Relatório de Refatoração e Testes

Disciplina: Testes de Software

Professor: Cleiton Silva

Aluno:

Bruno Evangelista

Engenharia de Software

10 de novembro de 2025

Conteúdo

1 Análise de Bad Smells

A análise manual do arquivo `src/ReportGenerator.js` original revelou diversos *Bad Smells*, que são sintomas de problemas mais profundos no design do código. Identificamos três principais:

1.1 Long Method (Método Longo)

O método `generateReport` era o principal culpado. Com mais de 60 linhas, ele violava claramente o Princípio da Responsabilidade Única (SRP).

- **Problema:** O método era responsável por *tudo*: gerar cabeçalhos de dois formatos diferentes, filtrar dados com base em regras de negócio (Admin vs. User), calcular o total e gerar os rodapés.
- **Impacto na Manutenção:** Qualquer alteração, por menor que fosse (ex: adicionar um tipo de relatório "JSON"ou mudar a regra de prioridade do Admin), exigiria modificar esse método monolítico, com alto risco de introduzir bugs em outras partes da lógica.
- **Impacto nos Testes:** Testar todas as combinações (Admin+CSV, Admin+HTML, User+CSV, User+HTML, lista vazia) tornava-se complexo, pois o teste precisava validar a lógica de negócio *e* a formatação da string ao mesmo tempo.

1.2 Repeated Switch (Condicionais Repetidas)

O *smell* mais evidente era a estrutura `if (reportType === 'CSV') ... else if (reportType === 'HTML')`.

- **Problema:** Essa verificação estava duplicada em três lugares: na geração do cabeçalho, na formatação de cada linha dentro do loop e na geração do rodapé.
- **Impacto na Manutenção:** Este é um pesadelo de manutenção. Para adicionar um novo formato de relatório (ex: JSON), o desenvolvedor precisaria "lembrar"de adicionar um `else if` em **todos os três lugares**. Esquecer um deles resultaria em um relatório quebrado ou incompleto.

1.3 Magic Numbers (Números Mágicos)

O código estava repleto de números literais sem explicação semântica.

- **Problema:** Vimos valores como `1000` (na regra de prioridade do Admin) e `500` (na regra de filtro do User) diretamente no código.
- **Impacto na Manutenção:** O que significa `1000`? É um valor máximo? Um limite de prioridade? Se essa regra de negócio precisasse mudar (ex: "agora o limite é 1200"), o desenvolvedor teria que caçar esse número no código, sem ter certeza se outros `1000` em outros arquivos teriam o mesmo significado.

2 Relatório da Ferramenta (ESLint)

Executamos o ESLint com o plugin `eslint-plugin-sonarjs` no código original. O resultado validou quantitativamente a nossa análise manual.

