Pontifícia Universidade Católica de Minas Gerais

Engenharia de Software
Teste de Software

Implementando Padrões de Teste (Test Patterns)

Aluno: Ana Flávia de Carvalho Santos
Matrícula: 764588
Professor: Prof. Cleiton Tavares

Belo Horizonte
8 de novembro de 2025

1 Padrões de Criação de Dados (Builders)

O padrão Object Mother, implementado no UserMother, é ideal para a criação de instâncias de objetos simples e fixos, ele funciona bem para entidades como usuário padrão ou premium, que raramente mudam entre os testes. Já o carrinho é um objeto complexo que varia muito, pode ter diferentes usuários, listas de itens, ou estar vazio. Se usássemos um CarrinhoMother para ele, teríamos que criar vários métodos diferentes para prever diferentes cenários, por exemplo um carrinho vazio, um com vários itens, um "premium", o que seria insustentável. O padrão Data Builder resolve isso ao fornecer uma instância padrão válida que pode ser customizada com métodos como .comUser() ou .vazio(), permitindo ao teste focar apenas no que é relevante para o cenário.

1.1 Exemplo Antes

```
const usuarioPremium = new User(
    2,
    'Usuário Premium',
    'premium@email.com',
    'PREMIUM'
);
const itens = [
    new Item('Item 1', 150.00),
    new Item('Item 2', 50.00)
];
const carrinho = new Carrinho(usuarioPremium, itens);
```

O código acima demonstra o Test Smell de "Setup Obscuro". Para entender o cenário (um carrinho premium de 200 reais), o leitor do teste precisa ler váriass linhas de inicialização, ignorar detalhes irrelevantes (IDs, e-mails, nomes de itens) e até realizar cálculos mentais (150 + 50). A intenção do teste fica perdida no meio da complexidade da implementação.

1.2 Exemplo Depois

```
const usuarioPremium = UserMother.usuarioPremium();

const itens = [
    new Item('Item 1', 150.00),
    new Item('Item 2', 50.00)
];

const carrinho = new CarrinhoBuilder()
    .comUser(usuarioPremium)
    .comItens(itens)
    .build();
```

Utilizando os padrões, o setup se torna uma declaração de intenção. O UserMother esconde a complexidade de criar um usuário premium. O CarrinhoBuilder torna explícito o que está sendo customizado (.comUser() e .comItens()). O teste fica limpo, focado, e se

lê como uma narrativa do cenário que está sendo preparado, melhorando drasticamente a clareza.

O Data Builder melhora a legibilidade e a manutenção dos testes. A legibilidade é aprimorada pois o "Setup Obscuro"é eliminado, o teste passa a focar na sua intenção sem aumentar a complexidade pelos detalhes de implementação. A manutenção torna-se mais robusta, pois o conhecimento sobre como construir os objetos fica centralizado nos Builders. Se, por exemplo, o construtor da classe User mudar, apenas o UserMother precisará ser atualizado, em vez de dezenas de testes quebrarem individualmente.

2 Padrões de Test Doubles (Mocks vs. Stubs)

Para a análise, foi escolhido o teste de "sucesso Premium". Neste cenário, o objetivo é testar o fluxo completo de um checkout bem-sucedido para um usuário que possui desconto, o que envolve múltiplas dependências externas. Neste teste, o GatewayPagamento e o PedidoRepository foram usados como Stubs e o EmailService foi usado como um Mock.

O GatewayPagamento atuou como um Stub porque sua principal função era fornecer uma resposta pré-definida para permitir que o fluxo do teste continuasse. Não estávamos testando o gateway em si, mas sim controlando sua saída para verificar o que acontecia depois do pagamento. Embora tenhamos verificado com quais argumentos ele foi chamado, seu papel principal era fornecer dados para a verificação de estado, permitindo que o SUT avançasse para salvar o pedido.

O EmailService, por outro lado, foi um Mock clássico e seu retorno não afeta o fluxo do checkout. O objetivo aqui não era verificar um estado, mas sim verificar um comportamento para garantir que, ao final do processo, o método enviarEmail foi chamado exatamente uma vez e com os argumentos corretos, que eram o e-mail do usuário, o assunto e o corpo da mensagem. Isso se classifica como verificação de comportamento.

3 Conclusão

O uso de Padrões de Teste, como os Builders e os Test Doubles, é um investimento direto na qualidade e longevidade de uma suíte de testes. Padrões de criação, como o Data Builder, previnem o Test Smell de "Setup Obscuro", como foi provado nesse estudo. Ao encapsular a complexidade da criação de objetos, eles simplificam o Arrange do teste, deixando esse trecho mais legível e focado na sua intenção, sem precisar dos detalhes da implementação. Além disso, os Test Doubles (stubs e mocks) previnem testes frágeis, outro test smell , pois permitem o isolamento da unidade sob teste, garantindo que o teste falhe apenas por razões relevantes e não por falhas em serviços de terceiros. Juntos esses padrões contribuem para uma suíte de testes sustentável, com um código que é fácil de manter, rápido de executar e confiável.