



Getting started

ArchUnit

Ricardo de Luna Galdino
Software Engineer

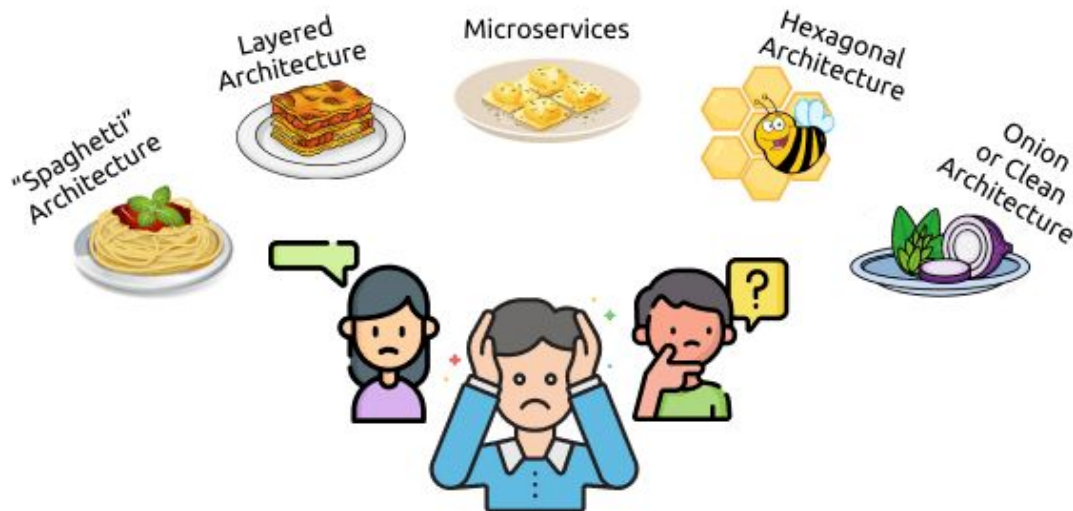
github.com/ricardogaldino/microlearning-archunit-dotnet

Software Architecture

É a **forma como** as “partes de um software”, ou “**componentes estruturais**”, são **organizados** em um sistema.

Traz questões como:

- **Definição da Estrutura** (organização) de um sistema de software;
- Identificação dos **relacionamentos** entre os componentes **Internos** e dos relacionamentos com componentes **Externos**;
- Separação das **responsabilidades** de cada componente, ou seja, o conjunto específico de funções que cada um deve realizar.



Software Architecture

Common web application architectures

Common web application architectures

Leitura:

docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

Common web application architectures

Article • 04/14/2022 • 20 minutes to read • 14 contributors



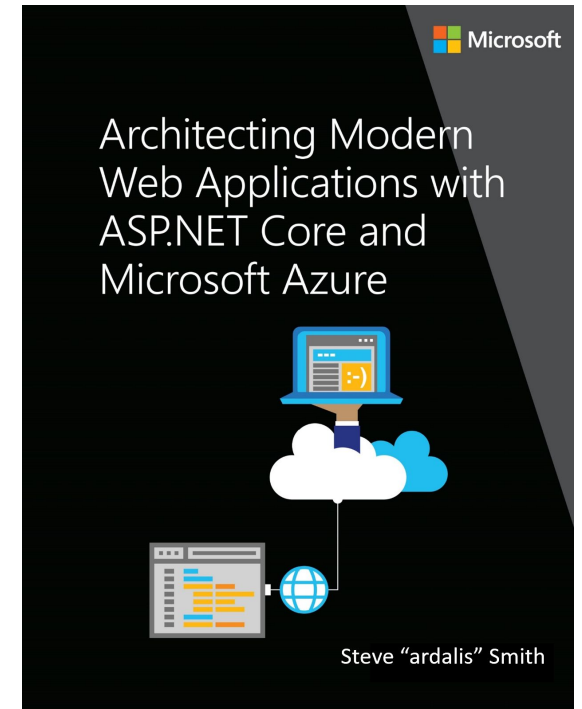
Tip

This content is an excerpt from the eBook, Architect Modern Web Applications with ASP.NET Core and Azure, available on .NET Docs or as a free downloadable PDF that can be read offline.

[Download PDF](#)



"If you think good architecture is expensive, try bad architecture." - Brian Foote and Joseph Yoder



Software Architecture

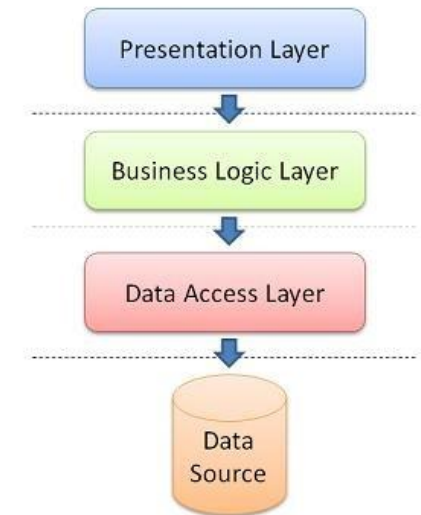
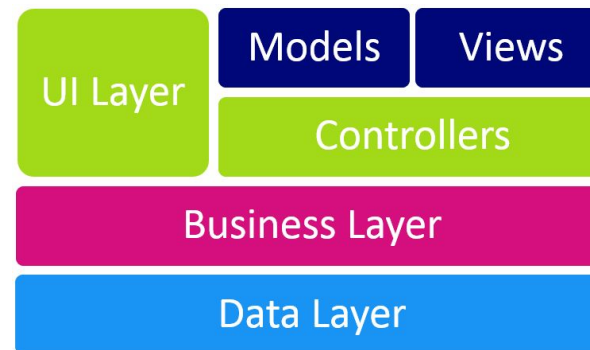
Common web application architectures

Three Layer Architecture

Arquitetura em 3 Camadas

Leitura:

docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures#traditional-n-layer-architecture-applications



Essas camadas são frequentemente abreviadas como **UI** (Interface do Usuário), **BLL** (Camada de Lógica de Negócios) e **DAL** (Camada de Acesso a Dados). Usando essa arquitetura, os usuários fazem solicitações por meio da camada de interface do usuário, que interage com a BLL. A BLL, por sua vez, pode chamar a DAL para solicitações de acesso a dados. A camada de interface do usuário não deve fazer solicitações à DAL diretamente nem interagir com persistência diretamente por outros meios. Da mesma forma, a BLL só deve interagir com persistência por meio da DAL. Assim, cada camada tem sua própria responsabilidade conhecida.

Software Architecture

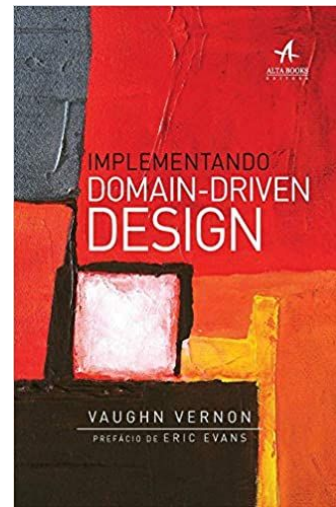
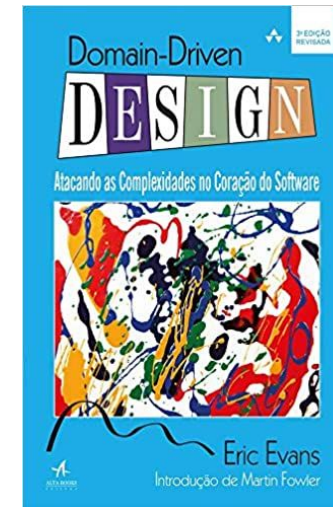
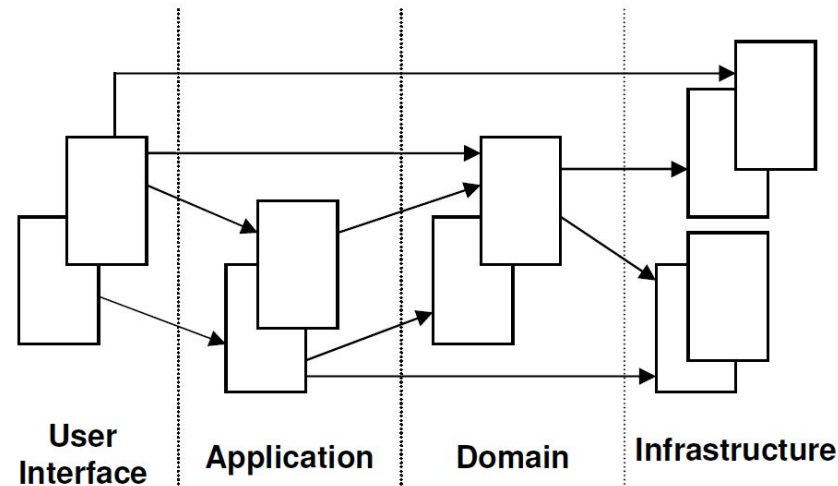
Common web application architectures

Domain-Driven Design (DDD)

Arquitetura orientada ao Domínio

(Compreensão dos processos e regras de um domínio)

Leitura: martinfowler.com/bliki/DomainDrivenDesign.html



Tradicionalmente separa as responsabilidades nas **camadas** de “**UI**” (User Interface), “**Application**” (Regras de negócio específicas da Aplicação), “**Domain**” (Entidades e Regras de negócio específicas da Entidade) e “**Infrastructure**” (Banco de Dados e Agentes Externos).

