Teste de unidade de arquitetura

- Postado por:
- travis
- Sem comentários

Eu vou assumir que você já comprou as vantagens do teste de unidade do seu código , e os méritos de fazer isso não precisam ser enumerados em outro artigo. Talvez mais interessante e único nas práticas de desenvolvimento de software de hoje seja a ideia de testar sua arquitetura de unidade!

A arquitetura em si tem MUITAS definições e pode significar algo ligeiramente diferente para cada pessoa (ou seja, “as coisas difíceis”). Nesse contexto, quando me refiro à *arquitetura de teste de unidade* , quero me referir às expectativas e contratos comuns de quais módulos ou projetos podem referenciar e usar. Ainda mais especificamente, um projeto pode depender de certas convenções de nomenclatura acordadas de objetos e classes ou interfaces. Muitas vezes, esses tipos de discussões e padrões são comunicados entre os líderes da equipe e os padrões de design iniciais colocados no início do projeto. Muito rapidamente o projeto cresce, evolui e outros colaboradores começam a trabalhar em um projeto, e esses requisitos iniciais de “arquitetura” não codificados são rapidamente esquecidos e você perde sua pureza arquitetônica.

Pureza arquitetônica ... por que me importo com isso? O projeto ainda parece funcional e a equipe continua lançando diariamente. Talvez pressionar por esse gráfico de dependência limpo entre os módulos tenha sido uma perda de tempo em primeiro lugar? Se você está fazendo essas perguntas, seu projeto provavelmente não está maduro o suficiente para sofrer as consequências dessa erosão. Referências circulares, bibliotecas monolíticas compartilhadas e abstrações de design orientadas a domínio zero rapidamente paralisarão seu projeto e tornarão qualquer tipo de refatoração incrivelmente difícil (sem mencionar os tempos de compilação estendidos e esgotados que podem ocorrer).

Na minha experiência, a única maneira de garantir que qualquer requisito (de negócios ou técnico) dure a duração de um projeto é codificá-lo no repositório em algum lugar. O método mais fácil para isso é incluir no “README.md” ou outra documentação relacionada que existe e evolui junto com o código-fonte. Infelizmente, como remarcação, nenhuma dessas informações será aplicada e requer algum pré-conhecimento de sua existência (mas esperamos que seja decentemente autodescoberta pela equipe). É claro que o outro método requer um pouco mais de trabalho, mas se você puder começar a codificar essas regras como testes de unidade, poderá começar a aplicar os requisitos de arquitetura sem que um engenheiro tenha qualquer conhecimento prévio sobre isso. Embora você possa conseguir isso usando seu próprio código e o Reflectionbibliotecas, parece muito feio e insustentável. É aí que entra um framework como o ArchUnit .

ArchUnit (Java)

ArchUnit é uma biblioteca Java que você pode adicionar aos seus testes de unidade que fornece um método de API muito fluente para validar algumas das restrições arquitetônicas que você pode querer adicionar a um projeto.

O ArchUnit permite que você seja bastante específico para uma classe específica ou mais ampla em um determinado pacote/importação. Como exemplo , você pode facilmente procurar todas as classes em um determinado pacote com uma anotação específica para validar que o nome é prefixado com um valor (como “Serviço”):

```
@ArchTest
public static ArchRule services_should_be_prefixed =
  Aulas()
    .that().resideInAPackage("..service..")
    .and().areAnnotatedWith(MyService.class)
    .should().haveSimpleNameStartingWith("Service");
```

As possibilidades do que essa API permitirá que você teste são infinitas. Existem alguns ótimos projetos de exemplo para analisar e revisar. Para citar alguns dos cenários:

- **Dependências de módulo** – teste camadas adequadas de referências de módulo, procure especificamente por dependências cíclicas, identifique módulos que nunca devem fazer referência uns aos outros.
- **Limitar a dependência externa** – impõe que um projeto use uma determinada dependência ou não use uma determinada dependência.
- **Convenções de nomenclatura** – teste para convenções de nomenclatura de serviços e controladores. Além disso, aplique regras mais amplas para nomenclatura de variáveis privadas ou até mesmo nomenclatura de módulos.
- **Convenções de codificação** – teste se os objetos são instanciados de uma certa maneira, a injeção de dependência é aplicada, etc.
- **DAO** – faça com que todos os acessos de dados aos seus bancos de dados aconteçam em um determinado módulo e que o DAO seja referenciado apenas por determinadas implementações de nível de serviço.

