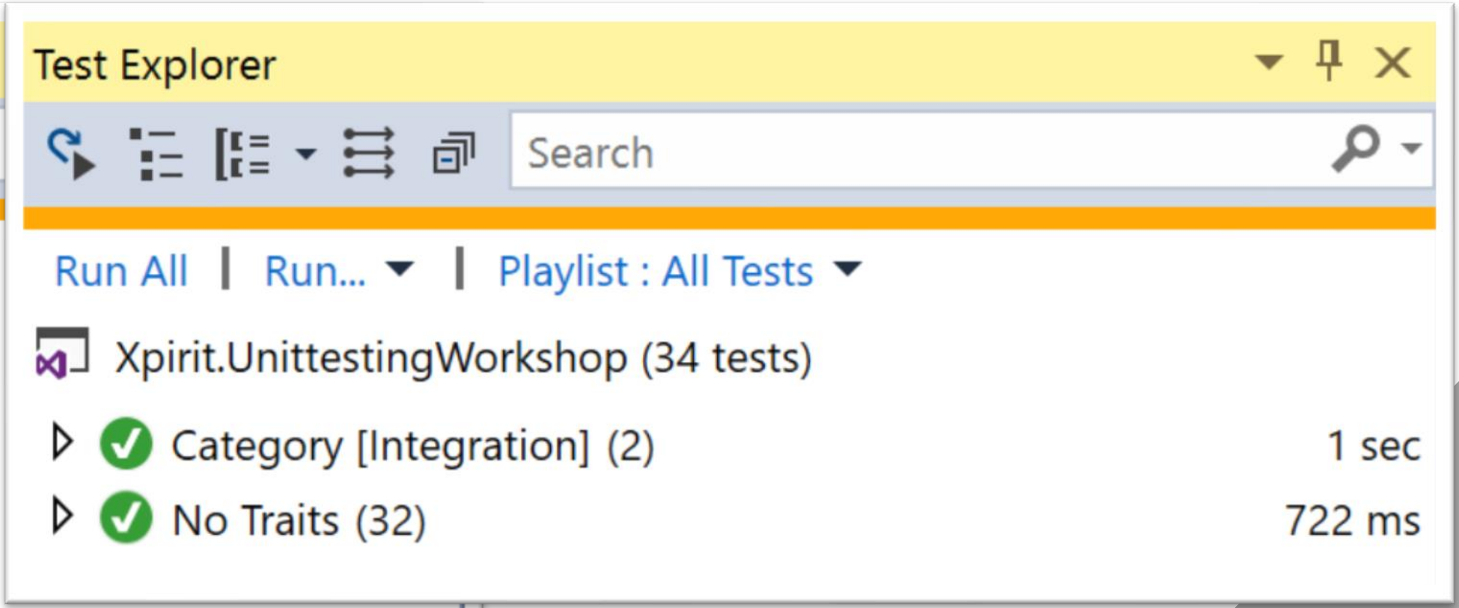
- **Perform TDD Kata 7 – Saving & loading game state (20 min timebox)**
  - → Pair programming
  - → Ask for help
  - → Central review afterwards

Separate Unit & Integration Testing

Traits & Categories

```
[Fact]
[Trait("Category", "Integration")]
public void LoadGameState_FromDisk_ThenGameStateShouldBeLoadedInGameEngine()
```





Xpirit.UnittestingWorkshop (34 tests)
Category [Integration] (2) 1 sec
No Traits (32) 722 ms

Hands-on-Labs

- **Perform TDD Kata 8 –loading game state from disk (15 min timebox)**

→ Pair programming

→ Ask for help

→ Central review afterwards

Test Doubles

Test Doubles

- Dummies
- Fakes
- Stubs
- Mocks

https://martinfowler.com/articles/mocksArentStubs.html

Dummies

- Objects which are required in SUT but their value is not of importance.
- Their state do not influence the outcome of the test.

Fakes

- Objects have working implementations, but usually take some shortcut which makes them not suitable for production.

Extract & Override with a Fake

```
public class GameLogger
{
   public void LogError(string message)
   {
       var fileName = GetLogFileName();
       Log(fileName, message);
   }

   private static string GetLogFileName()
   {   ...   }

   internal virtual void Log(string path, string message)
   {
         File.WriteAllText(path, message);
   }
}
```

Extract & Override with a Fake

```
public class FakeGameLogger : GameLogger
{
    internal override void Log(string path, string message)
    {
         // Don't do anything here
    }
}


[Fact]
public void TestGameWithLogger()
{
     // Arrange
     var fakeLogger = new FakeGameLogger();
     var gameEngine = new GameEngine(fakeLogger);
     ...
}
```

Stubs

- Objects to provide canned answers to calls made during the test.
- Usually not responding at all to anything outside what's programmed in for the test.

