

Catálogo de Test Smells

Dalton Nicodemos Jorge

daltonjorge@copin.ufcg.edu.br

Test smells são antipadrões que dão indícios de possíveis problemas no design e implementação em testes de software.

Catálogo de Test smells

Obs:

1. Este catálogo contempla apenas 16 smells selecionados para o estudo.
2. Os exemplos seguem o formato de saída para mensagens de erro do [@babel/code-frame](#), que aponta a linha e coluna onde ocorre o smell.

1. Assertion Roulette

- Vários asserts sem mensagem clara do que está sendo testado
- Dificulta identificar qual asserção falhou

Exemplo:

```
288 |         axios.get('http://localhost:4444/one').then(function (res) {
289 |             assert.equal(res.data, str);
> 290 |             assert.equal(res.request.path, '/two');
      |             ^ Assertion Roulette
291 |             done();
292 |         }).catch(done);
293 |     });
```

2. Conditional Test Logic

- Laços e condições tornam o teste confuso
- Difícil entender o fluxo e casos testados

Exemplo:

```
1776 | assert.deepStrictEqual(samples, Array.from(function* () {  
> 1777 |   for (let i = 1; i <= 10; i++) {  
      |   ^ Conditional Test Logic  
1778 |     yield ({  
1779 |       loaded: chunkLength * i,  
1780 |       total: contentLength,
```

3. Duplicate Assert

- Asserções iguais com os mesmos parametros em um caso de teste
- Compromete a responsabilidade do caso de teste em ter um único objetivo

Exemplo:

```
181 | headers.set('foo', 'bar=value1');
182 |
> 183 | assert.strictEqual(headers.has('foo'), true);
    | ^ Duplicate Assert
184 |
185 | headers.delete('foo', /baz=/);
186 |
> 187 | assert.strictEqual(headers.has('foo'), true);
    | ^ Duplicate Assert
}
```

4. Eager Test

- Teste que exercita múltiplos métodos do sistema sob teste.
- Adiciona mais dependência ao caso de teste

Exemplo:

```
17 | import { body2json, beautify } from '../template/src/jsonifier.mjs';
18 |
19 | describe('test jsonifier', function () {
20 |     it('should convert properly fields to json', async function () {
    ...
146 |
> 147 |         assert.strictEqual(body2json(context), beautify(fixture));
    |                                ^ Eager Test                ^ Eager Test
148 |         return;
149 |     });
150 | });
```

5. Empty Test

- Corpo do teste é um bloco vazio ou com conteúdo totalmente comentado
- Desperdício de processamento e falsa sensação de sucesso

Exemplo:

```
66 | });  
67 |  
> 68 | it("adding dynamic tests...", async function() {}); // this is required to work  
   |      ^ Empty Test  
69 | });  
70 |
```


6. Exception Handling

- Presença de blocos de tratamento de exceção
- Adiciona complexidade e dependência desnecessárias

Exemplo:

```
17 | it('name - required', async () => {  
> 18 |   try {  
    |   ^ Exception Handling  
19 |     await Tag.insert({});  
20 |   } catch (err) {  
21 |     err.message.should.be.eql('`name` is required!');
```

7. Global Variable

- Permite redeclaração e atualização das variáveis
- Gera inconsistência e imprevisibilidade

Exemplo:

```
7 | it('applyEach', function (done) {  
> 8 |     var call_order = [];  
   |     ^ Global Variable  
   |     var one = function (val, cb) {  
10 |         expect(val).to.equal(5);  
11 |         setTimeout(function () {
```

8. Ignored Test

- Usa alguma sintaxe específica para evitar a avaliação das asserções
- Recurso momentâneo que pode se tornar permanente

Exemplo:

```
> 195 | it.skip('render() - execute after_render:html', async () => {  
    |     ^ Ignored Test  
    196 |     const body = [  
    197 |         '{{ test }}'  
    198 |     ].join('\n');
```

