PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA
BACHARELADO EM ENGENHARIA DE SOFTWARE

MATHEUS NOLASCO MIRANDA SOARES

MATRÍCULA: 688826

Test Pattern

Testes de Software

Padrões de Criação de Dados (Builders)

No desenvolvimento dos testes deste projeto, optou-se pelo uso do padrão de criação de dados baseado em Builder, especificamente o CarrinhoBuilder, em vez de um CarrinhoMother. Essa escolha se deu porque o Object Mother, embora útil para criar objetos prontos para uso, tende a ser pouco flexível, pois cada método retorna um objeto fixo e qualquer variação exige a criação de novos métodos. No contexto do projeto, havia a necessidade de criar carrinhos com diferentes combinações de usuários e itens, incluindo carrinhos vazios ou com valores específicos para validar regras de desconto.

O CarrinhoBuilder, por sua vez, oferece valores padrão para um carrinho válido e permite a customização de apenas o que é relevante para o cenário de teste, por meio de métodos encadeados. Isso reduz a duplicação de código e torna a intenção do teste mais clara. Ao utilizar o Builder, o código dos testes foca no cenário que está sendo validado, sem se perder em detalhes de construção do objeto, e qualquer alteração na estrutura do carrinho pode ser ajustada apenas no Builder, sem a necessidade de modificar todos os testes que o utilizam.

Padrões de Test Doubles (Mocks vs. Stubs)

No cenário de teste que valida o sucesso de uma compra realizada por um cliente Premium, foi possível aplicar de forma clara os conceitos de Mocks e Stubs. Nesse teste, o Gateway de Pagamento foi implementado como um Stub, pois sua função era apenas fornecer uma resposta controlada de pagamento aprovado, permitindo verificar o estado final do sistema após o processamento do pedido. O foco estava no resultado e não na quantidade ou forma das chamadas realizadas.

O repositório de pedidos também foi tratado como um Stub, retornando um pedido salvo sem que fosse necessário verificar interações. Já o serviço de e-mail foi implementado como um Mock, pois nesse caso o objetivo era verificar o comportamento, garantindo que o método de envio de e-mail fosse chamado exatamente uma vez e com os parâmetros corretos, incluindo o endereço do cliente, o assunto e a mensagem. Essa distinção entre Stub e Mock é fundamental para que o teste mantenha clareza sobre o que está sendo validado, separando a verificação de estado da verificação de comportamento.

Conclusão

O uso deliberado de padrões como Builders e Test Doubles contribuiu significativamente para a qualidade e a sustentabilidade da suíte de testes. O Builder eliminou a duplicação e tornou o código de teste mais expressivo, facilitando a criação de cenários complexos sem poluir o teste com detalhes irrelevantes. Já os Test Doubles, ao isolarem o código testado de dependências externas, garantiram que os testes fossem rápidos, previsíveis e focados no que realmente importa.

A separação clara entre verificação de estado, feita com Stubs, e verificação de comportamento, feita com Mocks, evitou confusões e garantiu que cada teste tivesse um objetivo claro. Essas práticas ajudaram a prevenir problemas comuns em testes, como fragilidade, configuração excessiva e lentidão, resultando em uma suíte de testes mais confiável, legível e fácil de manter ao longo do tempo.