Xpirit

Stub functionality in Moq

```
// Arrange
var mock = new Mock<IGameEngine>();

mock.Setup(game => game.IsCompleted())
    .Returns(true);

mock.Setup(game => game.Start(It.IsAny<IEnumerable<Player>>()))
    .Returns(new GameState());
```

Mocks

- Objects pre-programmed with expectations which form a specification of the calls they are expected to receive.

Mock functionality in Moq

```
// Arrange with loose behavior
var mock = new Mock<IGameEngine>();

// Arrange with strict behavior
var mock = new Mock<IGameEngine>(MockBehavior.Strict);
mock.Setup(game => game.Move(It.IsAny<Player>(), It.IsAny<int>()));

// Assert
mock.Verify(game=> game.Move(It.IsAny<Player>(), It.IsAny<int>()),Times.Once);
```

Xpirit

AutoFixture

- “AutoFixture is an open source library for .NET designed to minimize the 'Arrange' phase of your unit tests in order to maximize maintainability.”
- Great for creating test objects (dummies, fakes, stubs).

https://github.com/AutoFixture/AutoFixture

AutoFixture examples

```
// Arrange
var fixture = new Fixture();

var someString = fixture.Create<string>();

var player = fixture.Create<Player>();

// Create IEnumerable of Player
var players = fixture.CreateMany<Player>();

// Setting one property
var playerWith5Moves = fixture.Build<Player>()
                             .With(p => p.NumberOfMoves, 5)
                             .Create();

var playerWithOneMove = fixture.Build<Player>()
                             .Do(p => p.Move())
                             .Create();
```

