Testes Unitários

Apresentação do Curso

Apresentação do Curso

- Testes Unitários têm sido cada vez mais cobrados em vagas de diversos níveis, tanto nacionais quanto internacionais
- Não somente isso, mas a sua adoção em projetos de diversos portes traz inúmeros benefícios, como melhor manutenibilidade e assertividade no código escrito
- Por conta disso, aprender testes unitários já virou algo essencial na carreira de desenvolvedores(as) .NET

Apresentação do Curso

- **Neste curso veremos:**
  - O que são Testes Unitários
  - Desafios na Implementação
  - Como testar código ruim
  - Dummies, Mocks, Stubs
  - Padrão de Escrita
  - Fundamentos de xUnit
  - Criando seus primeiros testes com xUnit
  - Fundamentos de NSubstitute com Prática
  - Fundamentos de Moq com Prática
  - Validação com Fluent Assertions com Prática
  - Geração de Dados com Bogus com Prática

Sobre o Instrutor

- **Luis Felipe (LuisDev)**
  - Mais de 7 anos de experiência como Desenvolvedor .NET
  - 4x Microsoft MVP
  - 10x Certificações Microsoft
  - 3x Experiências internacionais remotas para Estados Unidos e Irlanda
  - Mais de 1.000 alunos e centenas de mentorandos
  - Produzindo conteúdo a mais de 6 anos

O que são Testes Unitários

O que são Testes Unitários

- Testes Unitários são métodos que verificam o funcionamento de unidades de código, vulgo métodos, e seus objetos associados.
    - Validação de caminhos principais, alternativos, e de exceção
- O grande objetivo, por incrível que pareça, não é ter uma grande cobertura, e sim resultar em uma arquitetura melhor, menos acoplada, e de melhor manutenção.
- Classes com muitas dependências são muito difíceis de testar.
- **Métrica utilizada:** cobertura de código

Cada vez mais exigidos no mercado

Benefícios

Benefícios

- Cobertura de caminhos principais, alternativos e de exceção

Benefícios

- Cobertura de caminhos principais, alternativos e de exceção
- Detecção de regressão de erros

Benefícios

- Cobertura de caminhos principais, alternativos e de exceção
- Detecção de regressão de erros
- Documentação de regras de negócio

Benefícios

- Cobertura de caminhos principais, alternativos e de exceção
- Detecção de regressão de erros
- Documentação de regras de negócio
- Melhor desenho de código

Desafios na Implementação

Desafios na Implementação

- Códigos difíceis de se testar

Desafios na Implementação

- Códigos difíceis de se testar
- Falta de automação (SonarQube, Jenkins, Azure DevOps, AWS CodePipeline)

Desafios na Implementação

- Códigos difíceis de se testar
- Falta de automação (SonarQube, Jenkins, Azure DevOps, AWS CodePipeline)
- Limitação técnica

Desafios na Implementação

- Códigos difíceis de se testar
- Falta de automação (SonarQube, Jenkins, Azure DevOps, AWS CodePipeline)
- Limitação técnica
- Ego e sabotagem dos desenvolvedores (fator humano)

Desafios na Implementação

- Códigos difíceis de se testar
- Falta de automação (SonarQube, Jenkins, Azure DevOps, AWS CodePipeline)
- Limitação técnica
- Ego e sabotagem dos desenvolvedores (fator humano)
- Falta de clareza no processo de desenvolvimento

Desafios na Implementação

- Códigos difíceis de se testar
- Falta de automação (SonarQube, Jenkins, Azure DevOps, AWS CodePipeline)
- Limitação técnica
- Ego e sabotagem dos desenvolvedores (fator humano)
- Falta de clareza no processo de desenvolvimento
- Prazos curtos

Desafios na Implementação

- Códigos difíceis de se testar
- Falta de automação (SonarQube, Jenkins, Azure DevOps, AWS CodePipeline)
- Limitação técnica
- Ego e sabotagem dos desenvolvedores (fator humano)
- Falta de clareza no processo de desenvolvimento
- Prazos curtos
- Cultura da empresa

Como testar código ruim?

Como testar código ruim?

- Antes de se testar código "ruim", precisamos realizar uma refatoração, afinal, é comum encontrar código fortemente acoplado a implementações de componentes externos, como:
  - Bancos de dados
  - APIs externas
  - Serviços de nuvem

Como testar código ruim?