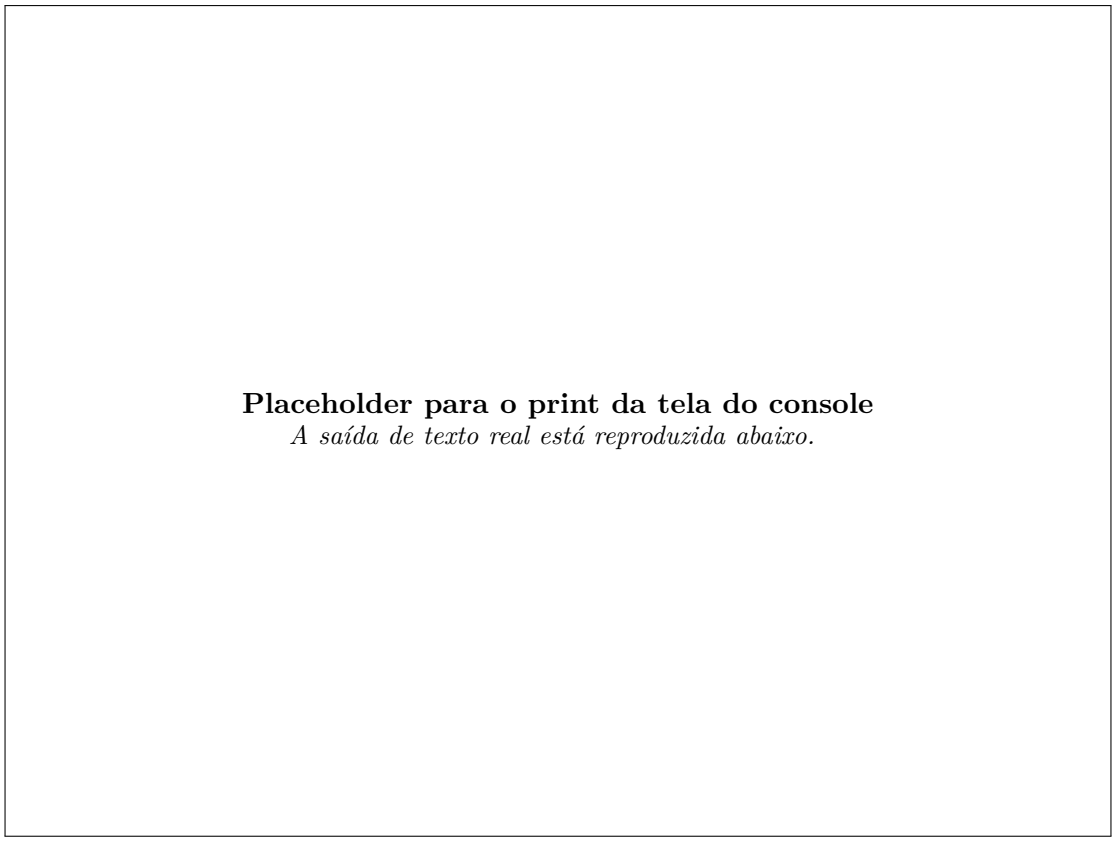

Figura 1: Resultado da execução do comando 'npx eslint src/'.

2.1 Saída do Console

O ESLint reportou dois erros críticos que apontavam diretamente para os *smells* de "Long Method"e "Conditional Hell".

```
C:\...\src\ReportGenerator.js
  11:3  error  Refactor this function to reduce its
               Cognitive Complexity from 27 to the 15 allowed
               sonarjs/cognitive-complexity

  43:14 error  Merge this if statement with the nested one
               sonarjs/no-collapsible-if

? 2 problems (2 errors, 0 warnings)
```

2.2 Análise do Resultado

Enquanto a análise manual foi subjetiva (o método "parece"longo), o SonarJS nos deu dados concretos:

- **Cognitive Complexity (27):** A ferramenta mediu a complexidade do método `generateReport` em 27, quase o dobro do limite aceitável de 15. Isso quantificou objetivamente o *smell* "Long Method"e "Conditional Hell". A alta pontuação deve-se ao excesso de aninhamento de `if` e `else` dentro do loop `for`.
- **No Collapsible If:** A ferramenta foi precisa ao ponto de sugerir a fusão do `if (user.role === 'USER')` com o `if (item.value <= 500)`, confirmando um ponto de complexidade desnecessária.

3 Processo de Refatoração

O *smell* mais crítico corrigido foi o **Repeated Switch**, pois ele era a causa raiz da alta complexidade e da duplicação de código.

Técnica Aplicada: *Replace Conditional with Polymorphism* (Substituir Condicional por Polimorfismo), que é a implementação do **Strategy Pattern**.

A lógica era: ao invés de perguntar *qual* o tipo de relatório e mudar o comportamento (usando `if`), vamos "dizer"ao gerador qual a "estratégia"de formatação (CSV ou HTML) ele deve usar, e delegar a responsabilidade.

3.1 Antes da Refatoração

O método `generateReport` inteiro era uma massa monolítica que misturava lógica de negócio (filtros, totais) com lógica de formatação (strings de CSV/HTML).

```
generateReport(reportType, user, items) {
  let report = '';
  let total = 0;

  // --- Se  o do Cabe alho (Repeti  o 1) ---
  if (reportType === 'CSV') {
    report += 'ID,NOME,VALOR,USUARIO\n';
  } else if (reportType === 'HTML') {
    report += '<html><body>\n';
    report += '<h1>Relat rio</h1>\n';
    // ...
  }

  // --- Se  o do Corpo (Repeti  o 2, Aninhada e Duplicada) ---
  for (const item of items) {
    if (user.role === 'ADMIN') {
      // ... (l gica de prioridade) ...

      if (reportType === 'CSV') {
        report += `${item.id},${item.name},${item.value},${user.name}\n`;
        total += item.value;
      } else if (reportType === 'HTML') {
        const style = item.priority ? ' style="font-weight:bold;"' : "";
        report += `<tr${style}><td>${item.id}</td>...</tr>\n`;
        total += item.value;
      }
    } else if (user.role === 'USER') {
      if (item.value <= 500) { // <-- Magic Number
        if (reportType === 'CSV') {
          report += `${item.id},${item.name},${item.value},${user.name}\n`;
          total += item.value;
        } else if (reportType === 'HTML') {
          report += `<tr><td>${item.id}</td>...</tr>\n`;
          total += item.value;
        }
      }
    }
  }

  // --- Se  o do Rodap  (Repeti  o 3) ---
  if (reportType === 'CSV') {
    report += '\nTotal,,\n';
    report += `${total},,\n`;
  } else if (reportType === 'HTML') {
    report += '</table>\n';
    report += `<h3>Total: ${total}</h3>\n`;
    // ...
```

```
    }

    return report.trim();
  }
```

