09 - Unit Tests

2025-09-25 8:13 am

- Testes devem ser permanentes, completos e facilmente reprodutíveis.

TRÊS LEIS DO TDD

1. Você não deve escrever código em produção até ter escrito um teste correspondente em falha
2. Você não deve escrever em um teste além do que o suficiente para falhar - inclusive falha por compilação
3. Você não deve escrever mais código em produção além do que é o suficiente para passar o teste em falha. 5. Teste e código de produção são escritos juntos.
4. Assim a cobertura será 100%.
    - Mas geri-los pode se tornar um problema por si só

MANTER TESTES LIMPOS

- Testes ruins são igual ou pior do que não ter testes.
    - Passam a falhar não se sabe por quê.
    - Vira um fardo.
- Testes devem seguir padrão e serem organizados.
- Código de teste é tão importante quando código de produção.
    - Requer tanto cuidado e atenção quanto.
- Testes dão flexibilidade ao desenvolvedor
    - Permite criar mudanças sem receios.

TESTES LIMPOS

- O que são?
- Mais importante**: LEGIBILIDADE**
    - Depende das mesmas coisas que tornam código de produção legível.
        - Clareza, simplicidade, etc.
    - Não deve ter detalhes desnecessários
- Padrão BUILD-OPERATE-CHECK de teste.
    - Três etapas simples.
- Uma alternativa é criar API específica para testes
- Podemos também criar uma linguagem para teste:
    - e.g. maiúsculo = ON, minúsculo = OFF

```
@Test
  public void turnOnCoolerAndBlowerIfTooHot() throws Exception {
    tooHot();
    assertEquals("hBChl", hw.getState());
  }

  @Test
  public void turnOnHeaterAndBlowerIfTooCold() throws Exception {
    tooCold();
    assertEquals("HBchl", hw.getState());
  }
```

DUPLO PADRÃO

- Código de teste não precisa ser eficiente.
  - Ambiente de teste não é o mesmo de produção
  - O que é ruim em um pode ser aceitável no outro
  - Mas ambos devem ser limpos

UM ASSERT POR TESTE

- Ideal é um assert por teste, mais fácil de ler
  - Se tiver vários, podemos dividir em vários testes
  - Mas às vezes isso gera código duplicado
- "given-when-then" convention:
  - Análogo ao build-operate-check
- Template Method:
  - Tenta evitar esta duplicação
- Podemos colocar o "given-when" em uma classe e criar os testes individuais apenas para o "then"
- Mas se for necessário, tudo bem colocar vários asserts.
  - O importante é que seja o mínimo necessário.
- O mais importante é ter UM CONCEITO POR TESTE

F.I.R.S.T.

- **Fast**:
  - Teste deve ser rápido, para rodar com frequência
- **Independent**:
  - Resultado de um teste não deve depender de outro, apenas dele em si. Sem efeito cascata.
- **Repeatable**:
  - Deve ser repetível em qualquer ambiente - seja de produção, de QA, no laptop, na rede principal, etc. Não deve depender do ambiente de forma alguma.
- **Self-Validating**:
  - Output deve ser Passar ou Falhar. Sem detalhes e sem requerer análise ou pensamento subjetivo.
- **Timely**:
  - Teste unitário deve ser escrito logo antes do código de produção que o faz passar. Código de produção portanto deve ser pensado para ser testável.