Implementação de Padrões de Teste

Disciplina: Teste de Software
Trabalho: Test Patterns
Nome: Matheus Dias Mendes
Data: 09/11/2025

Padrões de criação de dados(Builders)

No cenário apresentado, a criação de dados de teste para objetos como Carrinho exige uma flexibilidade maior do que a oferecida pelo padrão Object Mother. O CarrinhoBuilder é umas das melhores opções por conta da sua flexibilidade e fluidez, permitindo a criação de objetos de forma mais customizável e reutilizável, sem a necessidade de criar múltiplos métodos estáticos como no Object Mother. Ele oferece um construtor com valores padrão e métodos encadeados para modificar os atributos conforme o teste exige, como a inclusão de itens, usuários e outros detalhes dinâmicos do carrinho

Antes:

```
const usuario = new User('Nome', 'email@email.com');
const item1 = new Item('Produto1', 100);
const item2 = new Item('Produto2', 150);
const carrinho = new Carrinho();
carrinho.adicionarItem(item1);
carrinho.adicionarItem(item2);
carrinho.definirUsuario(usuario);
```

Depois:

```
const carrinho = new CarrinhoBuilder()
  .comUsuario(usuario)
  .comItens([item1, item2])
  .build();
```

O CarrinhoBuilder melhora a legibilidade, deixando claro que o foco do teste é o Carrinho e permitindo configurar seus dados de forma simples e intuitiva. Isso facilita a manutenção e a adição de novos cenários de teste, já que mudanças no setup não exigem alterações em múltiplos locais de código

Padrões de Test Doubles (Mocks vs. Stubs)

Escolha do teste "sucesso Premium" (Etapa 5):

Neste teste, o objetivo é verificar o comportamento do serviço de CheckoutService quando um cliente Premium finaliza a compra, recebendo um desconto de 10%.

Identificação do Stub e do Mock:

Stub: O GatewayPagamento foi utilizado como Stub, pois seu papel era fornecer uma resposta pré-definida success: true sem validar interações ou chamadas subsequentes. Seu objetivo era controlar o fluxo do teste, retornando um valor fixo para testar a lógica do CheckoutService.

Mock: O EmailService foi utilizado como Mock. O foco aqui era validar as interações, ou seja, garantir que o serviço de e-mail fosse chamado com os argumentos corretos. O uso jest.fn() para monitorar as chamadas e garantir que o e-mail fosse enviado corretamente para o cliente Premium.

Justificativa:

- O GatewayPagamento foi tratado como Stub porque o foco era validar o comportamento do sistema em resposta ao pagamento (verificação de estado), sem nos preocupar com os detalhes da implementação do gateway.

- O EmailService, por outro lado, foi tratado como Mock porque precisava verificar o comportamento do sistema, ou seja, garantir que o método enviarEmail fosse chamado com os parâmetros corretos, validando a interação.

Conclusão

A aplicação de padrões de teste, como Builders e Test Doubles (Mocks e Stubs), ajuda significativamente na prevenção de Test Smells, tornando a suíte de testes mais robusta, legível e de fácil manutenção. O uso do CarrinhoBuilder torna o setup dos testes mais expressivo e flexível, enquanto a diferenciação entre Mocks e Stubs permite uma abordagem mais clara e eficaz para validar tanto o estado quanto o comportamento do sistema.