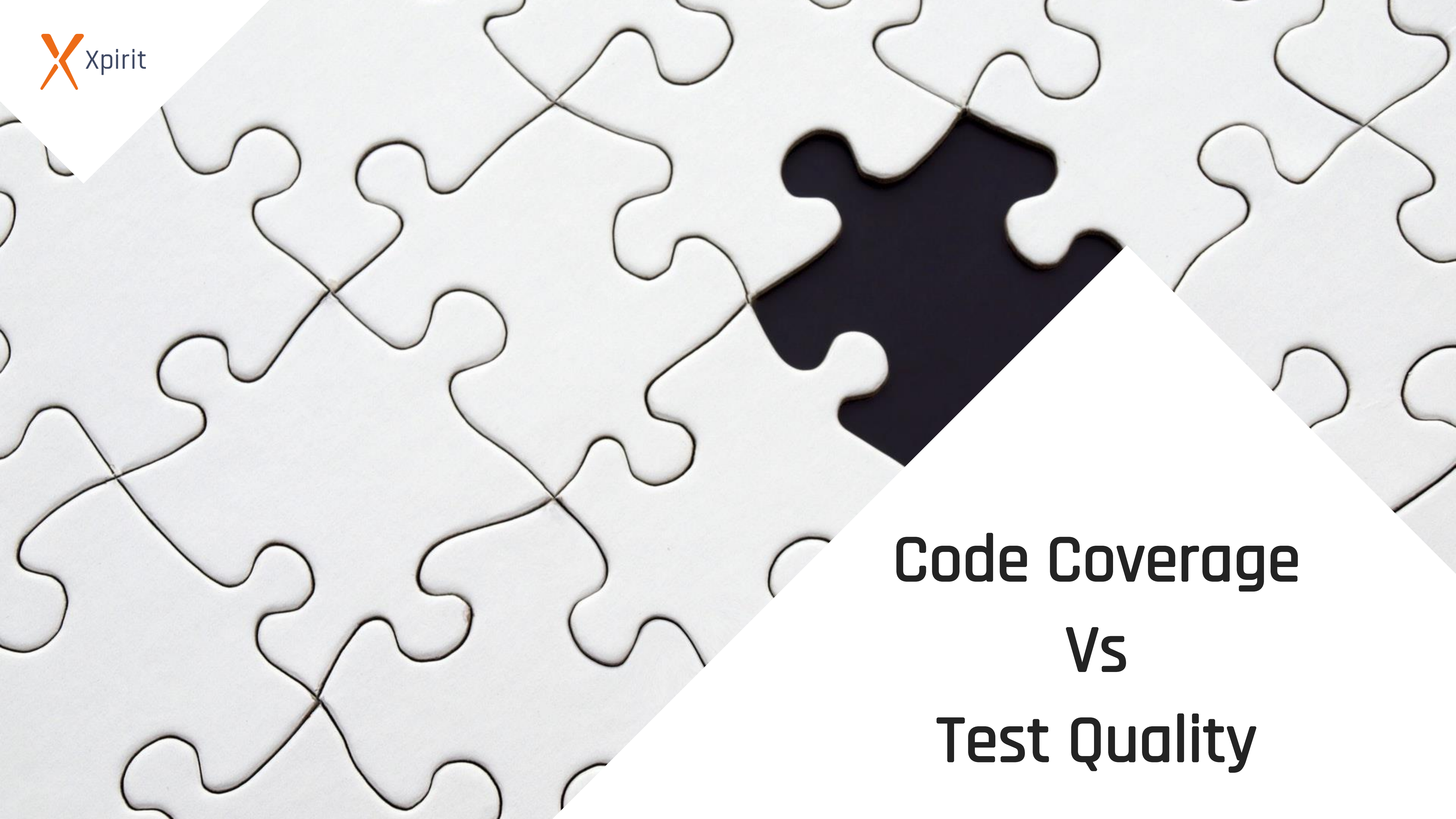
Hands-on-Labs

- **Perform TDD Kata 9 – Using AutoFixture (20 min timebox)**

→ Pair programming

→ Ask for help

→ Central review afterwards



Code Coverage
Vs
Test Quality

Code Coverage

- **Is:**

→ A metric that shows how much of your code is executed when your tests are run.

→ A way to visualize what scenarios you might have missed when testing.

- **Is not:**

→ A metric that shows how good your tests are or what the quality of your software is.

→ Something to strive for by itself.

Test Quality

- Quality of test code should be (near) quality of production code

→ See day 1 for a list of properties a good unit test has

- Tests should be reviewed

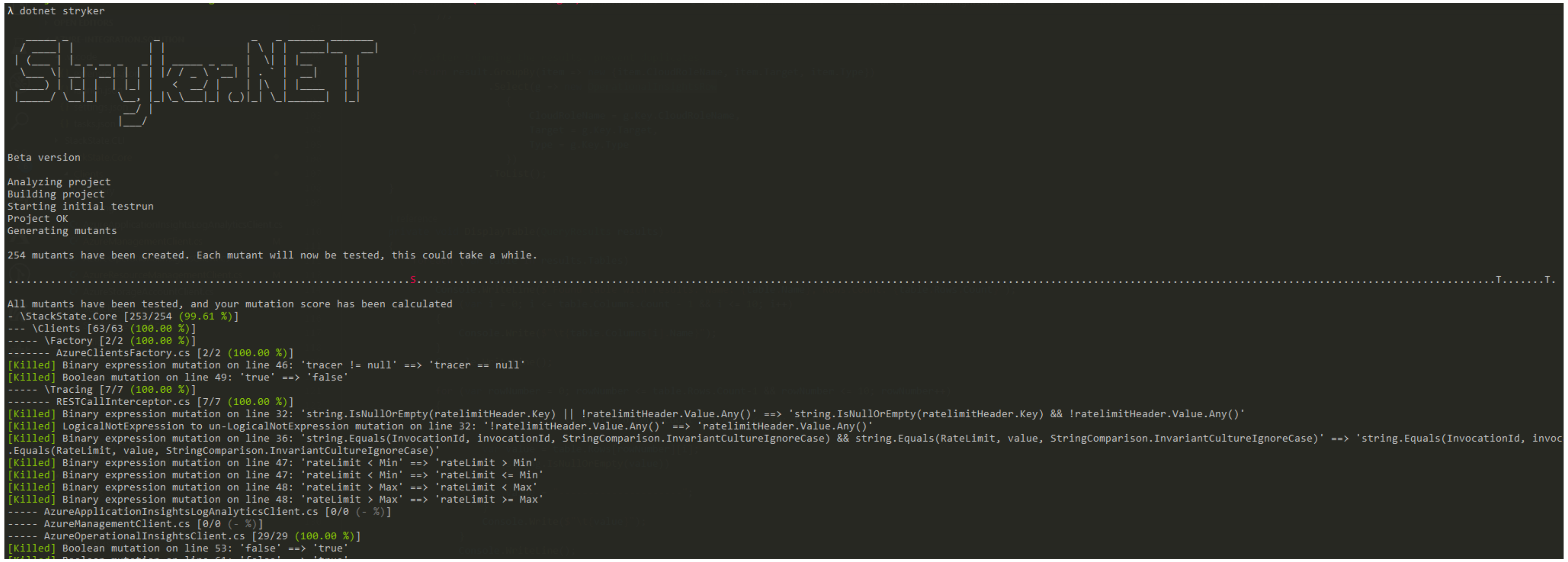
- Code Coverage should not be the leading metric