Software Architecture

Common web application architectures

Hexagonal Architecture (Ports-and-Adapters), Onion Architecture e Clean Architecture

Os softwares que seguem o princípio da **Inversão de Dependência**, bem como os princípios de **DDD** (Design Controlado por Domínio), **tendem a chegar a uma arquitetura semelhante**. Essa arquitetura foi conhecida por muitos nomes ao longo dos anos. Um dos primeiros nomes foi **Arquitetura Hexagonal (Portas e Adaptadores)**. Mais recentemente, **Arquitetura Cebola** e agora **Arquitetura Limpa**.

Embora essas arquiteturas variem um pouco em seus detalhes, elas são muito semelhantes. Todas elas têm o mesmo objetivo, que é a separação de interesses em camadas.

Cada uma tem pelo menos uma camada independente para "**Entidades e Regras de negócio específicas da Entidade**", "**Regras de negócio específicas da Aplicação**", "**Interface do Usuário**", "**Banco de Dados**" e "**Agentes Externos**".

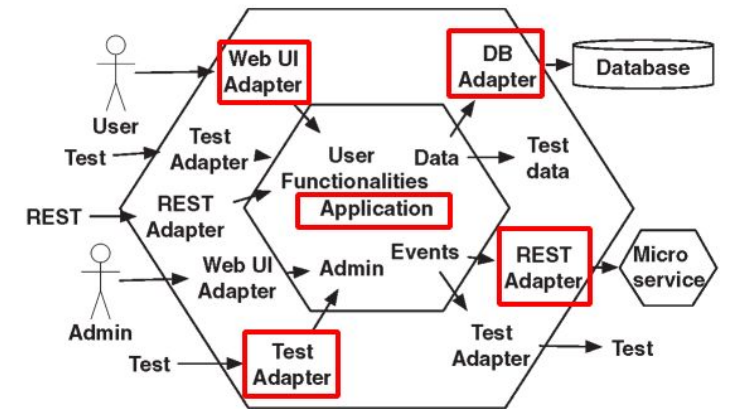
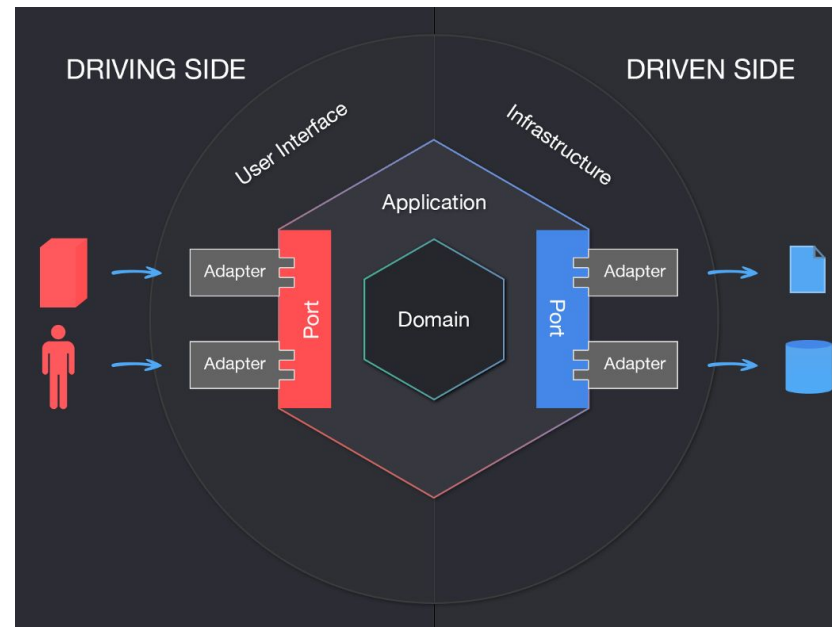
Hexagonal Architecture

(Ports-and-Adapters)

Leitura: alistair.cockburn.us/hexagonal-architecture

Software Architecture

Common web application architectures



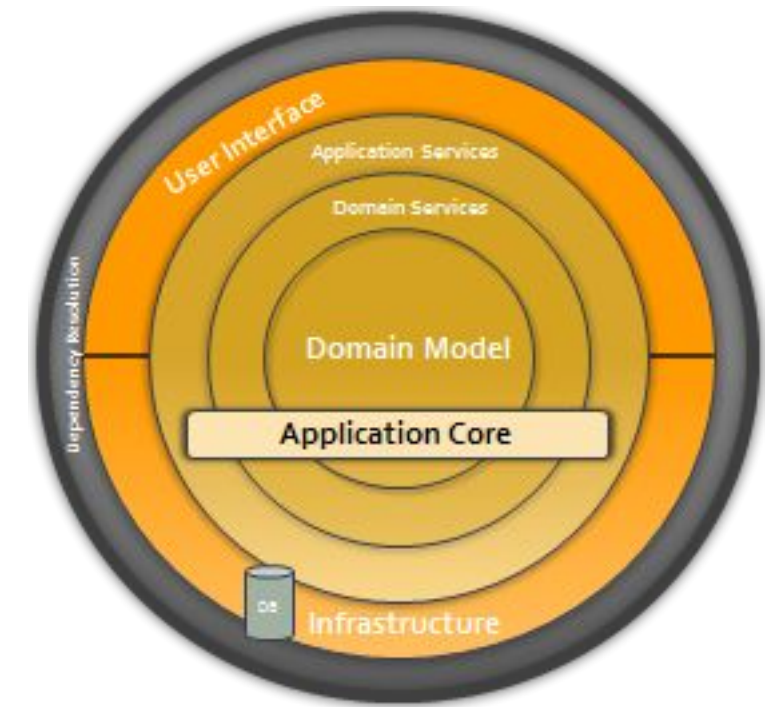
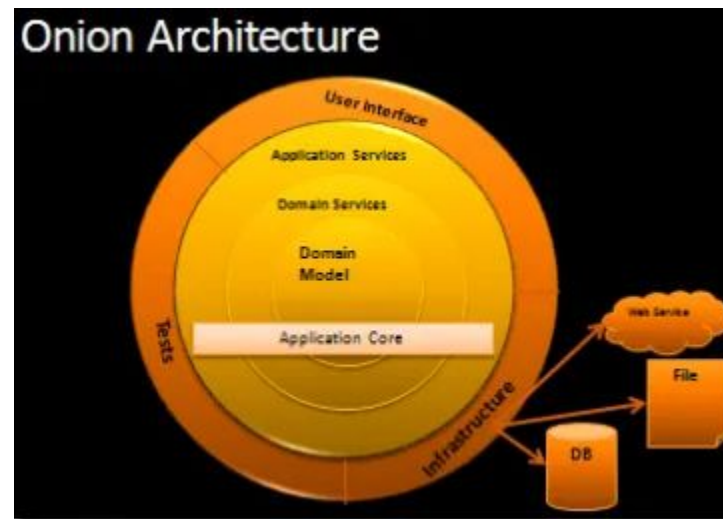
Software Architecture

Common web application architectures

Onion Architecture

(Arquitetura Cebola)

