Detecção de Bad Smells e Refatoração Segura

Autor: João Francisco Carvalho Soares de Oliveira Queiroga

Disciplina: Teste de Software

Projeto: Detecção de Bad Smells e Refatoração Segura

Análise de Smells

1. Long Method O método `generateReport()` possuía várias responsabilidades dentro de um único bloco — geração de cabeçalho, corpo, rodapé, controle de acesso e formatação — o que dificultava a leitura e aumentava a complexidade cognitiva. Isso torna o código mais difícil de testar, pois qualquer modificação exige reexecução de testes amplos e interdependentes.

2. Duplicated Code Trechos quase idênticos apareciam repetidos para os diferentes formatos de relatório (CSV e HTML) e perfis de usuário (ADMIN e USER). Essa duplicação gera alto custo de manutenção, pois qualquer mudança de formato ou regra precisa ser replicada em vários pontos.

3. Deeply Nested Conditionals A estrutura de `if` aninhados (role + tipo de relatório + regra de valor) prejudicava a clareza e aumentava o risco de erros lógicos. Esse tipo de *smell* também impacta a testabilidade, pois exige múltiplas combinações de casos de teste para garantir cobertura.

Relatório da Ferramenta

A execução do ESLint antes da refatoração produziu o seguinte resultado:

```
/home/joaoqueiroga/Projects/bad-smells-js-refactoring/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)
```

O `eslint-plugin-sonarjs` destacou dois pontos críticos: **alta complexidade cognitiva** e **condicionais colapsáveis**. Esses problemas nem sempre são perceptíveis na leitura manual, mas a ferramenta usa métricas objetivas para identificar partes do código com lógica excessivamente ramificada, orientando a refatoração com base em qualidade estrutural.

Processo de Refatoração

Trecho escolhido: Método `generateReport()` da classe `ReportGenerator`

Antes:

```
export class ReportGenerator {
  constructor(database) {
    this.db = database;
```

```
  }

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

    if (reportType === "CSV") {
      report += "ID,NOME,VALOR,USUARIO\n";
    } else if (reportType === "HTML") {
      report += "<html><body>\n";
      report += "<h1>Relatório</h1>\n";
      report += `<h2>Usuário: ${user.name}</h2>\n`;
      report += "<table>\n";
      report += "<tr><th>ID</th><th>Nome</th><th>Valor</th></tr>\n";
    }

    for (const item of items) {
      if (user.role === "ADMIN") {
        if (item.value > 1000) {
          item.priority = true;
        }
        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><td>${item.name}</td>
<td>${item.value}</td></tr>\n`;
          total += item.value;
        }
      } else if (user.role === "USER") {
        if (item.value <= 500) {
          if (reportType === "CSV") {
            report += `${item.id},${item.name},${item.value},${user.name}\n`;
            total += item.value;
          } else if (reportType === "HTML") {
            report += `<tr><td>${item.id}</td><td>${item.name}</td><td>${item.value}
</td></tr>\n`;
            total += item.value;
          }
        }
      }
    }

    if (reportType === "CSV") {
      report += "\nTotal,,\n";
      report += `${total},,\n";
    } else if (reportType === "HTML") {
      report += "</table>\n";
      report += `<h3>Total: ${total}</h3>\n`;
      report += "</body></html>\n";
    }
```

```
    return report.trim();
  }
}
```

Depois:

```
export class ReportGenerator {
  constructor(database) {
    this.db = database;
  }

  generateReport(reportType, user, items) {
    let report = this.buildHeader(reportType, user);
    const visibleItems = this.filterItemsByUser(items, user);
    const formattedItems = this.formatItems(visibleItems, reportType, user);
    const total = visibleItems.reduce((sum, item) => sum + item.value, 0);
    report += formattedItems.join("");
    report += this.buildFooter(reportType, total);
    return report.trim();
  }

  buildHeader(type, user) {
    if (type === "CSV") {
      return "ID,NOME,VALOR,USUARIO\n";
    }
    return `<html><body>\n<h1>Relatório</h1>\n<h2>Usuário: ${user.name}
</h2>\n<table>\n<tr><th>ID</th><th>Nome</th><th>Valor</th></tr>\n`;
  }

  buildFooter(type, total) {
    if (type === "CSV") {
      return `\nTotal,,\n${total},,\n`;
    }
    return `</table>\n<h3>Total: ${total}</h3>\n</body></html>\n`;
  }

  filterItemsByUser(items, user) {
    if (user.role === "ADMIN") return items.map(i => ({ ...i, priority: i.value >
1000 }));
    return items.filter(i => i.value <= 500);
  }

  formatItems(items, type, user) {
    return items.map(item => {
      if (type === "CSV") {
        return `${item.id},${item.name},${item.value},${user.name}\n`;
      }
      const style = item.priority ? ' style="font-weight:bold;"' : "";
      return `<tr${style}><td>${item.id}</td><td>${item.name}</td><td>${item.value}
</td></tr>\n`;
```

```
    });
  }
}
```

Técnicas aplicadas:

- *Extract Method*: Separação das partes de cabeçalho, corpo e rodapé.
- *Decompose Conditional*: Divisão da lógica condicional complexa em métodos menores.
- *Remove Duplicated Code*: Eliminação de repetições entre formatos de relatório.

Conclusão

A refatoração reduziu a complexidade cognitiva de 27 para menos de 15 pontos, conforme apontado pelo ESLint. O código ficou mais legível e modular, permitindo a escrita de testes independentes para cada parte do processo de geração. O uso de testes automatizados garantiu segurança durante a refatoração, servindo como uma rede de proteção contra regressões. A diminuição de *code smells* resultou em um software mais confiável, sustentável e de fácil manutenção.