→ https://github.com/riezebosch/tdd-journey/blob/master/slides/20-frustration/8-mooooooooahr.md

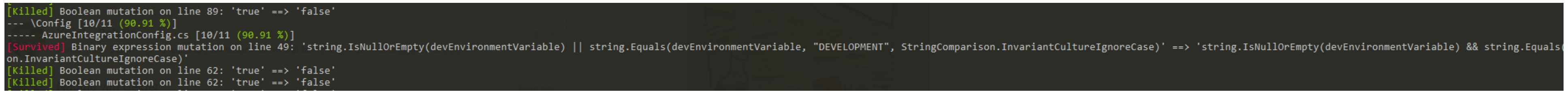
→ A better way? Mutation testing!

Mutation testing

- Stryker: Testing your tests by temporarily inserting bugs



```
λ dotnet stryker

Beta version

Analyzing project
Building project
Starting initial testrun
Project OK
Generating mutants

254 mutants have been created. Each mutant will now be tested, this could take a while.

..............................................................S..............................................................................................................................T.......T.

All mutants have been tested, and your mutation score has been calculated
- \StackState.Core [253/254 (99.61 %)]
--- \Clients [63/63 (100.00 %)]
----- \Factory [2/2 (100.00 %)]
------- AzureClientsFactory.cs [2/2 (100.00 %)]
[Killed] Binary expression mutation on line 46: 'tracer != null' ==> 'tracer == null'
[Killed] Boolean mutation on line 49: 'true' ==> 'false'
----- \Tracing [7/7 (100.00 %)]
------- RESTCallInterceptor.cs [7/7 (100.00 %)]
[Killed] Binary expression mutation on line 32: 'string.IsNullOrEmpty(ratelimitHeader.Key) || !ratelimitHeader.Value.Any()' ==> 'string.IsNullOrEmpty(ratelimitHeader.Key) && !ratelimitHeader.Value.Any()'
[Killed] LogicalNotExpression to un-LogicalNotExpression mutation on line 32: '!ratelimitHeader.Value.Any()' ==> 'ratelimitHeader.Value.Any()'
[Killed] Binary expression mutation on line 36: 'string.Equals(InvocationId, invocationId, StringComparison.InvariantCultureIgnoreCase) && string.Equals(RateLimit, value, StringComparison.InvariantCultureIgnoreCase)' ==> 'string.Equals(InvocationId, invoc
.Equals(RateLimit, value, StringComparison.InvariantCultureIgnoreCase)'
[Killed] Binary expression mutation on line 47: 'rateLimit < Min' ==> 'rateLimit > Min'
[Killed] Binary expression mutation on line 47: 'rateLimit < Min' ==> 'rateLimit <= Min'
[Killed] Binary expression mutation on line 48: 'rateLimit > Max' ==> 'rateLimit < Max'
[Killed] Binary expression mutation on line 48: 'rateLimit > Max' ==> 'rateLimit >= Max'
----- AzureApplicationInsightsLogAnalyticsClient.cs [0/0 (- %)]
----- AzureManagementClient.cs [0/0 (- %)]
----- AzureOperationalInsightsClient.cs [29/29 (100.00 %)]
[Killed] Boolean mutation on line 53: 'false' ==> 'true'
```



```
[Killed] Boolean mutation on line 89: 'true' ==> 'false'
--- \Config [10/11 (90.91 %)]
----- AzureIntegrationConfig.cs [10/11 (90.91 %)]
[Survived] Binary expression mutation on line 49: 'string.IsNullOrEmpty(devEnvironmentVariable) || string.Equals(devEnvironmentVariable, "DEVELOPMENT", StringComparison.InvariantCultureIgnoreCase)' ==> 'string.IsNullOrEmpty(devEnvironmentVariable) && string.Equals(
on.InvariantCultureIgnoreCase)'
[Killed] Boolean mutation on line 62: 'true' ==> 'false'
[Killed] Boolean mutation on line 62: 'true' ==> 'false'
```

Hands-on-Labs

- Run Stryker.NET on your TDD Katas code and kill some mutants

→https://github.com/stryker-mutator/stryker-net

Stryker – Work in progress

- **Version 0.4.0 was broken, 0.5.0 was broken, 0.6.0 works, sort of:**
  - → Skipping tests based on categories/traits not possible
  - → Feature 'default literal' is not available in C# 7.0. Please use language version 7.1 or greater.
  - → .ConfigureAwait(false) raises lots of “false positives”
  - → Not reliable yet: 99.78% score on a project with really not enough unit tests