Leitura: jeffreypalermo.com/2008/07/the-onion-architecture-part-1



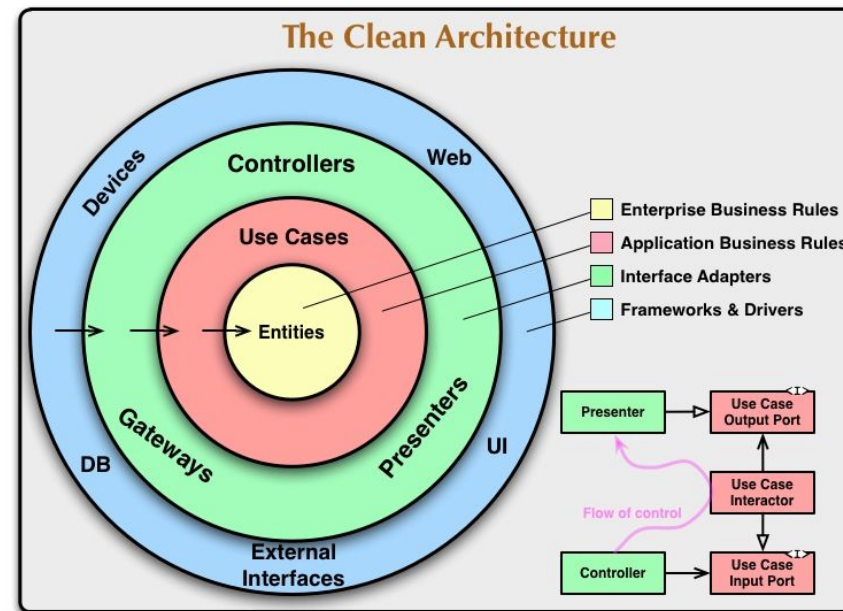
Software Architecture

Common web application architectures

Clean Architecture

Arquitetura Limpa

Leitura: blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

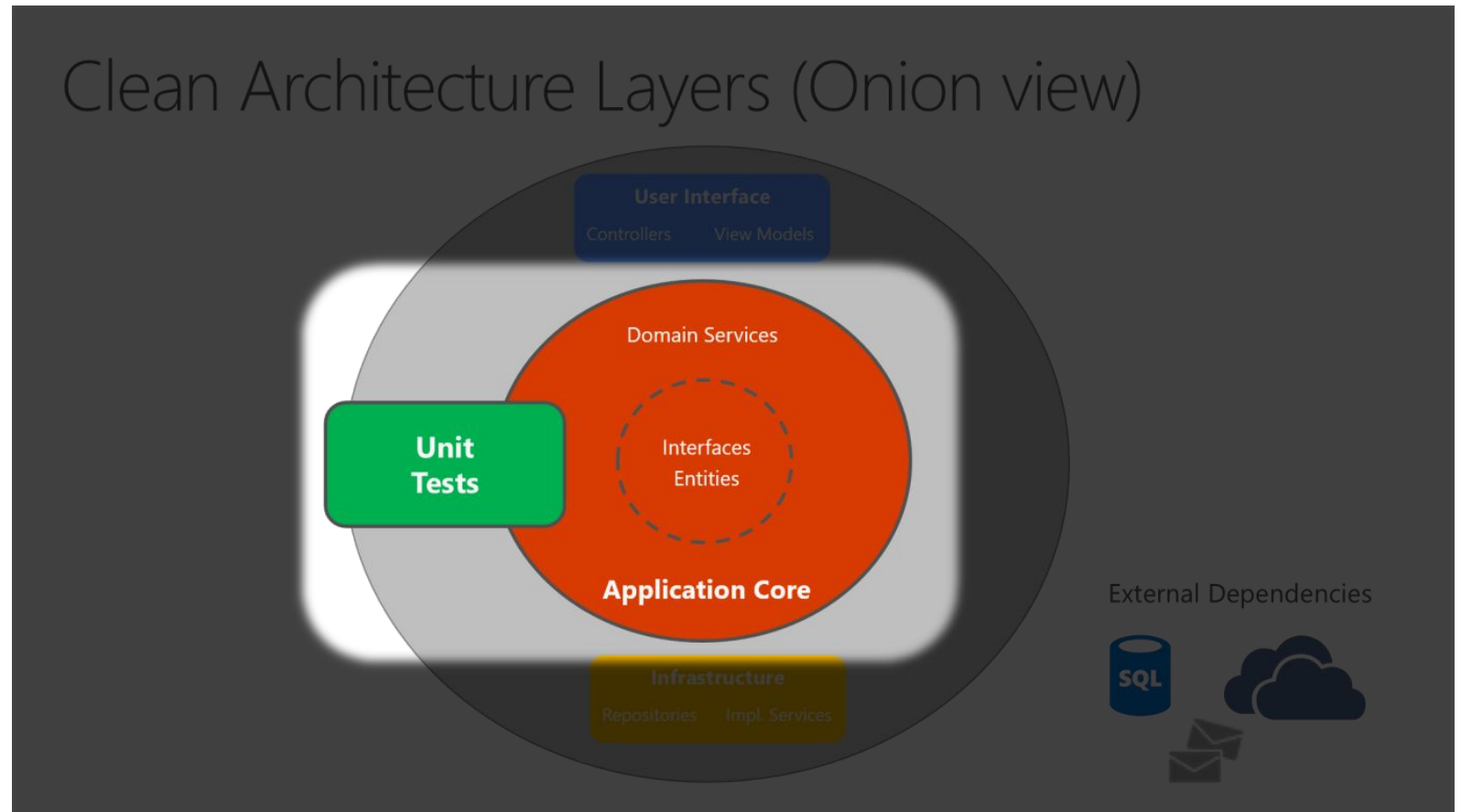


Baseada no Domain-Driven Design (DDD) separa as responsabilidades nas camadas de “Enterprise Business Rules” (**Entities**), “Application Business Rules” (**Use Cases**), “Interface Adapters” (**Controllers (UI)**, **Presenters (UI)** e **Gateways (DB)**) e “Frameworks & Drivers” (**UI**, **DB** e **Externals Agents**)

Software Architecture

Common web application architectures

Unit Test Architecture

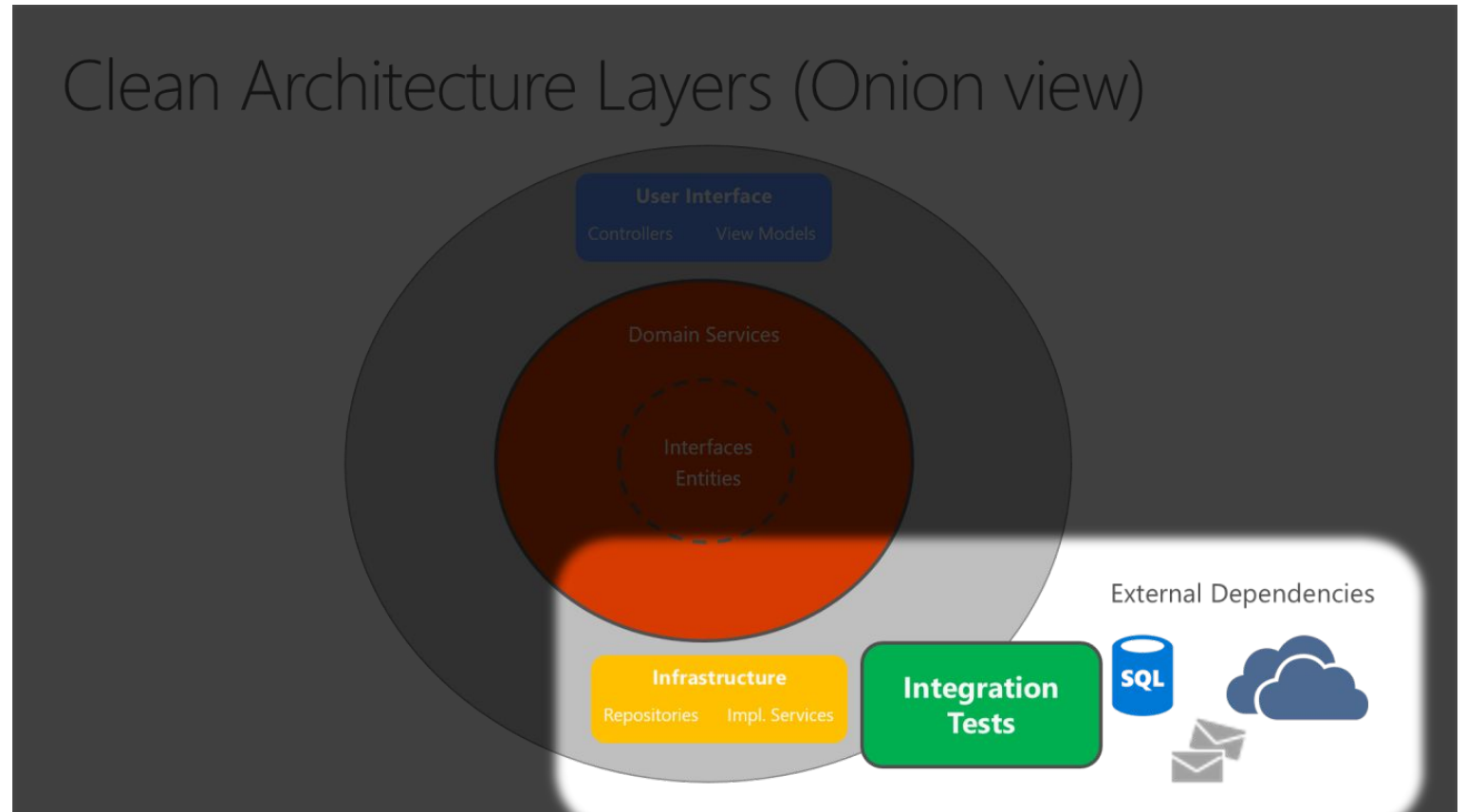


Com as arquiteturas baseadas no **Domain-Driven Design (DDD)** conseguimos isolar os **Testes Unitários** somente para as camadas que contém **Regra de Negócio: "Application" e "Domain"**. A Regra de negócio não deve e não depende dos componentes de UI, Database ou Agentes Externos!

Software Architecture

Common web application architectures

Integration Test Architecture



Com as arquiteturas baseadas no **Domain-Driven Design (DDD)** conseguimos isolar os **Testes de Integração** somente para as **camadas de comunicação externa: "UI" e "Infrastructure"** (Banco de Dados e Agentes Externos).

ArchUnit .NET

Architectural Unit Testing



ArchUnit .NET é uma biblioteca simples e gratuita para verificar a arquitetura do código em C#. É o fork do ArchUnit Java.

Com ela podemos verificar dependências entre classes, membros, interfaces e muito mais.

Seu foco principal é testar automaticamente a Arquitetura e as Regras de Codificação.