9. Lazy Test

- Um mesmo método de produção é passado como argumento nas asserções em múltiplos casos de testes de uma mesma suite
- Dificulta a manutenibilidade da suite de testes

Exemplo:

```
17 | it('casts array with ObjectIds to $in query', function() {
17 |     const schema = new Schema({ x: Schema.Types.ObjectId });
18 |     const ids = [new ObjectId(), new ObjectId()];
> 19 |     assert.deepEqual(cast(schema, { x: ids }), { x: { $in: ids } });
    |                      ^ Lazy Test
20 | });
21 |
22 | it('casts array with ObjectIds to $in query when values are strings', function() {
23 |     const schema = new Schema({ x: Schema.Types.ObjectId });
24 |     const ids = [new ObjectId(), new ObjectId()];
> 25 |     assert.deepEqual(cast(schema, { x: ids.map(String) }), { x: { $in: ids } });
    |                      ^ Lazy Test
26 | });
```

10. Magic Number Rule

- Números mágicos sem significado claro
- Difícil entender a lógica do teste

Exemplo:

```
238 | assert.equal(doc.nested.age, 5);  
    |                                     ^ Magic Number  
239 | assert.equal(String(doc.nested.cool), '4c6c2d6240ced95d0e00003c');  
> 240 | assert.equal(doc.nested.agePlus2, 7);  
    |                                     ^ Magic Number
```

11. Mystery Guest

- Teste faz uso de recursos externos, ex: arquivos, banco de dados ou serviços web
- Torna o teste não-determinístico

Exemplo:

```
845 | it('should support HTTPS proxies', function (done) {  
846 |     var options = {  
> 847 |         key: fs.readFileSync(path.join(__dirname, 'key.pem')),  
      |         ^ Mystery Guest  
848 |         cert: fs.readFileSync(path.join(__dirname, 'cert.pem'))  
849 |     };  
      |
```

12. Redundant Assertion

- Asserções que sempre retornam mesmo resultado
- Indica que a asserção não tem objetivo claro

Exemplo:

```
42 |         it('each empty array', function(done) {  
43 |             async.each([], function(x, callback){  
> 44 |                 assert(false, 'iteratee should not be called');  
         |                 ^ Redundant Assertion  
45 |                 callback();  
46 |             }, function(err){  
47 |                 if (err) throw err;
```

13. Redundant Print

- Decorre de uma má prática de depuração
- Aumentam o tempo de execução do caso de teste afetado

Exemplo:

```
27 | it('should throw an error when calling with wrong arguments', function () {  
> 28 |     assert.throws(function () { console.log(new AccessorNode()) }, TypeError)  
    |                                     ^ Redundant Print
```


14. Resource Optimism

- Teste assume que recursos externos estarão disponíveis
- Pode falhar em produção por recursos indisponíveis

Exemplo:

```
79 |  
80 |  
> 81 |         try {  
      |             await fs.readFile(path.join(destDir, 'cards.min.css'), 'utf-8');  
      |             ^ Resource Optimism  
82 |             should.fail(cardAssets, 'CSS file should not exist');  
83 |         } catch (error) {  
84 |             if (error instanceof should.AssertionError) {
```

15. Sleepy Test

- Uso de interrupções temporárias no teste
- Resultados dos testes tornam-se imprevisíveis

Exemplo:

```
> 92 |      setTimeout(() => {  
    |      ^ Sleepy Test  
    93 |          scope.adapter._pingUrl.calledOnce.should.eql(true);  
    94 |          done();  
    95 |      }, 50);
```

16. Unknown Test

- Caso de teste sem asserções
- Mesmas consequências do *Empty Test*

Exemplo:

```
> 20 | it('should work in dry run mode', async function () {  
   | ^ Unknown Test  
   21 |     const app = {  
   22 |         options: {  
   23 |             dryRun: true,  
   24 |         },  
   25 |         log: logger,  
   26 |     };  
   27 |  
   28 |     const writer = new Writer({}, app);  
   29 |     return writer.write();  
   30 | });
```