- O ideal é aplicar técnicas de refatoração!
- Uma técnica bem simples mas muito eficiente é:
  - Extrair método
  - Extrair classe
  - Extrair interface
  - Substituir no código cliente pelo uso da interface via injeção de dependência
    - Em muitas situações, será necessário utilizar o container IoC do framework

Dummies, Mocks, Stubs

Dummies, Mocks, Stubs

- **Dummies:** dados fakes, geralmente criados para apenas serem passados em alguma chamada do método de interesse

Dummies, Mocks, Stubs

- **Dummies:** dados fakes, geralmente criados para apenas serem passados em alguma chamada do método de interesse
- **Mocks:** objeto pré-programado em que são definidas expectativas nas chamadas e respostas que ele possa tratar, com possibilidade de verificação nas chamadas feitas

Dummies, Mocks, Stubs

- **Dummies:** dados fakes, geralmente criados para apenas serem passados em alguma chamada do método de interesse
- **Mocks:** objeto pré-programado em que são definidas expectativas nas chamadas e respostas que ele possa tratar, com possibilidade de verificação nas chamadas feitas
- **Stubs:** definem respostas “chumbadas” em métodos para simplificar os testes, sobrescrevendo as reais implementações, por exemplo
    - *Exemplo:* seu método espera um *INotificationService*, e você cria um *FakeNotificationService* que implementa essa interface, para substituir na hora de passar para a classe a ser testada

Padrão de Escrita

Padrão de Escrita

- **AAA: Arrange, Act, Assert**
  - **Arrange:** preparação para o teste, criação e configuração de mocks
  - **Act:** ação a ser testada
  - **Assert:** checagem do estado pós-ação

Padrão de Escrita

- **AAA: Arrange, Act, Assert**
  - **Arrange:** preparação para o teste, criação e configuração de mocks
  - **Act:** ação a ser testada
  - **Assert:** checagem do estado pós-ação
- **Given_When_Then**
  - Uma classe de teste por caso de uso
  - Uma classe de teste por classe testada

Fundamentos de xUnit

Fundamentos de xUnit

- **xUnit** é um framework de teste unitário para .NET, projetado para ser simples e extensível.
- **Histórico:**
  - Criado por Brad Wilson e Jim Newkirk, que também contribuíram para NUnit.
- **Objetivo:**
  - Promover práticas de teste modernas e padrões de código limpo.

Fundamentos de xUnit

- **Componentes:**
  - **Classe de Teste:** Contém métodos de teste.

Fundamentos de xUnit

- **Componentes:**
  - **Classe de Teste:** Contém métodos de teste.
  - **Método de Teste:** Um método que verifica uma funcionalidade específica.

Fundamentos de xUnit

- **Componentes:**
  - **Classe de Teste:** Contém métodos de teste.
  - **Método de Teste:** Um método que verifica uma funcionalidade específica.
- **Atributos principais:**
  - **[Fact]:** Usado para indicar que um método é um teste de unidade..

Fundamentos de xUnit

- **Componentes:**
  - **Classe de Teste:** Contém métodos de teste.
  - **Método de Teste:** Um método que verifica uma funcionalidade específica.
- **Atributos principais:**
  - **[Fact]:** Usado para indicar que um método é um teste de unidade.
  - **[Theory], [InlineData] e [MemberData]:** Usados para testes parametrizados.

Fundamentos de xUnit

- **Assertivas:**
  - **Principais Métodos:**
    - **Assert.Equal(expected, actual):** Verifica se dois valores são iguais.
    - **Assert.NotEqual(notExpected, actual):** Verifica se dois valores não são iguais.
    - **Assert.True(condition):** Verifica se a condição é verdadeira.
    - **Assert.False(condition):** Verifica se a condição é falsa.

Criando seus primeiros testes com xUnit

Fundamentos de NSubstitute

Fundamentos de NSubstitute

- **NSubstitute** é uma biblioteca para criação de objetos substitutos (mocks) usados em testes unitários.
- **Objetivo:** Facilitar a substituição de dependências em testes unitários, permitindo a simulação de comportamentos e verificação de interações.
- **Características**
    - **Sintaxe Simples e Intuitiva:** Uso de expressões lambda para configuração e verificação.
    - **Suporte para Proxies Dinâmicos:** Criação de objetos substitutos sem a necessidade de configuração explícita.
    - **Integração com Diversos Frameworks de Teste:** Compatível com xUnit, NUnit, MSTest, entre outros.