Legacy

Hands-on-Labs

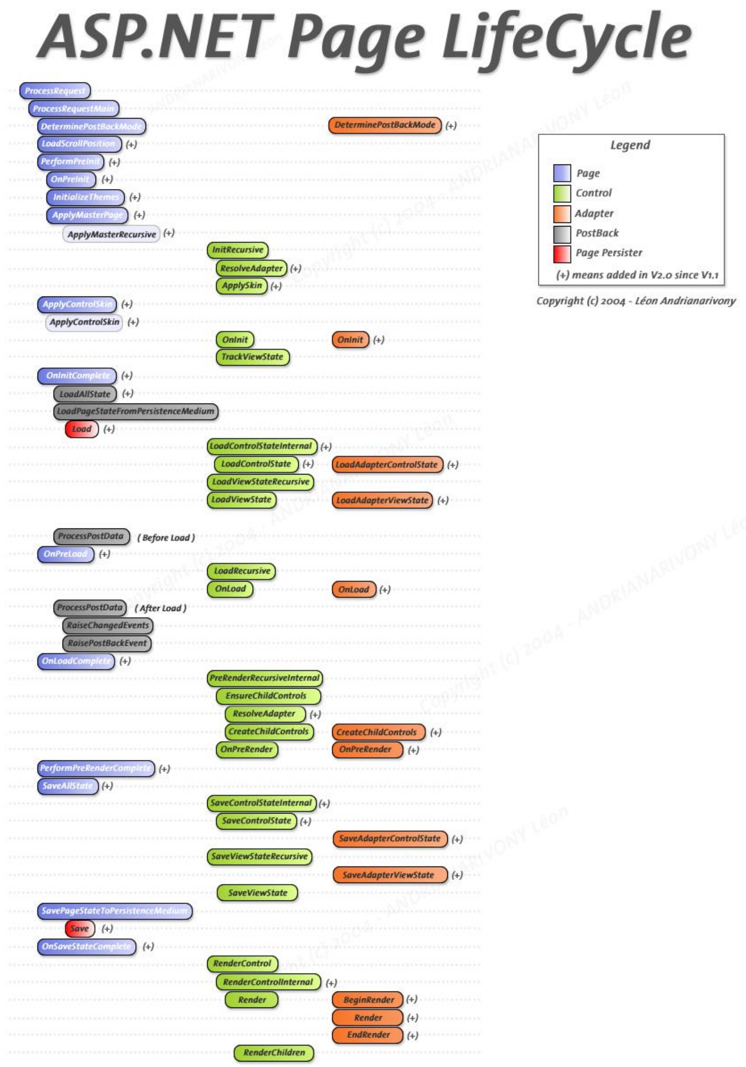
- Open UnitTesting\Xpirit.UnittestingLegacy\CCDSchool.sln

→ Instructions are in HandsOnLab\todo.md

Xpirit

Evaluation

- Extract classes
- Refactor / rewrite
- Manual testing



.NET Core

.NET Core Advantages

- Lightweight, faster & cross platform
- Possibility for self-contained apps
- Some new tools, such as Stryker, only support .NET Core
- Rich featureset for modern architectures