Ótimo artigo da Java Magazine no ArchUnit para uma visão mais aprofundada.

Teste NetArch (.NET)

Inspirado nos testes de estilo ArchUnit , o projeto NetArchTest permite testar suas restrições de arquitetura em .NET. Muitas das equipes em que trabalho são baseadas em Java ou .NET, e essas duas bibliotecas fornecem paridade de recursos e capacidade muito semelhantes para criar testes de unidade de arquitetura. O NetArchTest também é totalmente fluente e você se sentirá em casa.

Um exemplo simples do GitHub que mostra como você pode validar que as referências entre apresentação ou camada de interface do usuário não se vinculam diretamente aos repositórios de acesso a dados:

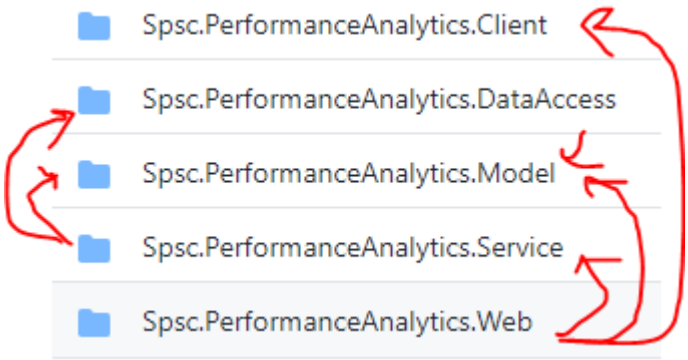
```
// As classes na apresentação não devem referenciar diretamente os repositórios
var resultado = Types.InCurrentDomain()
  .Que()
  .ResideInNamespace("NetArchTest.SampleLibrary.Presentation")
  .Não deveria()
  .HaveDependencyOn("NetArchTest.SampleLibrary.Data")
  .GetResult()
  .É bem-sucedido;
```

Embora eu tenha tido um bom sucesso com a biblioteca NetArchTest, você deve estar ciente de que ArchUnit tem uma biblioteca oficial agora que suporta .NET, ArchUnitNET . Licenças um pouco mais novas e diferentes na biblioteca, definitivamente considere usar ArchUnitNET se for uma vantagem ter uma interface semelhante entre seus projetos Java e .NET. Dito isto, não tive a oportunidade de explorar o ArchUnitNET em profundidade… deixe alguns comentários se você já experimentou e como ele se compara!

Exemplos

Veja a seguir um exemplo de como uma de nossas equipes no SPS usa os testes de unidade de arquitetura para inicializar um novo serviço no .NET Core.

A solução seed do .NET Core contém 5 projetos para uma arquitetura em camadas, cada um com uma finalidade específica e um conjunto muito intencional de dependências de outros projetos.



