Aplicando a governança de arquitetura de software para projetos .net usando testes de unidade

- Postagem publicada:5 de julho de 2020

As discussões sobre a arquitetura de software são a fase mais importante em qualquer projeto de engenharia de software empresarial. A maioria das decisões de arquitetura de software são documentadas no início de um projeto. Essa documentação é vital para a equipe de desenvolvimento de software, pois compartilha uma linguagem comum e um entendimento de como os componentes interagem entre si.

Normalmente, arquitetos de software e/ou líderes técnicos são responsáveis por garantir que a equipe de desenvolvimento cumpra as decisões de arquitetura por meio de

- envolvendo a equipe de desenvolvimento em discussões de arquitetura
- manter um registro de cada decisão arquitetônica
- mantendo registros de decisão arquitetural
- revisar projetos de software,
- implementar e manter analisadores de código estáticos e
- realizando revisões intensivas de código.

Hoje, a maioria das metodologias ágeis está incentivando o desenvolvimento rápido de software para implementar ou aprimorar recursos o mais rápido possível. Como resultado, o projeto cresce continuamente com uma base de código cada vez maior, forçando arquitetos e/ou líderes técnicos a ignorar linhas de código-fonte ao realizar revisões de código. Além disso, os documentos de arquitetura de software não são atualizados continuamente e esses documentos parecem ser esquecidos à medida que a base de código aumenta e os desenvolvedores vêm e vão. Com o tempo, a governança em torno da arquitetura de software é reduzida.

O teste de unidade de arquitetura é uma maneira de impor a governança das decisões de arquitetura. Esse tipo de teste de unidade se concentra em testar a base de código no contexto da arquitetura de software e pode ser facilmente automatizado em um pipeline de compilação.

O teste de unidade de arquitetura fornece a seguinte verificação de governança de arquitetura automatizada

- dependências entre projetos
- dependências entre classes, membros e interfaces
- dependências entre camadas
- dependências cíclicas e muito mais.

Duas bibliotecas comuns de código aberto que podem ser usadas para escrever testes de arquitetura específicos de aplicativos em .net são

- NetArchTest e
- ArchUnitNet .

Começando

Leia o artigo sobre arquitetura onion localizado aqui, pois o restante deste artigo se concentrará em escrever alguns testes de unidade de arquitetura para garantir que a arquitetura onion seja respeitada. Os exemplos abaixo usam NetArchTest com xUnit .

NetArchTest pode ser instalado usando nuget

```
Install-Package NetArchTest.Rules
```

Exemplo

A camada de domínio está no centro da arquitetura onion . Como todo acoplamento em uma arquitetura onion é voltado para o centro, o domínio é acoplado apenas a si mesmo.

Assim, no núcleo da aplicação, a camada de domínio não deve referenciar a camada de repositórios e a camada de serviços. Usando NetArchTest, dois testes de unidade de arquitetura podem ser escritos. Considere o teste abaixo que verifica se a camada de domínio faz referência à camada de repositórios.

```
[Fact]
public void Architecture_DomainLayer_ShouldNotHaveReferenceToRepositoriesLayer()
{
    Types types = Types.InAssembly(typeof(ICoreModule).Assembly);

    bool actualResult = types
        .That()
        .ResideInNamespace("Demo.Core.Domain")
        .ShouldNot()
        .HaveDependencyOn("Demo.Core.Repositories")
        .GetResult()
        .IsSuccessful;

    actualResult.Should().BeTrue();
}
```

DomainTests.cs hosted with ❤ by **GitHub** **view raw**

Como pode ser visto na base de código acima, o NetArchTest fornece uma API fluente muito simples para escrever testes de unidade de arquitetura. Da mesma forma, um teste para verificar se a camada de domínio não faz referência à camada de serviços pode ser escrito da seguinte maneira.

```
[Fact]
public void Architecture_DomainLayer_ShouldNotHaveReferenceToServicesLayer()
{
    Types types = Types.InAssembly(typeof(ICoreModule).Assembly);

    bool actualResult = types
        .That()
        .ResideInNamespace("Demo.Core.Domain")
        .ShouldNot()
        .HaveDependencyOn("Demo.Core.Services")
        .GetResult()
        .IsSuccessful;

    actualResult.Should().BeTrue();
}
```

DomainTests.cs hosted with ❤ by **GitHub** **view raw**

Os testes de unidade de arquitetura acima são considerados regras e são testados independentemente. O NetArchTest também oferece a capacidade de agrupar regras em políticas. Considere a política abaixo para a camada de domínio.

```
[Fact]
public void Architecture_DomainLayer_ShouldAdhereToTheOnionArchitectureDomainLayerPolicy()
{
    PolicyDefinition domainLayerPolicy = Policy
        .Define("Onion Architecture Domain Layer Policy", "The domain layer should not have
        .For(Types.InAssembly(typeof(ICoreModule).Assembly))
        .Add(t => t.That()
                   .ResideInNamespace("Demo.Core.Domain")
                   .ShouldNot()
                   .HaveDependencyOn("Demo.Core.Repositories"),
             "Enforcing Domain Layer Policy", "Domain layer should not have references to t
        )
        .Add(t => t.That()
                   .ResideInNamespace("Demo.Core.Domain")
                   .ShouldNot()
                   .HaveDependencyOn("Demo.Core.Services"),
             "Enforcing Domain Layer Policy", "Domain layer should not have references to t
        );

    bool actualResult = domainLayerPolicy.Evaluate().HasViolations;

    actualResult.Should().BeFalse();
}
```

DomainLayerPolicy.cs hosted with ❤ by **GitHub** **view raw**

As políticas são uma maneira eficaz de agrupar regras. Depois de executar uma política, o resultado de cada regra com falha também pode ser gerado em um teste. Um exemplo disso pode ser visto no exemplo NetArchTest .

Resumo

Impor a governança da arquitetura de software é um desafio na maioria dos projetos de engenharia de software empresarial. Para tornar esse processo mais robusto, o teste de unidade de arquitetura pode ser implementado como uma possível solução para impor a governança das decisões de arquitetura.

Por ter testes de arquitetura automáticos, o

- arquitetura pode evoluir à medida que o projeto cresce,
- arquitetos/desenvolvedores podem garantir que novos componentes/recursos aderem à arquitetura de software e
- os desenvolvedores são incentivados a entender e contribuir para as decisões de arquitetura.

Graças à comunidade de código aberto que trabalha duro, bibliotecas como NetArchTest e ArchUnitNet existem para facilitar a criação e manutenção de testes de unidade de arquitetura, reforçando e automatizando a governança de arquitetura de software.

O código-fonte usado neste artigo pode ser encontrado aqui .

Mais informações sobre testes unitários de arquitetura podem ser encontradas nos seguintes links:

- Escrevendo testes de unidade de arco para dotnet
- Teste NetArch

- ArchUnitNET
- ArchUnit

Versões de software usadas neste artigo

- Jetbrains Rider v2020.1.3
- NetArchTest.Rules v1.2.5
- xUnit v2.4.0
- FluentAssertions v5.10.3