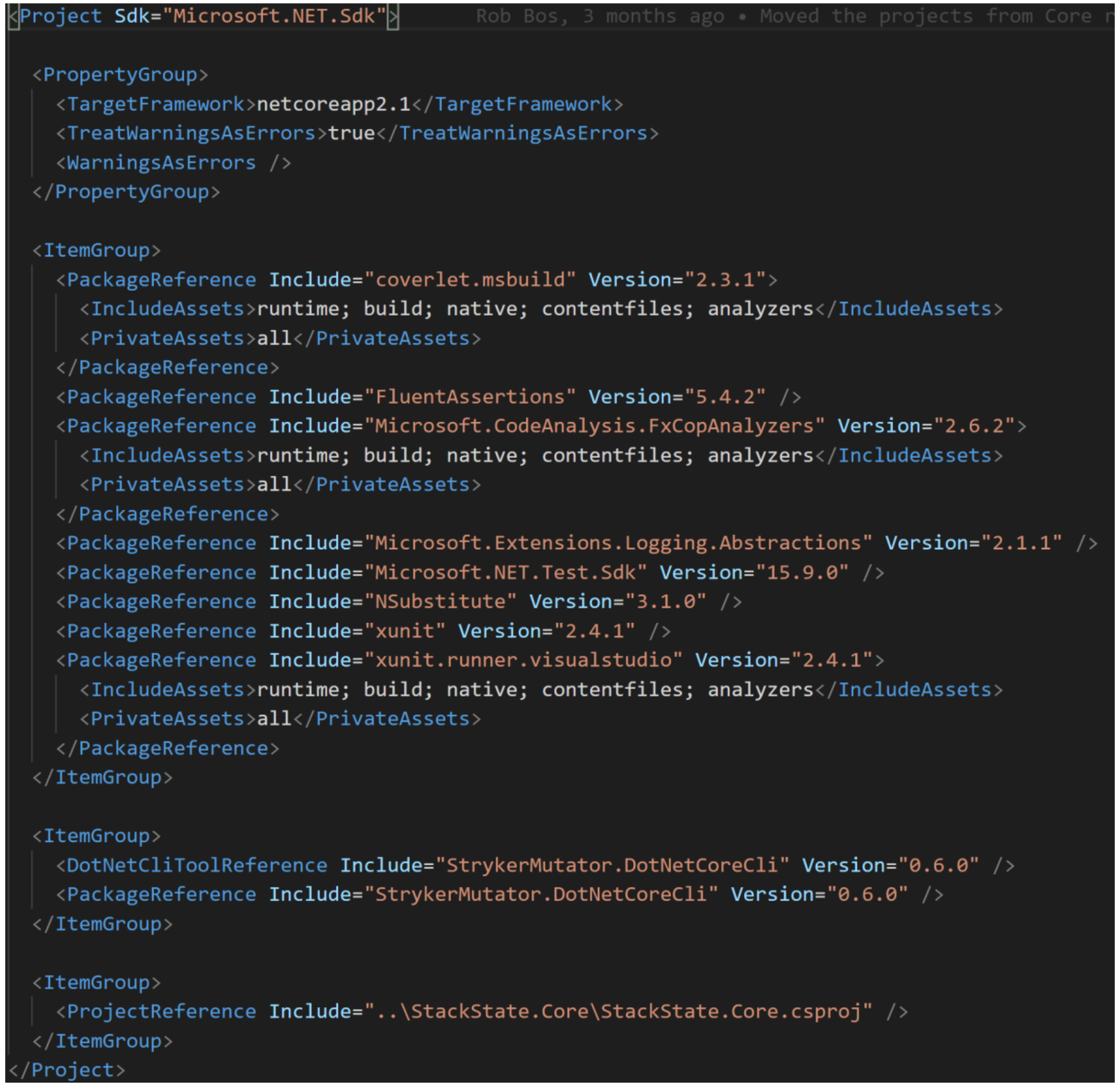
  → Like “TestServer” for integration testing a REST service

  → Or easy in-memory Entity Framework integration testing

  → Easy integration with Polly, OpenAPI and more
- Extensions, such as “dotnet outdated”
- New csproj (*also possible with .NET Framework*)