ArchUnit .NET


Architectural Unit Testing

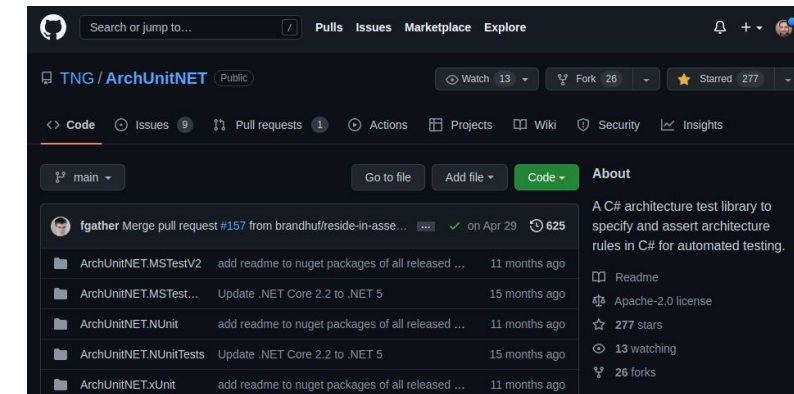
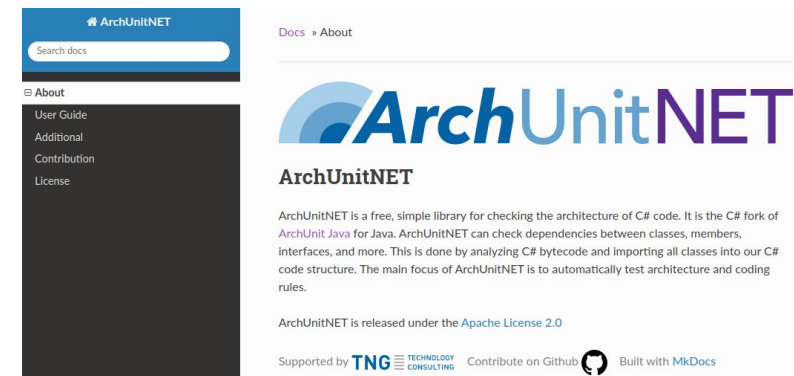
Referências

- **Docs:** archunitnet.readthedocs.io/en/latest
- **Git**Hub: github.com/TNG/ArchUnitNET
- **NuGet:** www.nuget.org/profiles/fgather



fgather

-  **TngTech.ArchUnitNET** by: [fgather](#)
↓ 415,233 total downloads · ⌚ last updated 3 months ago · 📦 Latest version: 0.10.1
🔗 [test arch archunit xunit nunit](#)
C# Version of ArchUnit (see: archunit.org)
-  **TngTech.ArchUnitNET.xUnit** by: [fgather](#)
↓ 214,802 total downloads · ⌚ last updated 3 months ago · 📦 Latest version: 0.10.1 · 🔗 [tes](#)
xUnit Extension for the C# Version of ArchUnit (see: archunit.org)
-  **TngTech.ArchUnitNET.NUnit** by: [fgather](#)
↓ 157,096 total downloads · ⌚ last updated 3 months ago · 📦 Latest version: 0.10.1 · 🔗 [tes](#)
NUnit Extension for the C# Version of ArchUnit (see: archunit.org)
-  **TngTech.ArchUnitNET.MSTestV2** by: [fgather](#)
↓ 4,671 total downloads · ⌚ last updated 3 months ago · 📦 Latest version: 0.10.1



Por onde começar?

Definição de regras, camadas e componentes a serem avaliados

Componentes:

- Em nosso exemplo estaremos avaliando as camadas “Application”, “Domain”, “Infrastructure” e “UI” e os componentes “Controller”, “Service” e “Repository”.

Regras:

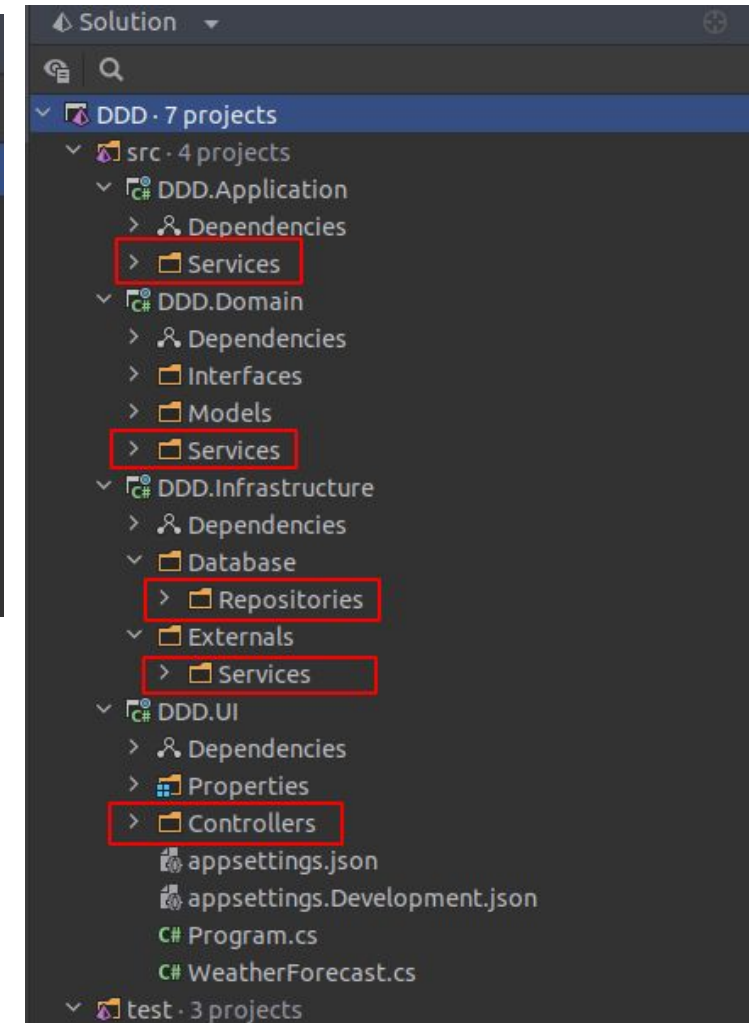
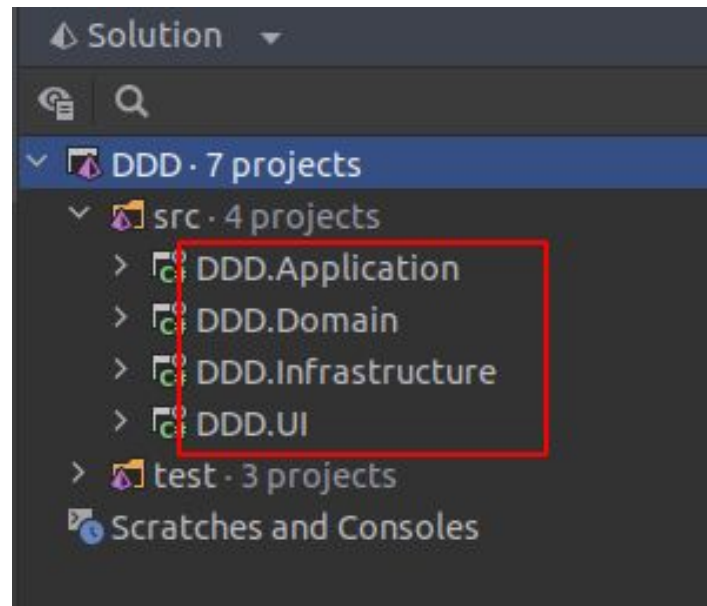
- **Nomenclatura** de Classes e Interfaces;
- **Namespace**;
- **Dependência** entre Camadas;
- **Injeção de Dependência**.

Camadas e Componentes

Domain-Driven Design (DDD)

ArchUnit .NET

Architectural Unit Testing

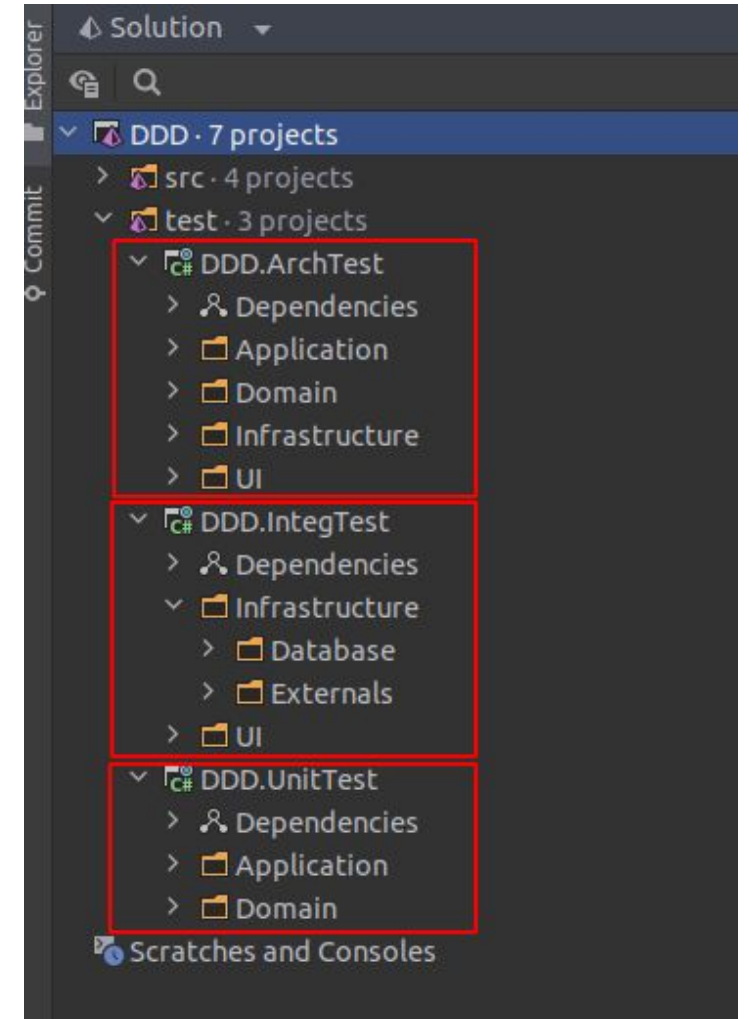
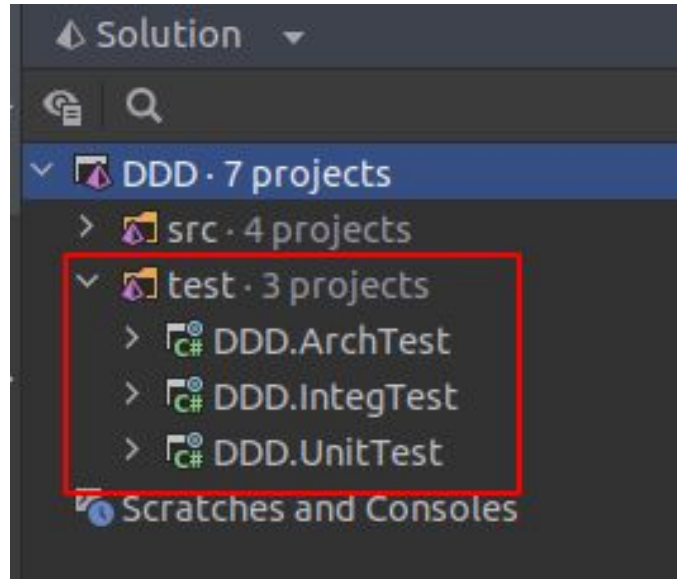


