Relatório de Análise e Refatoração de Bad Smells

Análise de Smells

1. Alta Complexidade Cognitiva

    O método generateReport tinha complexidade 27 (limite 5)

    Múltiplos níveis de aninhamento de ifs

    Dificulta o entendimento do fluxo de código e aumenta chances de bugs

    Torna os testes mais complexos pois há muitos caminhos a testar

2. Falta de Coesão / Múltiplas Responsabilidades

    Método generateReport mistura lógicas de:

    - Formatação (HTML/CSV)
    - Regras de negócio (filtro por usuário)
    - Cálculos (total)

    Viola o princípio Single Responsibility

    Dificulta reuso e manutenção

3. Duplicação de Código

    Lógica de soma do total repetida em vários blocos

    Formatação de dados repetida para CSV e HTML

    Aumenta chance de erros em manutenções

    Requer mudanças em múltiplos lugares para correções

Relatório da Ferramenta

```
/Users/educbank/my-projects/bad-smells-js-refactoring/src/ReportGenerator.js
  11:3   error  Refactor this function to reduce its Cognitive Complexity from 27 to the 5 allowed  sonarjs/co
  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 foi crucial para:

Identificar precisamente a complexidade cognitiva (27)

Detectar ifs aninhados que poderiam ser consolidados

Fornecer métricas objetivas que guiaram a refatoração

Processo de Refatoração

O smell mais crítico era a alta complexidade cognitiva. Aqui está a transformação:

Antes:

```
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';
    // ... mais HTML
  }

  for (const item of items) {
    if (user.role === 'ADMIN') {
      if (item.value > 1000) {
        item.priority = true;
      }
      if (reportType === 'CSV') {
        // ... formatação CSV
      } else if (reportType === 'HTML') {
        // ... formatação HTML
      }
    } else if (user.role === 'USER') {
      if (item.value <= 500) {
        if (reportType === 'CSV') {
          // ... formatação CSV
        } else if (reportType === 'HTML') {
          // ... formatação HTML
        }
      }
    }
  }
  // ... rodapé
}
```

Depois:

```
class ReportStrategy {
  generateHeader(user) { /*...*/ }
  generateRow(item, user, isPriority) { /*...*/ }
  generateFooter(total) { /*...*/ }
}

class ItemFilter {
  static filterByUserRole(items, user) {
    if (user.role === 'ADMIN') {
      return items.map(item => ({
        ...item,
        priority: item.value > 1000
      }));
    }
    return items.filter(item => item.value <= 500);
  }
}

generateReport(reportType, user, items) {
  const strategy = this.strategies[reportType];
  const filteredItems = ItemFilter.filterByUserRole(items, user);

  let report = strategy.generateHeader(user);
  let total = 0;

  for (const item of filteredItems) {
    report += strategy.generateRow(item, user, item.priority);
    total += item.value;
  }

  return (report + strategy.generateFooter(total)).trim();
}
```

Técnicas aplicadas:

1. Strategy Pattern: Para separar diferentes formatos de relatório
2. Extract Class: Criação de ItemFilter para regras de negócio
3. Template Method: Para estrutura comum de geração de relatórios

Conclusão

Os testes foram fundamentais durante a refatoração, servindo como rede de segurança para garantir que o comportamento do código permanecesse igual mesmo com mudanças significativas na estrutura. A redução dos bad smells resultou em:

- Código mais fácil de entender e manter
- Melhor separação de responsabilidades
- Maior facilidade para adicionar novos formatos de relatório
- Menor risco de bugs em manutenções futuras

A combinação de análise automática (ESLint/SonarJS) com testes robustos permitiu uma refatoração segura e efetiva, resultando em um código mais limpo e sustentável.