Xpirit

New csproj

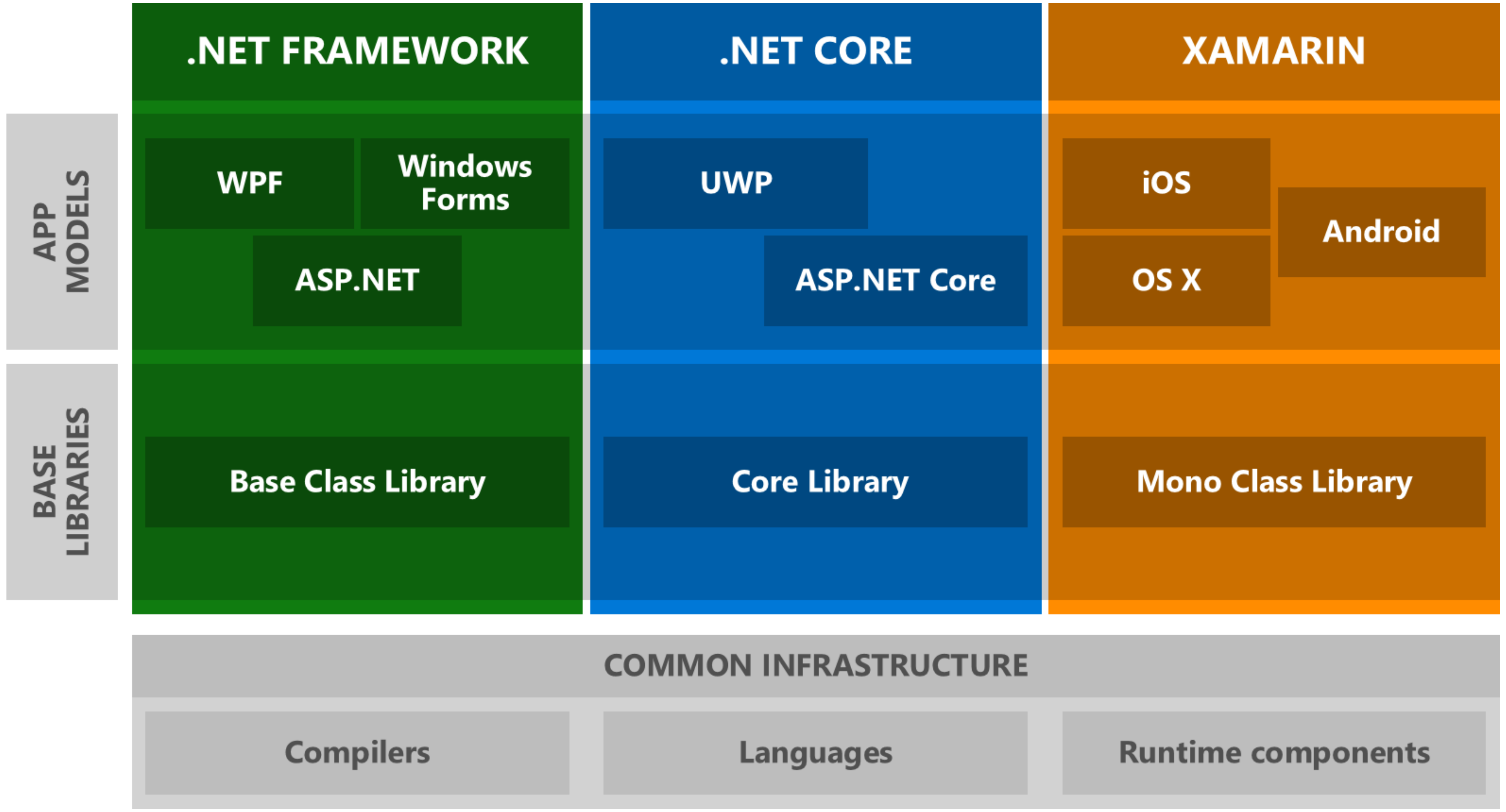


```
<Project Sdk="Microsoft.NET.Sdk">        Rob Bos, 3 months ago • Moved the projects from Core r

  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
    <TreatWarningsAsErrors>true</TreatWarningsAsErrors>
    <WarningsAsErrors />
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="coverlet.msbuild" Version="2.3.1">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="FluentAssertions" Version="5.4.2" />
    <PackageReference Include="Microsoft.CodeAnalysis.FxCopAnalyzers" Version="2.6.2">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.Extensions.Logging.Abstractions" Version="2.1.1" />
    <PackageReference Include="Microsoft.NET.Test.Sdk" Version="15.9.0" />
    <PackageReference Include="NSubstitute" Version="3.1.0" />
    <PackageReference Include="xunit" Version="2.4.1" />
    <PackageReference Include="xunit.runner.visualstudio" Version="2.4.1">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
  </ItemGroup>

  <ItemGroup>
    <DotNetCliToolReference Include="StrykerMutator.DotNetCoreCli" Version="0.6.0" />
    <PackageReference Include="StrykerMutator.DotNetCoreCli" Version="0.6.0" />
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\StackState.Core\StackState.Core.csproj" />
  </ItemGroup>
</Project>
```