ArchUnit .NET

Architectural Unit Testing

Testes e Camadas

Domain-Driven Design (DDD)

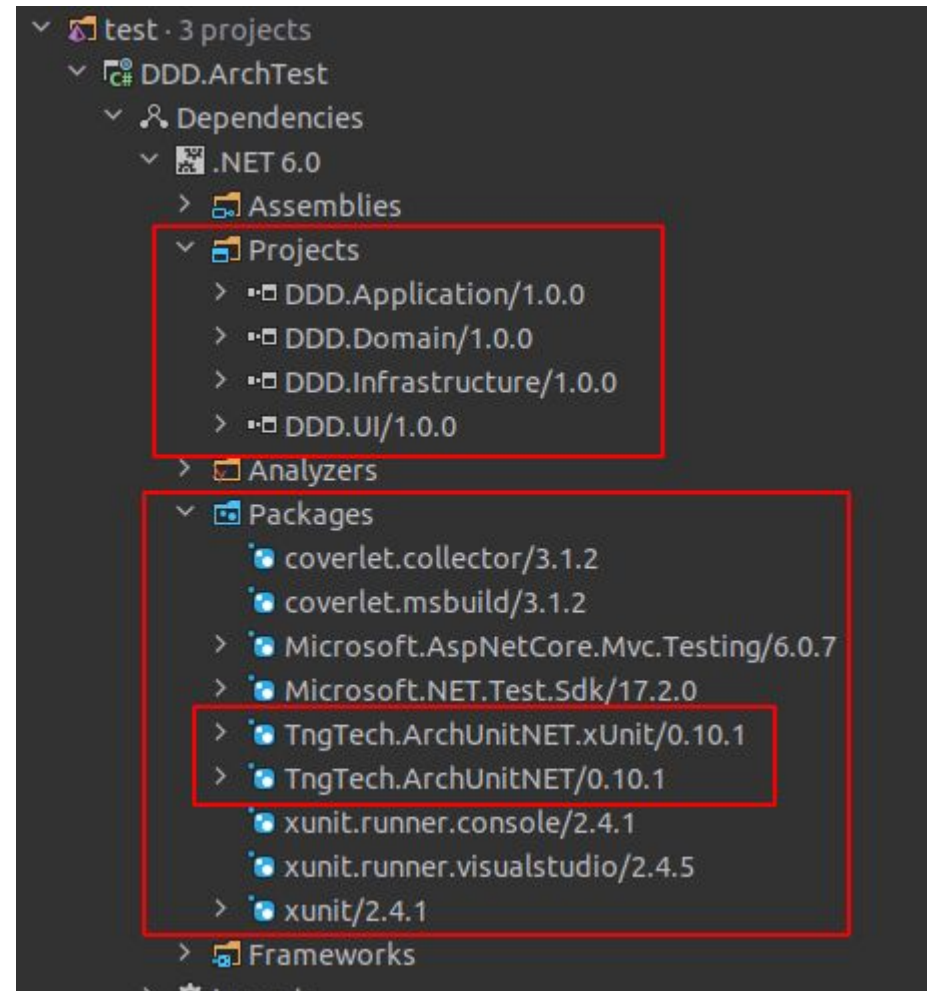


ArchUnit .NET

Architectural Unit Testing

ArchUnit Libraries

NuGet Packages

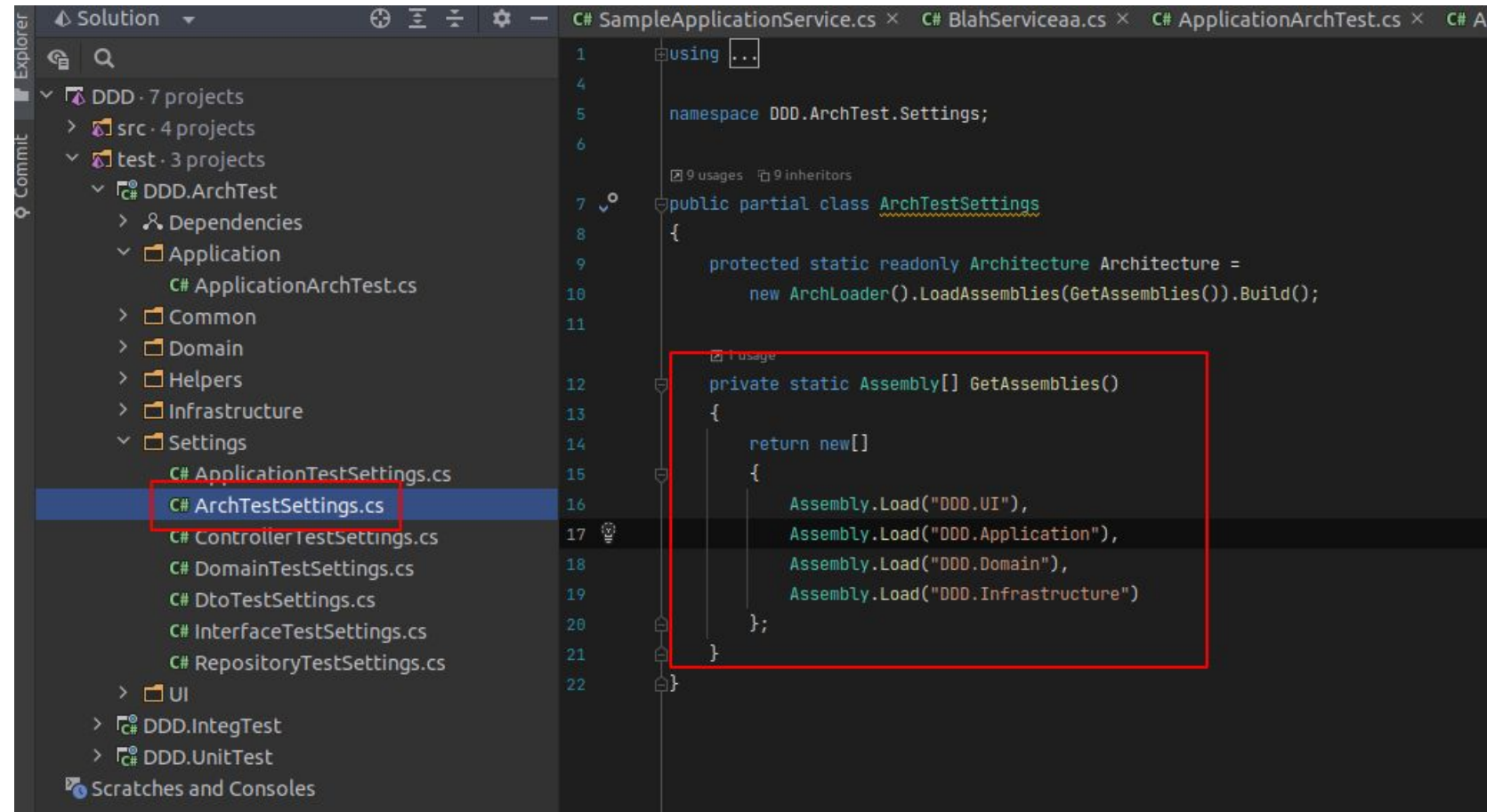


ArchUnit .NET

Architectural Unit Testing

ArchUnit Settings

Definição da Arquitetura

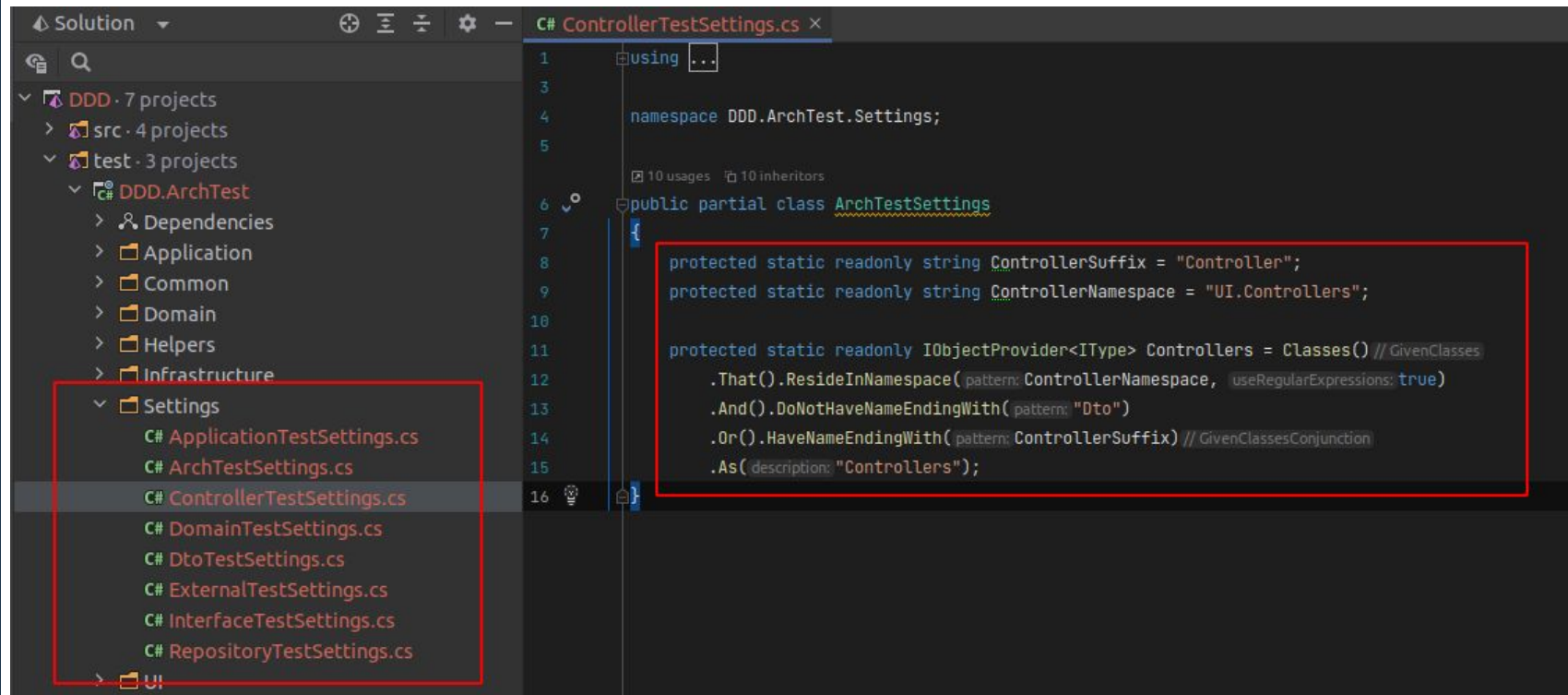


ArchUnit .NET

Architectural Unit Testing

ArchUnit Settings

Definição dos Componentes

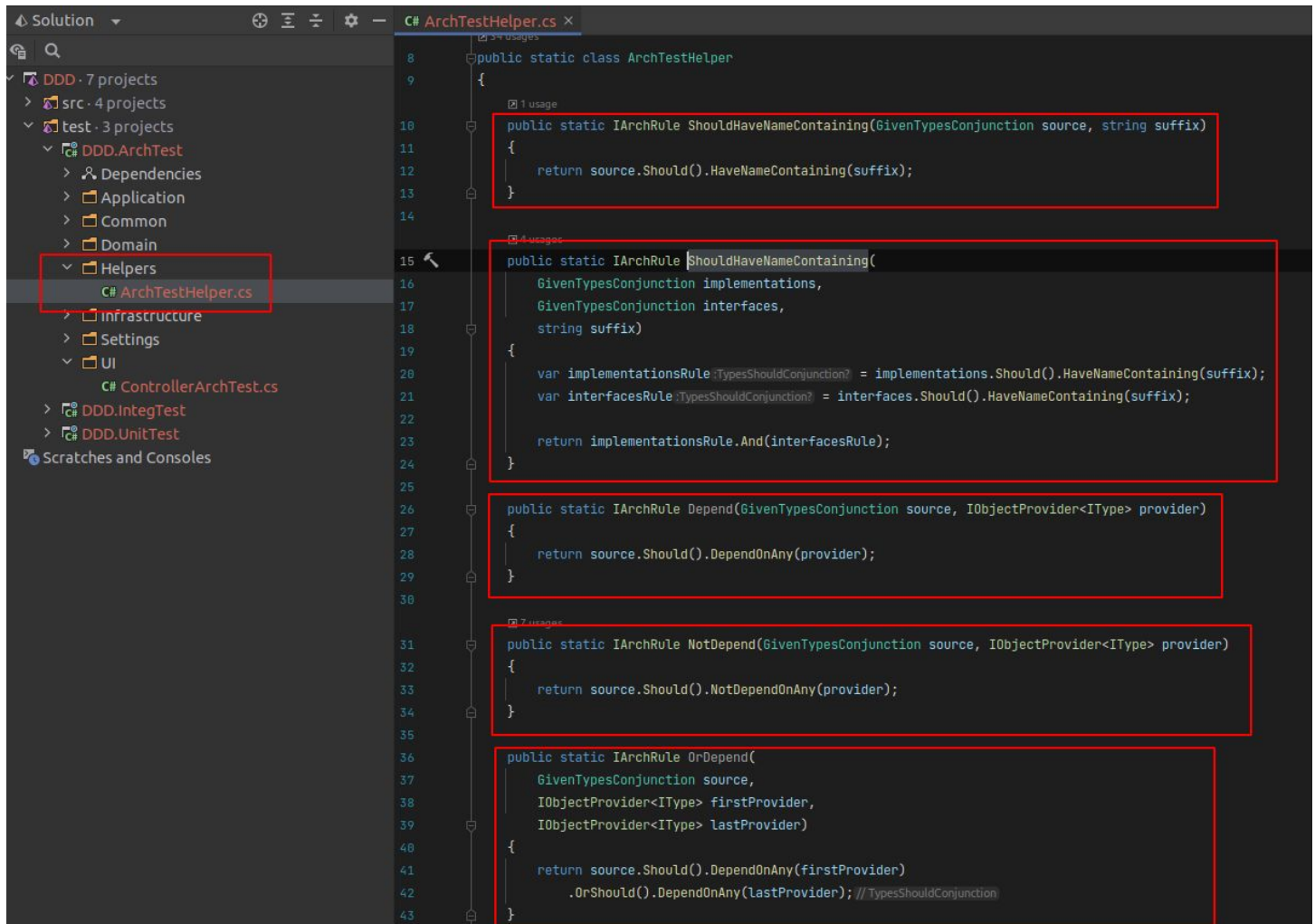


ArchUnit .NET

Architectural Unit Testing

ArchUnit Rules

Definição das Regras



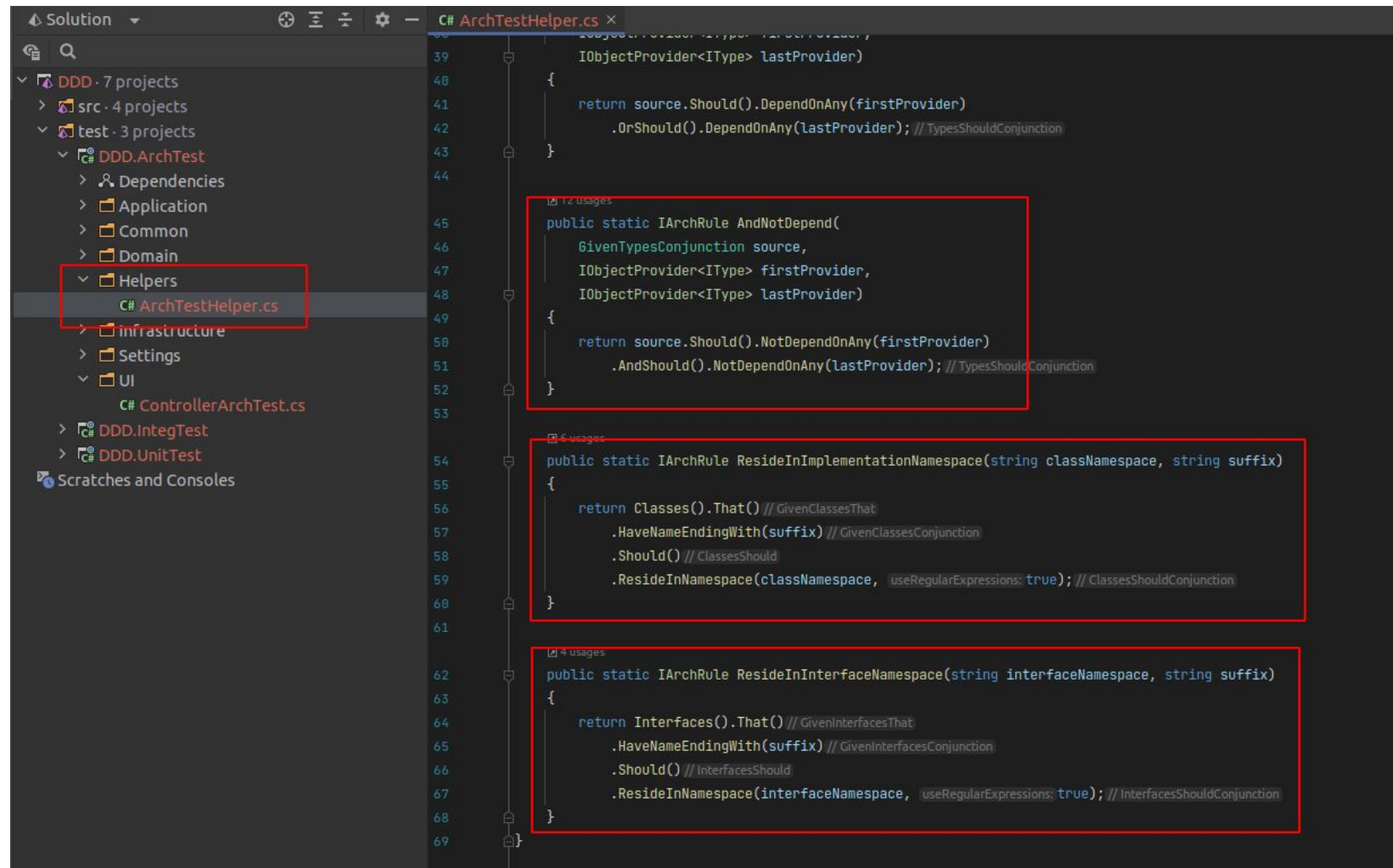
```
8 public static class ArchTestHelper
9 {
10     public static IArchRule ShouldHaveNameContaining(GivenTypesConjunction source, string suffix)
11     {
12         return source.Should().HaveNameContaining(suffix);
13     }
14
15     public static IArchRule ShouldHaveNameContaining(
16         GivenTypesConjunction implementations,
17         GivenTypesConjunction interfaces,
18         string suffix)
19     {
20         var implementationsRule :TypesShouldConjunction? = implementations.Should().HaveNameContaining(suffix);
21         var interfacesRule :TypesShouldConjunction? = interfaces.Should().HaveNameContaining(suffix);
22
23         return implementationsRule.And(interfacesRule);
24     }
25
26     public static IArchRule Depend(GivenTypesConjunction source, IObjectProvider<IType> provider)
27     {
28         return source.Should().DependOnAny(provider);
29     }
30
31     public static IArchRule NotDepend(GivenTypesConjunction source, IObjectProvider<IType> provider)
32     {
33         return source.Should().NotDependOnAny(provider);
34     }
35
36     public static IArchRule OrDepend(
