UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ CAMPUS LUIZ MENEGHEL BANDEIRANTES- PR CENTRO DE CIÊNCIAS TECNOLÓGICAS RESUMO

ÍCARO QUEIROZ RECCANELLO

PARTE FINAL - TESTES

Bandeirantes - Paraná

2024

Classes de testes implementadas:

[Repositório no GitHub]

Tests Smells encontrados:

Classe `AlunoTest`

1. **Assertion Roulette**
    - **Problema**: Alguns testes têm múltiplos assertions sem mensagens descritivas (`testVerficaAlunoValido`, `testVerificaDebitoSemDebito`).
    - **Detecção**: Falta de explicação detalhada no caso de falha.
2. **Magic Number Test**
    - **Problema**: O RA `"10"` e `"123"` são usados diretamente como valores literais nos testes.
    - **Detecção**: Utilização de números mágicos no lugar de constantes significativas.

Classe `ControleTest`

1. **General Fixture**
    - **Problema**: A configuração dos códigos de livros e o objeto `Controle` são utilizados de forma redundante em vários testes.
    - **Detecção**: A fixture não é otimizada entre os métodos de teste.
2. **Magic Number Test**
    - **Problema**: Códigos de livros `{1, 2, 3}` aparecem diretamente nos métodos.
    - **Detecção**: Substituir por constantes descritivas.
3. **Eager Test**

- **Problema**: Métodos como `testEmprestarComLivrosValidos` chamam múltiplos métodos de produção, cobrindo mais de um comportamento.
- **Detecção**: Testes deveriam ser segmentados para cada funcionalidade.

Classe `EmprestimoTest`

1. **Assertion Roulette**
   - **Problema**: O teste `testCalculaDataDevolucao` contém assertions sem mensagens explicativas.
   - **Detecção**: Difícil identificar o motivo do erro caso o teste falhe.
2. **Magic Number Test**
   - **Problema**: O prazo 5 e outros números literais aparecem diretamente.
   - **Detecção**: Uso de constantes descritivas seria preferível.
3. **Redundant Print**
   - **Problema**: Não detectado nesta classe.
   - **Detecção**: Não há prints no código.

Classe `LivroTest`

1. **Magic Number Test**
   - **Problema**: Os valores 2, 1 e 3 são usados diretamente nos métodos.
   - **Detecção**: Deveriam ser substituídos por constantes descritivas.
2. **Eager Test**
   - **Problema**: Métodos como `testVerPrazo` e `testVerificaLivroValido` invocam múltiplos métodos de produção.
   - **Detecção**: Testes poderiam ser divididos para maior clareza.

Classe `DebitoTest`

1. **Magic Number Test**
    - **Problema**: O código do aluno 4 e 1 são usados diretamente.
    - **Detecção**: Substituir por constantes.
2. **Assertion Roulette**
    - **Problema**: Falta de mensagens descritivas nos assertions (`testVerificaDebitoComDebito`, `testVerificaDebitoSemDebito`).
    - **Detecção**: Melhora a compreensão dos erros.

Classe `ItemTest`

1. **General Fixture**
    - **Problema**: O mesmo objeto `Livro` é recriado em vários métodos.
    - **Detecção**: Poderia ser inicializado em uma fixture compartilhada.
2. **Assertion Roulette**
    - **Problema**: Assertions sem mensagens explicativas no teste `testCalculaDataDevolucao`.
    - **Detecção**: Dificulta a identificação de falhas.
3. **Magic Number Test**
    - **Problema**: O prazo 5 usado diretamente.
    - **Detecção**: Poderia ser substituído por uma constante significativa.

Melhorias Introduzidas

1. Resolução do "Assertion Roulette"

- **Alteração**: Adicionamos mensagens explicativas aos métodos de teste em todos os assertions.
- **Impacto Positivo**:
    - Facilita a identificação do motivo de falhas nos testes, reduzindo o tempo de depuração.
    - Melhora a compreensão do propósito de cada teste por novos desenvolvedores ou membros da equipe.

2. Resolução do "Magic Number Test"

- **Alteração**: Substituímos números mágicos (e.g., `10`, `123`, `4`) por constantes nomeadas com significado claro.
- **Impacto Positivo**:
    - Aumenta a legibilidade, pois as constantes comunicam o propósito dos valores usados nos testes.
    - Reduz erros decorrentes de alterações nos valores dos números mágicos, uma vez que as constantes são fáceis de atualizar globalmente.

3. Otimização de Fixtures (Resolução Parcial do "General Fixture")

- **Alteração**: Consolidamos configurações repetitivas nos métodos de teste, reutilizando variáveis e introduzindo constantes globais.
- **Impacto Positivo**:
    - Redução de duplicação de código, tornando os testes mais compactos.
    - Facilita a manutenção ao centralizar configurações comuns.

4. Melhor Segmentação de Testes (Mitigação do "Eager Test")

- **Alteração**: Dividimos métodos que testavam múltiplas funcionalidades em testes específicos.
- **Impacto Positivo**:
    - Cada teste agora valida um único comportamento, alinhando-se às boas práticas de granularidade.
    - Reduz a complexidade e facilita a análise de falhas.

5. Melhor Documentação Implícita

- **Alteração**: Adicionamos mensagens em todos os assertions e renomeamos testes para refletir melhor o comportamento testado.
- **Impacto Positivo**:
    - Documentação implícita nos testes ajuda na comunicação do propósito do código, sem a necessidade de comentários extensos.
    -

Impactos e Limitações

A introdução de constantes e a divisão dos testes podem, em um primeiro momento, aumentar a complexidade inicial para novos desenvolvedores que precisem interagir com o código. Isso ocorre porque a estrutura mais detalhada e segmentada exige maior compreensão das convenções e da organização adotadas nos testes. No entanto, essa complexidade inicial é amplamente compensada por uma compreensão mais clara e intuitiva do código a longo prazo, o que facilita a manutenção e a evolução do projeto.

Embora as melhorias realizadas tenham focado na qualidade e na clareza dos testes, a cobertura dos cenários de teste existentes não foi expandida de forma significativa. Ainda há oportunidades para introduzir mais casos de borda que lidem com situações extremas ou inesperadas no código, assim como para desenvolver testes de integração que validem o comportamento de fluxos completos do sistema. Esses testes adicionais poderiam contribuir para uma validação mais abrangente e robusta do software.

As fixtures utilizadas nos testes foram otimizadas em algumas áreas, mas ainda existem situações em que inicializações repetitivas poderiam ser substituídas por métodos de configuração global. O uso de anotations como `@Before` ou `@BeforeEach` poderia centralizar essas configurações, reduzindo ainda mais a duplicação de código e simplificando a manutenção.

Por fim, é importante destacar que não foram detectados smells como "Lazy Tests" ou "Redundant Print" no código analisado. Apesar disso, mantivemos atenção constante para evitar que esses problemas fossem introduzidos durante o processo de refatoração.

As mudanças introduzidas melhoraram substancialmente a **qualidade do código de teste**, abordando os principais problemas de manutenibilidade e legibilidade. O projeto está agora mais alinhado às boas práticas de testes unitáriosmais robusto e preparado para futuras manutenções ou incrementos.