Fundamentos de NSubstitute

- **Principais usos:**
  - **Substituição de Dependências:** Substituição de dependências para isolar a unidade sendo testada.

Fundamentos de NSubstitute

- **Principais usos:**
  - **Substituição de Dependências:** Substituição de dependências para isolar a unidade sendo testada.
  - **Simulação de Comportamentos Complexos:** Quando métodos têm comportamentos complexos ou dependem de serviços externos.

Fundamentos de NSubstitute

- **Principais usos:**
  - **Substituição de Dependências:** Substituição de dependências para isolar a unidade sendo testada.
  - **Simulação de Comportamentos Complexos:** Quando métodos têm comportamentos complexos ou dependem de serviços externos.
  - **Verificação de Interações:** Verificar se métodos são chamados corretamente durante a execução do teste.

Mocking com NSubstitute

Fundamentos de Moq

Fundamentos de Moq

- **Moq** é uma biblioteca amplamente utilizada para criação de mocks, stubs e fakes em testes unitários no ecossistema .NET.
- **Objetivo:** Facilitar a substituição de dependências em testes de unidade, permitindo a simulação de comportamentos, a verificação de interações e a criação de testes mais isolados e confiáveis.
- **Características**
    - **Sintaxe Simples e Intuitiva:** Uso de expressões lambda para configuração e verificação.
    - **Mocks Fortemente Tipados:** Permite criar objetos substitutos fortemente tipados, garantindo mais segurança e fluidez ao escrever testes.
    - **Configuração Dinâmica de Retornos:** Possibilidade de configurar facilmente comportamentos como retornos condicionais, exceções lançadas, ou execução de callbacks.
    - **Integração com Diversos Frameworks de Teste:** Compatível com xUnit, NUnit, MSTest, entre outros.

Fundamentos de Moq

- **Principais usos:**
  - **Substituição de Dependências:** Substituição de dependências para isolar a unidade sendo testada.
  - **Simulação de Comportamentos Complexos:** Quando métodos têm comportamentos complexos ou dependem de serviços externos.
  - **Verificação de Interações:** Verificar se métodos são chamados corretamente durante a execução do teste.

Mocking com Moq

Validações com Fluent Assertions

Validação com Fluent Assertions

- **Fluent Assertions** é uma biblioteca .NET para escrever assertivas de testes de forma fluida, legível e descritiva.
- **Objetivo:**
  - Melhorar a clareza e a precisão na verificação de resultados esperados em testes unitários e de integração.
- **Características:**
  - Sintaxe fluida e human-readable.
  - Suporte a diferentes tipos de objetos e cenários de teste.
  - Compatível com xUnit, NUnit, MSTest, entre outros.
  - Possibilidade de validações customizadas.

Validação com Fluent Assertions

- **Principais Métodos**
  - **.Should().Be() -** Verifica se valores são iguais.

Validação com Fluent Assertions

- **Principais Métodos**
  - **.Should().Be() -** Verifica se valores são iguais.
  - **.Should().BeNull() / .Should().NotBeNull() -** Verifica se valores são ou não nulos.

Validação com Fluent Assertions

- **Principais Métodos**
  - **.Should().Be() -** Verifica se valores são iguais.
  - **.Should().BeNull() / .Should().NotBeNull() -** Verifica se valores são ou não nulos.
  - **.Should().BeEmpty() / Should().NotBeEmpty() / .Should().Contain() -** Verifica se uma coleção contém está ou não vazia, se contém um item, respectivamente.

Validação com Fluent Assertions

- **Principais Métodos**
  - **.Should().Be() -** Verifica se valores são iguais.
  - **.Should().BeNull() / .Should().NotBeNull() -** Verifica se valores são ou não nulos.
  - **.Should().BeEmpty() / Should().NotBeEmpty() / .Should().Contain() -** Verifica se uma coleção contém está ou não vazia, se contém um item, respectivamente.
  - **.Should().Throw<Exception>() -** Verifica se uma exceção específica foi lançada.

Validação com Fluent Assertions

- **Principais Métodos**
  - **.Should().Be() -** Verifica se valores são iguais.
  - **.Should().BeNull() / .Should().NotBeNull() -** Verifica se valores são ou não nulos.
  - **.Should().BeEmpty() / Should().NotBeEmpty() / .Should().Contain() -** Verifica se uma coleção contém está ou não vazia, se contém um item, respectivamente.
  - **.Should().Throw<Exception>() -** Verifica se uma exceção específica foi lançada.
  - **.Should().BeEquivalentTo() -** Verifica equivalência profunda de objetos.