37         GivenTypesConjunction source,
38         IObjectProvider<IType> firstProvider,
39         IObjectProvider<IType> lastProvider)
40     {
41         return source.Should().DependOnAny(firstProvider)
42             .OrShould().DependOnAny(lastProvider); // TypesShouldConjunction
43     }
```


ArchUnit .NET

Architectural Unit Testing

ArchUnit Rules

Definição das Regras



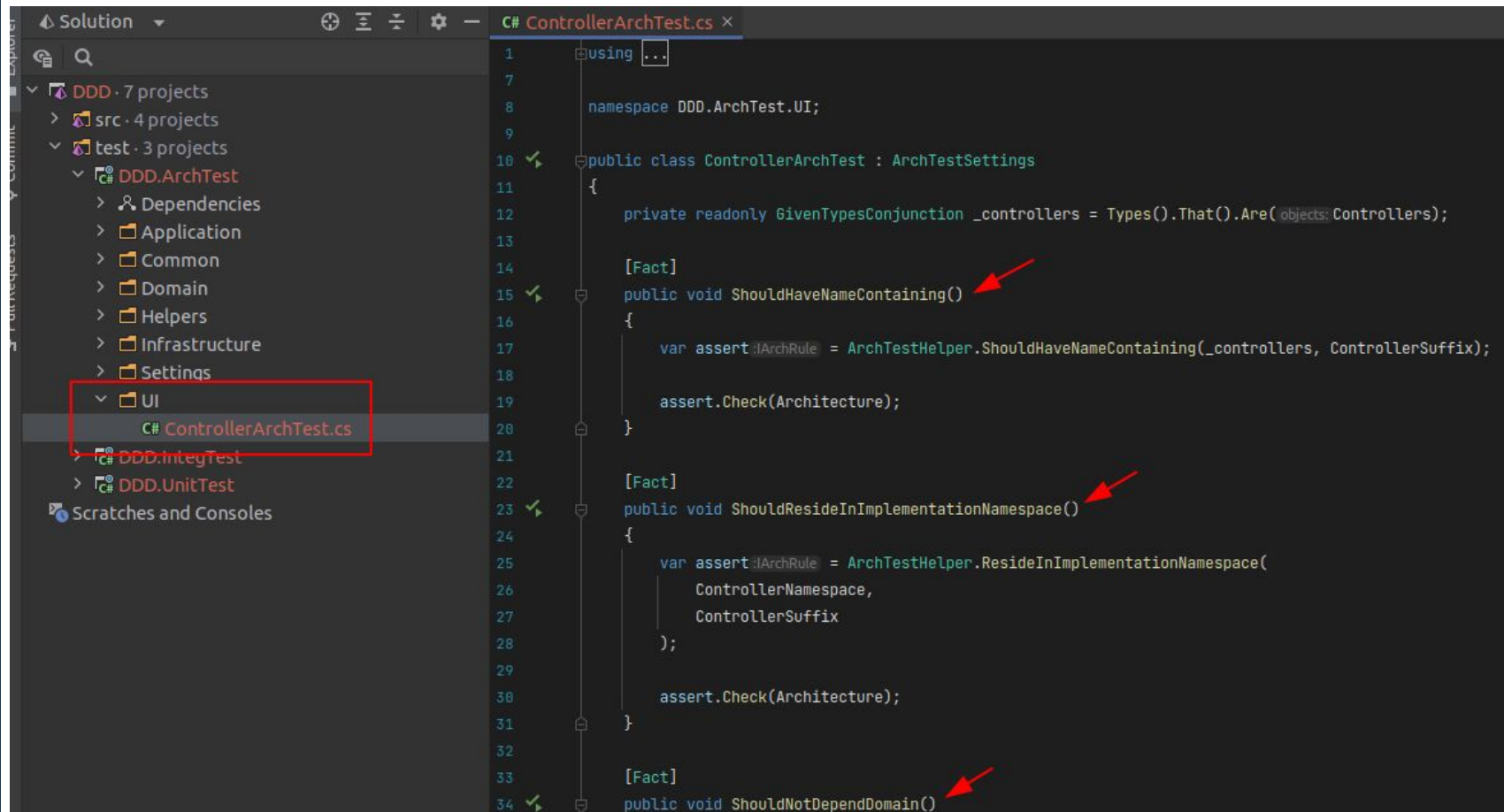
```
39  IObjectProvider<IType> lastProvider)
40  {
41      return source.Should().DependOnAny(firstProvider)
42          .OrShould().DependOnAny(lastProvider); // TypesShouldConjunction
43  }
44
45  public static IArchRule AndNotDepend(
46      GivenTypesConjunction source,
47      IObjectProvider<IType> firstProvider,
48      IObjectProvider<IType> lastProvider)
49  {
50      return source.Should().NotDependOnAny(firstProvider)
51          .AndShould().NotDependOnAny(lastProvider); // TypesShouldConjunction
52  }
53
54  public static IArchRule ResideInImplementationNamespace(string classNamespace, string suffix)
55  {
56      return Classes().That() // GivenClassesThat
57          .HaveNameEndingWith(suffix) // GivenClassesConjunction
58          .Should() // ClassesShould
59          .ResideInNamespace(classNamespace, useRegularExpressions: true); // ClassesShouldConjunction
60  }
61
62  public static IArchRule ResideInInterfaceNamespace(string interfaceNamespace, string suffix)
63  {
64      return Interfaces().That() // GivenInterfacesThat
65          .HaveNameEndingWith(suffix) // GivenInterfacesConjunction
66          .Should() // InterfacesShould
67          .ResideInNamespace(interfaceNamespace, useRegularExpressions: true); // InterfacesShouldConjunction
68  }
69  }
```

ArchUnit .NET

