

Test Pattern

Thiago Andrade Ramalho

808904

01.11.2025

Teste de Software

Padrões de Criação de Dados (Builders)

A melhor opção é o CarrinhoBuilder em vez de um Carrinho Mother, pois com o builder, a legibilidade e compreensão do teste é muito maior, pois o código acaba sendo um passo a passo das ações. Outro ponto importante é a flexibilidade para variações, por conta do carrinho ter muitas combinações, o builder permite cobrir só o que muda, além também de que se a estrutura interna mudar, basta alterar a parte necessária, já com o mother teria que alterar vários testes.

```
describe('quando um cliente Premium finaliza a compra - ANTES', () => {
  test('deve aplicar desconto de 10% e enviar e-mail de confirmação', async () => [
    const usuarioPremium = new User(7, 'Carlos Premium', 'premium@email.com', 'PREMIUM');

    const itemA = new Item('Produto A', 100);
    const itemB = new Item('Produto B', 100);
    const itens = [itemA, itemB];

    const carrinho = new Carrinho(usuarioPremium, itens);

    const gatewayStub = {
      cobrar: jest.fn().mockResolvedValue({ success: true })
    };

    const repositoryStub = {
      salvar: jest.fn().mockResolvedValue({ id: 99, /* ... */ })
    };

    const emailMock = {
      enviarEmail: jest.fn().mockResolvedValue()
    };

    const service = new CheckoutService(gatewayStub, repositoryStub, emailMock);

    const pedido = await service.processarPedido(carrinho, '4111111111111111');

    expect(gatewayStub.cobrar).toHaveBeenCalledWith(180, '4111111111111111');
    expect(repositoryStub.salvar).toHaveBeenCalled();
    expect(emailMock.enviarEmail).toHaveBeenCalledWith(
      usuarioPremium.email,
      'Seu Pedido foi Aprovado!',
      expect.stringContaining('R$180')
    );
  ]);
}
```

```

27  ✓ describe('quando um cliente Premium finaliza a compra', () => {
28    ✓ test('deve aplicar desconto de 10% e enviar e-mail de confirmação', async () => {
29      const usuarioPremium = UserMother.umUsuarioPremium();
30      const itens = [new Item('Produto A', 100), new Item('Produto B', 100)]; // total 200
31
32      const carrinho = new CarrinhoBuilder().comUser(usuarioPremium).comItens(itens).build()
33
34      const gatewayStub = { cobrar: jest.fn().mockResolvedValue({ success: true }) };
35      const repositoryStub = { salvar: jest.fn().mockResolvedValue({ id: 42 }) };
36
37      const emailMock = { enviarEmail: jest.fn().mockResolvedValue(undefined) };
38
39      const checkoutService = new CheckoutService(gatewayStub, repositoryStub, emailMock);
40
41      const pedidoSalvo = await checkoutService.processarPedido(carrinho, '4111111111111111');
42
43      expect(gatewayStub.cobrar).toHaveBeenCalledWith(180, '4111111111111111');
44      expect(repositoryStub.salvar).toHaveBeenCalled();
45
46      expect(emailMock.enviarEmail).toHaveBeenCalledTimes(1);
47      expect(emailMock.enviarEmail).toHaveBeenCalledWith(
48        usuarioPremium.email,
49        'Seu Pedido foi Aprovado!',
50        `Pedido ${pedidoSalvo.id} no valor de R${pedidoSalvo.valor}` );
51    });
52  });
53});

```

Padrões de Test Doubles (Mocks vs. Stubs)

No teste de sucesso premium, o GatewayPagamento e o PedidoRepository são stubs pois servem apenas para retornar respostas esperadas , seja o suces: true para o gateway, e o repository retorna o id, ou seja, não é testado nada, apenas verificação de estado, como por exemplo, se o pedido foi criado e o valor a ser cobrado. Já o EmailService é mock, pois a função é justamente verificar se a ação aconteceu e como aconteceu, ou seja, verificar o comportamento, se o email foi enviado e com quais dados.

Conclusão

Usar Builders e Test Doubles ajuda a manter a suítes de testes limpa e sustentável. Builders reduzem a duplicação de código e melhoram a legibilidade e compreensão do código do teste, isso evita smells como Long Setup e Obscure Intent. Já com os Doubles bem aplicados, fazem com que stubs controlem o estado/entradas com respostas esperadas e mocks verificam efeitos colaterais. Isso isola o SUT e deixa os testes mais determinísticos. Assim o teste mostra

apenas o que muda e o que deve acontecer, reduzindo acoplamento e white-box testing. Dessa forma se tem suíte mais rápida, menos frágil e mais fácil de refatorar.

```
● PS C:\Users\Thiago Andrade\Documents\GitHub\test-pattern> npm test
  > test-pattern@1.0.0 test
  > jest

    PASS  __tests__/builders/CarrinhoBuilder.js
    PASS  __tests__/builders/UserMother.js
    PASS  __tests__/CheckoutService.test.js

  Test Suites: 3 passed, 3 total
  Tests:       7 passed, 7 total
  Snapshots:   0 total
  Time:        2.375 s, estimated 6 s
  Ran all test suites.
```