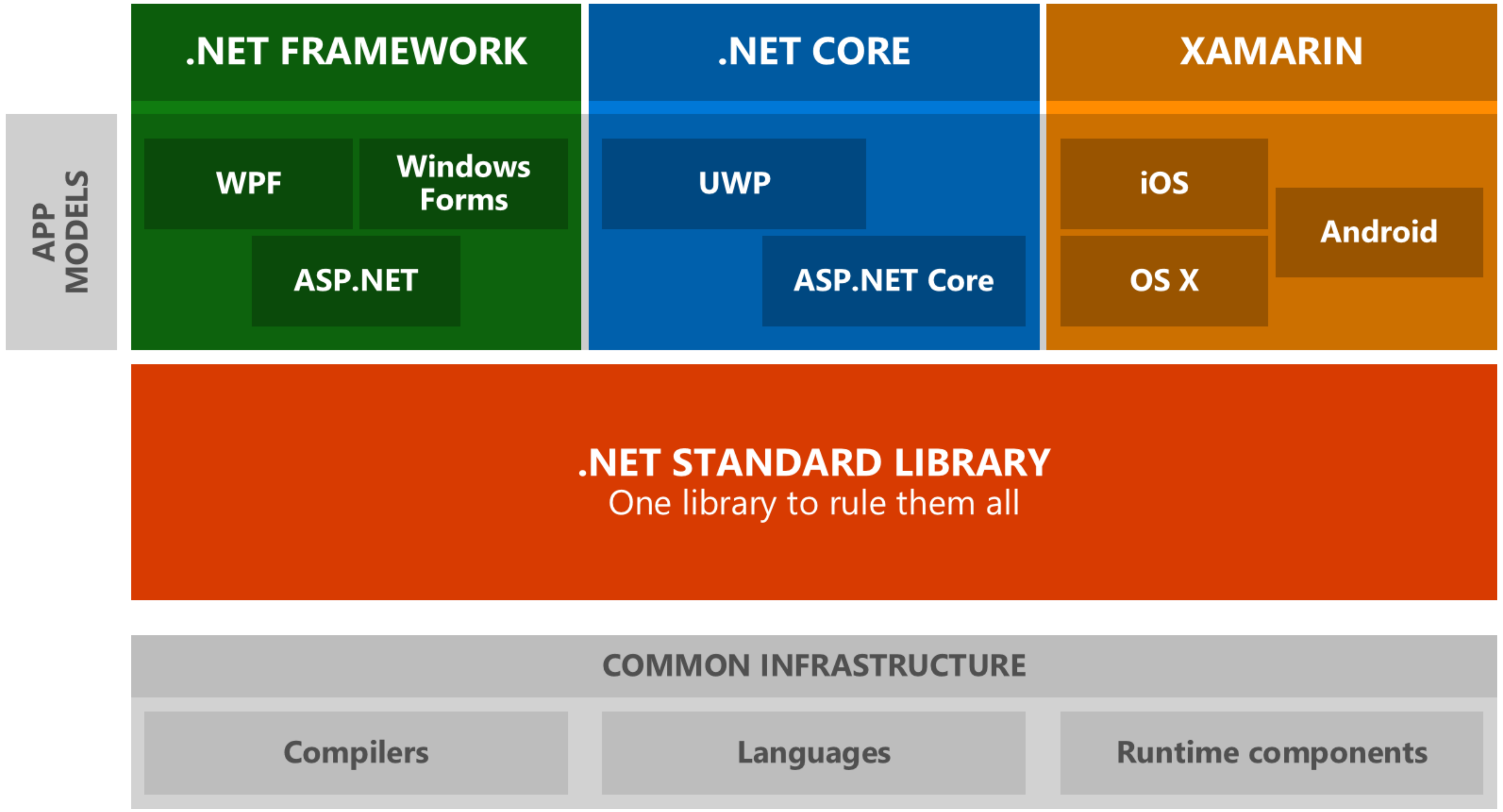
.NET Core Disadvantages

- Not everything is supported yet (WCF, WPF and more)
- Not all libraries support .NET Core (NServiceBus)
- Some stuff isn’t as polished as in full framework (code coverage)
- Few new things to learn
- Migrating takes some effort

.NET Standard



.NET Standard



.NET Standard

The following table lists the minimum platform versions that support each .NET Standard version.

.NET Standard	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**	**2.0**
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0
.NET Framework [1]	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1	4.6.1	4.6.1
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0
Universal Windows Platform	10.0	10.0	10.0	10.0	10.0	10.0.16299	10.0.16299	10.0.16299
Windows	8.0	8.0	8.1					
Windows Phone	8.1	8.1	8.1					

.NET Core Migration

- Manual steps required
- Install SDK's: locally, build servers, target servers
- Learn the CLI: dotnet build, dotnet clean, dotnet …
- Supporting libraries to .NET Standard first

  → Redo the .csproj, or create a new project and copy stuff back in

  → Everything is in .csproj: assemblyinfo, nuspec, nuget packages, etc
- Entry-points to .NET Core 2.x last
- Update build pipeline if you want

  → Build csproj instead of solution

Xpirit

.NET Core Experience

- **Works great in greenfield**

- **Works fine in brownfield, needs some work**

  → Run https://docs.microsoft.com/en-us/dotnet/standard/analyzers/portability-analyzer to get an idea of how much work

Check out

- What did you like best?
- What could be improved?
- Which topics should we cover next lesson?

Xebia