Architectural Unit Testing

ArchUnit Tests

Definição dos Testes



The screenshot shows the Visual Studio IDE. On the left, the Solution Explorer displays a project structure for 'DDD'. Under the 'test' folder, there are three projects: 'DDD.ArchTest', 'DDD.IntegTest', and 'DDD.UnitTest'. The 'DDD.ArchTest' project is expanded, showing subfolders: 'Dependencies', 'Application', 'Common', 'Domain', 'Helpers', 'Infrastructure', 'Settings', and 'UI'. The 'UI' folder is selected, and the file 'ControllerArchTest.cs' is highlighted. On the right, the code editor shows the content of 'ControllerArchTest.cs'. The code defines a class 'ControllerArchTest' that inherits from 'ArchTestSettings'. It includes three test methods: 'ShouldHaveNameContaining()', 'ShouldResideInImplementationNamespace()', and 'ShouldNotDependDomain()'. Red arrows point to the method signatures of 'ShouldHaveNameContaining()' and 'ShouldResideInImplementationNamespace()'.

```
1  using ...
2
3  namespace DDD.ArchTest.UI;
4
5  public class ControllerArchTest : ArchTestSettings
6  {
7      private readonly GivenTypesConjunction _controllers = Types().That().Are(objects: Controllers);
8
9      [Fact]
10     public void ShouldHaveNameContaining()
11     {
12         var assert : ArchRule = ArchTestHelper.ShouldHaveNameContaining(_controllers, ControllerSuffix);
13         assert.Check(Architecture);
14     }
15
16     [Fact]
17     public void ShouldResideInImplementationNamespace()
18     {
19         var assert : ArchRule = ArchTestHelper.ResideInImplementationNamespace(
20             ControllerNamespace,
21             ControllerSuffix
22         );
23         assert.Check(Architecture);
24     }
25
26     [Fact]
27     public void ShouldNotDependDomain()
```

