Testes unitários

By Gabriel Pedroza

Conceito:

São métodos que validam o funcionamento de outros métodos.

Objetivo:

O maior objetivo da escrita dos testes não é apenas a cobertura dos métodos, mas garantir que todos os caminhos principais, caminhos alternativos e exceções estejam validados, garantindo uma boa arquitetura, mais coesa e menos acoplada.

O acoplamento dificulta a escrita dos testes.

Benefícios:

• Cobertura de caminhos principais, alternativos e exceções dos métodos testados.
• Facilita a refatoração de métodos.
• Documentação de regras de negócio.

Desafios:

• Acoplamento;
• Limitação Técnica;
• Fatores humanos (Ego e sabotagem);
• TDD: Teste depois do deploy;
• Prazos curtos;
• Priorizar a qualidade à rapidez da entrega;

Características de um código ruim de se testar:

É comum encontrarmos códigos fortemente acoplados a implementação de componentes externos, como:

• Banco de dados;
• API's externas;
• Serviços de Cloud;

Antes de criarmos testes para um código ruim, o ideal é começarmos pela refatoração do mesmo, que consistem em algumas etapas, como:

• Extrair métodos;
• Extrair Classes;
• Extrair Interfaces;
• Substituir o código externo por interface, sendo possível utilizar via injeção de dependência;

Dummies, Stubs e Mocks:

• **Dummies**: Objetos "falsos" que são usados apenas para preencher parâmetros obrigatórios nos métodos, mas que não são realmente utilizados no teste. Servem apenas como "espaçadores".

• **Stubs**: Objetos que substituem métodos ou funções reais e retornam respostas pré-definidas. São usados para simular o comportamento de componentes que o teste precisa, mas não quer executar de verdade.

• **Mocks**: São objetos que não apenas substituem métodos ou funções reais, mas também verificam se determinadas interações ocorrem durante o teste. Além de

Testes unitários

By Gabriel Pedroza

simular, os mocks validam se certas chamadas ou ações foram feitas.

Qual a diferença entre mock e stub?

Enquanto um stub apenas provê respostas prontas para as chamadas que serão feitas durante o teste, o mock vai mais além e, além de prover as respostas, também valida as chamadas - ele conhece o comportamento esperado do sistema e testa este comportamento.

Em quais situações usar um é mais vantajoso que usar o outro?

Estabelecido que ambos servem para substituir componentes reais (eles são "dublês" destes componentes) durante os testes, e entendida a diferença entre eles, fica nítido quando usar um e quando usar outro:

• Use **stub** para testar se um código, dada uma determinada entrada (respostas prontas dos métodos do stub), produz determinada saída.

• Use **mock** para testar se um código se comporta da maneira esperada no que tange a interações com o componente que o mock está substituindo.

Padrões de escrita:

• **AAA:** Arrange, Act e Assert
    • **Arrange:** Preparação para o teste.
    • **Act:** Ação a ser testada.
    • **Assert:** Validação pós ação.

• **Given_When_Then:**
    • Uma classe por caso de uso.
    • Uma classe por classe testada.

Links úteis

• **Repositório com exemplos de Dummy, Stub e Mock:** GPedrozaR/UnitTestOverview (github.com).