Como seria de esperar, nosso projeto Web especificamente só tem permissão para fazer referência ao projeto Service (ou seja, não ao DataAccess). Além disso, queremos garantir que a camada de serviço seja totalmente independente do projeto da Web ou da interface do usuário. O que pode parecer um pouco engraçado é que o projeto Web depende do projeto Cliente. O projeto do cliente geralmente é um cliente HTTP de alta fidelidade criado com o Refit, mas eles compartilham os modelos da Web contratuais definidos em um único local no cliente (para evitar a necessidade de criar um pacote NuGet distribuído separado para o modelo compartilhado). Este é um padrão realmente fantástico, mas não tão intuitivo que espero que uma futura postagem no blog possa revelar um pouco mais. Por enquanto, entenda que é essencial que a Web dependa do Cliente, mas que NADA mais pode e o Cliente não pode ter outras referências internas do projeto. Podemos codificar essas regras assim:

```
[Método de teste]
public void ClientDependsOnNothingTest()
{
  var resultado = Tipos
    .InAssembly(typeof(SupplierConfigurationServiceFactory).Assembly)
    .Não deveria()
    .HaveDependencyOnAny(
      "Spsc.PerformanceAnalytics.DataAccess",
      "Spsc.PerformanceAnalytics.Model",
      "Spsc.PerformanceAnalytics.Service",
      "Spsc.PerformanceAnalytics.Web")
    .GetResult();

  Assert.IsTrue(result.IsSuccessful,
    "O Projeto Cliente HTTP não deve fazer referência a nenhum outro projeto.");
}

[Método de teste]
public void ModelDependsOnNothingTest()
{
  var resultado = Tipos
    .InAssembly(typeof(AppSettings).Assembly)
    .Não deveria()
    .HaveDependencyOnAny(
      "Spsc.PerformanceAnalytics.DataAccess",
      "Spsc.PerformanceAnalytics.Service",
      "Spsc.PerformanceAnalytics.Web")
    .GetResult();

  Assert.IsTrue(result.IsSuccessful,
    "Projeto Modelo não deve fazer referência a nenhum outro projeto.");
}

[Método de teste]
public void ServiceAgnosticDependencyTest()
{
  var resultado = Tipos
    .InCurrentDomain()
    .Que()
    .ResideInNamespace("Spsc.PerformanceAnalytics.Service")
    .Não deveria()
    .HaveDependencyOnAny(
      "Spsc.PerformanceAnalytics.Web",
```

```
      "Spsc.PerformanceAnalytics.Client"
    )
    .GetResult();

  Assert.IsTrue(result.IsSuccessful,
    "O Projeto de Serviço não deve fazer referência a projetos da Web ou Cliente.");
}

[Método de teste]
public void DataAccessAgnosticDependencyTest()
{
  var resultado = Tipos
    .InCurrentDomain()
    .Que()
    .ResideInNamespace("Spsc.PerformanceAnalytics.DataAccess")
    .Não deveria()
    .HaveDependencyOnAny(
      "Spsc.PerformanceAnalytics.Service",
      "Spsc.PerformanceAnalytics.Web",
      "Spsc.PerformanceAnalytics.Client")
  .GetResult();

  Assert.IsTrue(result.IsSuccessful,
    "O Projeto de Acesso a Dados não deve fazer referência a nenhum outro projeto, exceto
o modelo.");
}
```

Para o arquiteto e a empresa

Como arquiteto , estou constantemente interessado em como as práticas de desenvolvimento de software em torno de codificação, estilo, CI/CD, segurança e SDLC em geral se comportam quando aplicadas no nível corporativo a uma organização. Atualmente, a discussão acima e os padrões no SPS não são algo que é universalmente aplicado a todas as equipes, mas sim adotado por equipes individuais à medida que começam a ver os benefícios.

Do ponto de vista do Arquiteto, é bom poder codificar essas funções de aptidãodiretamente como um projeto começa a garantir sua longevidade em apoio a essa pureza. No exemplo mostrado acima, é bastante simples e não incrivelmente abrangente para a capacidade total do framework. Mas vale a pena perguntar até onde você realmente gostaria de levar o uso desse framework e testes, especialmente em um mundo Microservice ou mesmo TinyService. Em alguns casos, um framework como o ArchUnit é muito mais valioso na organização quando se olha para aplicativos de estilo monolítico ou aplicativos de estilo mono-repo onde tal aplicação é mais um problema.

Como com qualquer coisa relacionada à “Arquitetura”, a questão sobre se e como você deve implementar testes de arquitetura em sua organização chegará a “depende”. Dito isto, parece que há um nicho apropriado para testes de unidade de arquitetura como um decorador para aprimorar ou preencher a lacuna com base no fator de forma arquitetural que você está seguindo.

Seguindo em frente, gostaria de considerar como os testes de unidade de arquitetura podem ser pré-configurados em modelos de serviço organizacional padronizados, modelos de sementes e andaimes para ajudar a incentivar seu uso e o pré-conhecimento de seu uso por outros líderes de equipe e arquitetos.