ArchUnit .NET

Architectural Unit Testing

ArchUnit Tests

Definição dos Testes

```
[Fact]
public void ShouldHaveNameContaining()
{
    var assert :IArchRule = ArchTestHelper.ShouldHaveNameContaining(
        // ...

    assert.Check(Architecture);
}
```

```
[Fact]
public void ShouldNotDependDomain()
{
    var assert :IArchRule = ArchTestHelper.AndNotDepend(
        _controllers,
        firstProvider: IDomainServices,
        lastProvider: DomainServices
    );

    assert.Check(Architecture);
}
```

```
[Fact]
public void ShouldNotDependRepository()
{
    var assert :IArchRule = ArchTestHelper.AndNotDepend(
        _controllers,
        firstProvider: IRepositories,
        lastProvider: Repositories
    );

    assert.Check(Architecture);
}
```

```
[Fact]
public void ShouldResideInImplementationNamespace()
{
    var assert :IArchRule = ArchTestHelper.ResideInImplementationNamespace(
        ControllerNamespace,
        ControllerSuffix
    );

    assert.Check(Architecture);
}
```

```
[Fact]
public void ShouldNotDependExternals()
{
    var assert :IArchRule = ArchTestHelper.AndNotDepend(
        _controllers,
        firstProvider: IExternalServices,
        lastProvider: ExternalServices
    );

    assert.Check(Architecture);
}
```

```
[Fact]
public void ShouldNotDependApplicationImplementations()
{
    var assert :IArchRule = ArchTestHelper.NotDepend(_controllers,
        // ...

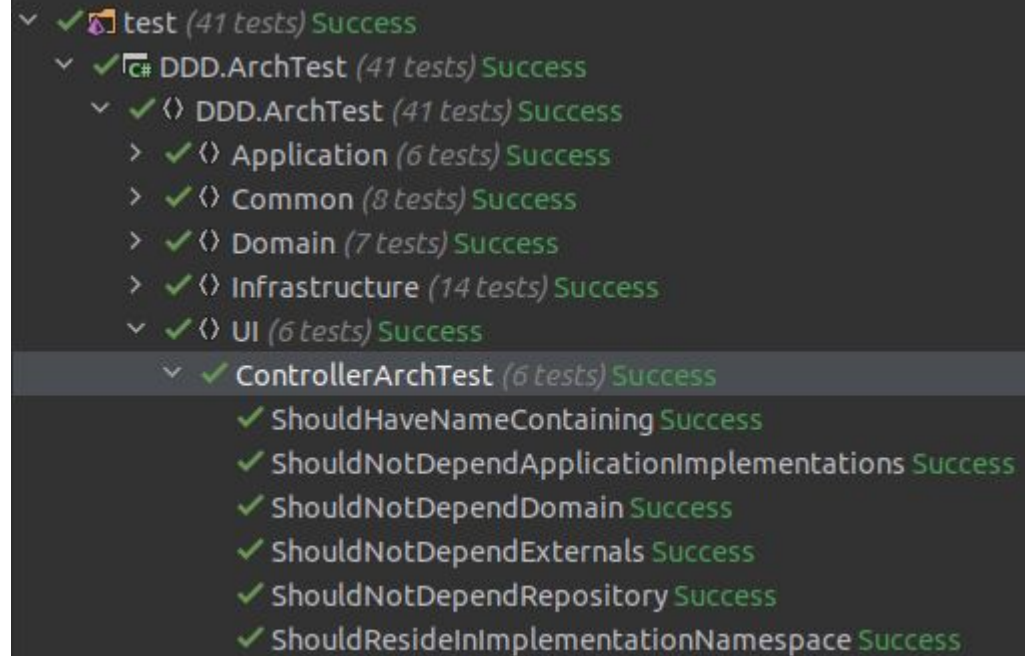
    assert.Check(Architecture);
}
```

ArchUnit .NET

Architectural Unit Testing

ArchUnit Run Tests

Caso de Sucesso



```

✓ test (41 tests) Success
  ✓ DDD.ArchTest (41 tests) Success
    ✓ {} DDD.ArchTest (41 tests) Success
      > ✓ {} Application (6 tests) Success
      > ✓ {} Common (8 tests) Success
      > ✓ {} Domain (7 tests) Success
      > ✓ {} Infrastructure (14 tests) Success
      > ✓ {} UI (6 tests) Success
        ✓ ControllerArchTest (6 tests) Success
          ✓ ShouldHaveNameContaining Success
          ✓ ShouldNotDependApplicationImplementations Success
          ✓ ShouldNotDependDomain Success
          ✓ ShouldNotDependExternals Success
          ✓ ShouldNotDependRepository Success
          ✓ ShouldResideInImplementationNamespace Success

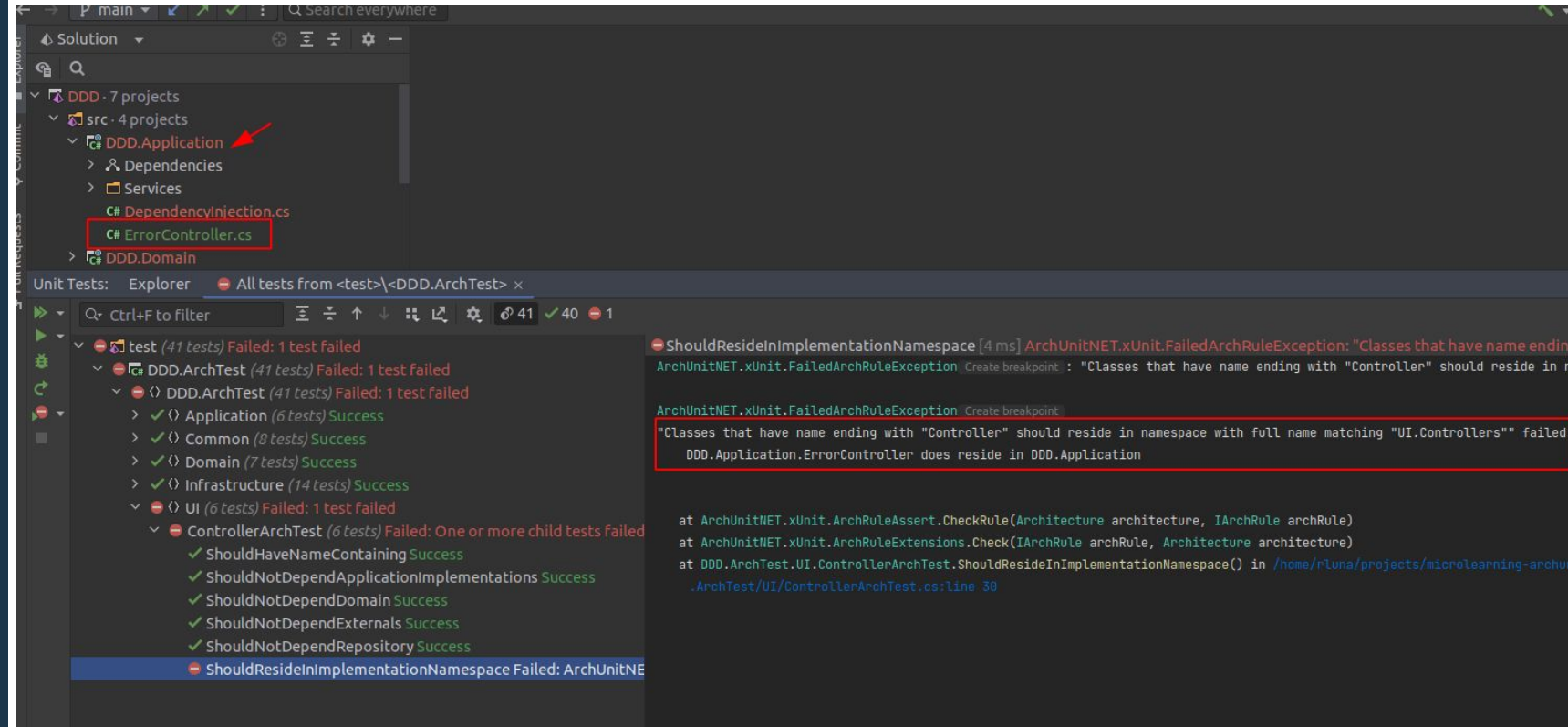
```


ArchUnit Run Tests

Caso de Falha

ArchUnit .NET

Architectural Unit Testing



ArchUnit .NET

Architectural Unit Testing

That's all folks!

LinkedIn:

- linkedin.com/in/ricardo-galdino

GitHub:

- github.com/ricardogaldino

WhatsApp Group:

- archsoft.com.br

github.com/ricardogaldino/microlearning-archunit-dotnet