Geração de dados com Bogus

Geração de Dados com Bogus

- **Bogus** é uma biblioteca .NET para gerar dados fictícios (fake data) de maneira fácil e rápida, ideal para testes, prototipação e preenchimento de bancos de dados.
- **Objetivo:**
  - Ajudar a criar dados realistas para simular cenários de uso, garantindo maior confiabilidade em testes e aplicações de demonstração.
- **Características:**
  - Suporte a diferentes tipos de dados (nomes, endereços, datas, números, etc.).
  - Localização para gerar dados específicos de culturas (ex.: en-US, pt-BR).
  - Configuração simples e intuitiva.

Geração de Dados com Bogus

- **Principais Métodos**
  - **.Name.FirstName() -** Gera um primeiro nome.

Geração de Dados com Bogus

- **Principais Métodos**
  - **.Name.FirstName() -** Gera um primeiro nome.
  - **.Name.LastName() -** Gera um sobrenome.

Geração de Dados com Bogus

- **Principais Métodos**
  - **.Name.FirstName() -** Gera um primeiro nome.
  - **.Name.LastName() -** Gera um sobrenome.
  - **.Address.City() -** Gera um nome de cidade.

Geração de Dados com Bogus

- **Principais Métodos**
  - **.Name.FirstName() -** Gera um primeiro nome.
  - **.Name.LastName() -** Gera um sobrenome.
  - **.Address.City() -** Gera um nome de cidade.
  - **.Phone.PhoneNumber() -** Gera um número de telefone.

Geração de Dados com Bogus

- **Principais Métodos**
  - **.Name.FirstName() -** Gera um primeiro nome.
  - **.Name.LastName() -** Gera um sobrenome.
  - **.Address.City() -** Gera um nome de cidade.
  - **.Phone.PhoneNumber() -** Gera um número de telefone.
  - **.Date.Past() -** Gera uma data no passado.

Geração de Dados com Bogus

- **Principais Métodos**
  - **.Name.FirstName() -** Gera um primeiro nome.
  - **.Name.LastName() -** Gera um sobrenome.
  - **.Address.City() -** Gera um nome de cidade.
  - **.Phone.PhoneNumber() -** Gera um número de telefone.
  - **.Date.Past() -** Gera uma data no passado.
  - **.Finance.CreditCardNumber() -** Gera números de cartão de crédito válidos.

Geração de Dados com Bogus

- **Principais Métodos**
  - **.Name.FirstName() -** Gera um primeiro nome.
  - **.Name.LastName() -** Gera um sobrenome.
  - **.Address.City() -** Gera um nome de cidade.
  - **.Phone.PhoneNumber() -** Gera um número de telefone.
  - **.Date.Past() -** Gera uma data no passado.
  - **.Finance.CreditCardNumber() -** Gera números de cartão de crédito válidos.
  - **.Internet.Email() -** Gera um endereço de email.

Geração de Dados com Bogus

- **Principais Métodos**
  - **.Name.FirstName() -** Gera um primeiro nome.
  - **.Name.LastName() -** Gera um sobrenome.
  - **.Address.City() -** Gera um nome de cidade.
  - **.Phone.PhoneNumber() -** Gera um número de telefone.
  - **.Date.Past() -** Gera uma data no passado.
  - **.Finance.CreditCardNumber() -** Gera números de cartão de crédito válidos.
  - **.Internet.Email() -** Gera um endereço de email
  - **Internet.UserName() -** Gera um nome de usuário.

Geração de Dados com Bogus

- **Também permite a definição de regras customizadas para a geração de nomes, como:**
  - Identificador auto-incremento.

Geração de Dados com Bogus

- **Também permite a definição de regras customizadas para a geração de nomes, como:**
  - Identificador auto-incremento
  - Seleção aleatória a partir de uma coleção definida de itens..

Geração de Dados com Bogus

- **Também permite a definição de regras customizadas para a geração de nomes, como:**
  - Identificador auto-incremento
  - Seleção aleatória a partir de uma coleção definida de itens
  - Número aleatório a partir de um intervalo específico.

Geração de Dados com Bogus

- **Também permite a definição de regras customizadas para a geração de nomes, como:**
  - Identificador auto-incremento
  - Seleção aleatória a partir de uma coleção definida de itens
  - Número aleatório a partir de um intervalo específico
  - Entre outras regras personalizadas.