Listing 1: O método original generateReport (60+ linhas)

3.2 Depois da Refatoração

Criamos "Formatadores"(Strategies) e uma "Factory"para escolhê-los. A classe `ReportGenerator` agora apenas coordena o processo.

```
// Interface (Strategy)
class ReportFormatter {
  generateHeader(user) { /* ... */ }
  generateRow(item, user) { /* ... */ }
  generateFooter(total) { /* ... */ }
}

// Implementa  o Concreta 1
class CsvFormatter extends ReportFormatter {
  generateHeader(user) { return 'ID,NOME,VALOR,USUARIO\n'; }
  // ... (l gica de linha e rodap  movida para c  )
}

// Implementa  o Concreta 2
class HtmlFormatter extends ReportFormatter {
  generateHeader(user) { return '<html><body><h1>Relat rio</h1>...'; }
  // ... (l gica de linha e rodap  movida para c  )
}

// Factory para selecionar a Strategy
class ReportFormatterFactory {
  static getFormatter(reportType) {
    if (reportType === 'CSV') return new CsvFormatter();
    if (reportType === 'HTML') return new HtmlFormatter();
    // ...
  }
}
```

Listing 2: Novas classes de Formatação (Strategy Pattern)

```
  // Constantes definidas no topo do arquivo (sem Magic Numbers)
  const PRIORITY_THRESHOLD = 1000;
  const STANDARD_USER_VALUE_LIMIT = 500;

  generateReport(reportType, user, items) {
    // 1. Seleciona a Strategy
    const formatter = ReportFormatterFactory.getFormatter(reportType);

    // 2. Aplica l gica de neg cio (Extra do para _processItems)
    const processedItems = this._processItems(user, items);

    // 3. Calcula total (Extra do para _calculateTotal)
    const total = this._calculateTotal(processedItems);

    // 4. Delega a formata  o para a Strategy
    let report = '';
    report += formatter.generateHeader(user);

    for (const item of processedItems) {
      report += formatter.generateRow(item, user);
    }
```

```
22
23      report += formatter.generateFooter(total);
24
25      return report.trim();
26    }
27
28    // (Métodos privados _processItems e _calculateTotal omitidos por brevidade)
```

Listing 3: O novo método generateReport (limpo e conciso)

4 Conclusão

Este exercício demonstrou a importância crítica da suíte de testes (nossa "rede de segurança") e os benefícios de qualidade de software ao eliminar *Bad Smells*.

Testes como Rede de Segurança A refatoração realizada foi profunda. Trocamos uma estrutura monolítica de `if/else` por um padrão de design complexo (Strategy/Factory). Seria impossível realizar essa mudança com confiança sem a suíte de testes. A cada alteração, podíamos rodar `npm test` e ter a certeza imediata de que, embora a estrutura interna do código tivesse mudado drasticamente, o comportamento externo (a "saída"do relatório) permanecia 100% correto. Os testes nos deram a coragem para refatorar agressivamente.

Melhoria na Qualidade do Software O código refatorado é objetivamente superior:

- **Legibilidade:** A complexidade cognitiva despencou. O novo `generateReport` lê como uma "receita"do que o relatório faz, sem se perder em detalhes de formatação.
- **Manutenibilidade (SRP):** A classe `ReportGenerator` agora cuida apenas da lógica de negócio. As classes `Formatter` cuidam apenas da formatação.
- **Extensibilidade (OCP):** O código agora obedece ao Princípio Aberto/Fechado (Open/-Closed Principle). Para adicionar um `JSONFormatter`, não precisamos tocar no `ReportGenerator` ou em qualquer formatador existente. Apenas criamos a nova classe e a registramos na Factory. Isso é uma melhoria massiva.