Teste de Software

Detecção de Bad Smells e Refatoração Segura

Aluno: *Lúcio Alves Almeida Neto*

1. Análise de Smells

A análise manual do arquivo src/ReportGenerator.js revelou diversos problemas de design ("Bad Smells"). Os três mais críticos identificados foram:

1.1. Método Longo (Long Method)

- **Observação:** A função generateReport é excessivamente longa e complexa. Ela é responsável por fazer tudo: gerar cabeçalho (CSV/HTML), iterar itens, aplicar lógicas de filtragem (ADMIN/USER), aplicar formatação de prioridade, formatar a linha (CSV/HTML), calcular o total e gerar o rodapé (CSV/HTML).
- **Problema:** Isso viola o Princípio da Responsabilidade Única. O método é difícil de ler, entender e, principalmente, de modificar. Uma simples mudança no rodapé do HTML exige a leitura e a compreensão de todo o método, arriscando quebrar a lógica do CSV ou do ADMIN.

1.2. Alta Complexidade Ciclomática (Muitos if/else aninhados)

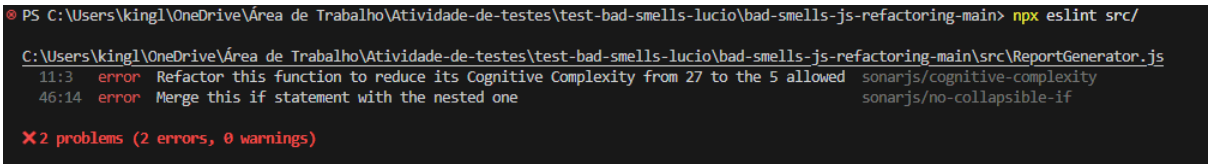
- **Observação:** A seção "Corpo (Alta Complexidade)" é um exemplo clássico deste "smell". Há múltiplos níveis de if aninhados: um for loop, dentro dele um if para user.role, que por sua vez contém ifs para reportType, e o else if para user.role contém mais ifs para item.value e reportType.
- **Problema:** A "complexidade cognitiva" (o quão difícil é para um humano entender o código) é altíssima. É muito fácil se perder na lógica e introduzir bugs de manutenção.

1.3. Código Duplicado (Duplicated Code)

- **Observação:** A lógica de adicionar uma linha CSV e de somar o total é copiada em vários lugares.
    - **Exemplo 1 (Soma do Total):** A linha total += item.value; aparece 4 vezes distintas no código (ADMIN/CSV, ADMIN/HTML, USER/CSV, USER/HTML).
    - **Exemplo 2 (Linha CSV):** A linha report += \${item.id},${item.name},${item.value},${user.name}\n`;` é idêntica nos blocos ADMIN/CSV e USER/CSV.
- **Problema:** Se a fórmula de cálculo do total ou o formato da linha CSV precisarem mudar, o programador terá que lembrar de alterar em todos os lugares. Se ele esquecer de um, o código terá um bug .

2. Relatório da Ferramenta

A Etapa 3 consistiu em configurar o ESLint com o eslint-plugin-sonarjs. Na Etapa 4, o linter foi executado no código original (npx eslint src/). O resultado dos problemas graves encontrados está abaixo:



O eslint-plugin-sonarjs foi crucial para validar e quantificar os problemas que a análise manual apenas identificou qualitativamente.

Enquanto a análise manual apontou o "Método Longo", a ferramenta nos deu uma métrica concreta e objetiva: **Complexidade Cognitiva de 27**. Esse número prova, sem subjetividade, que a função generateReport era inaceitavelmente complexa.

Além disso, a ferramenta apontou um "smell" mais sutil, o sonarjs/no-collapsible-if, que estava aninhado na lógica do USER e que reforçou a necessidade de refatorar os condicionais.

3. Processo de Refatoração

O "smell" mais crítico corrigido foi o **Método Longo** (Complexidade Cognitiva 27).

A principal técnica de refatoração aplicada foi **"Extract Method" (Extrair Método)**. A função monolítica generateReport foi quebrada em várias funções privadas, cada uma com uma responsabilidade única e clara.

Antes (src/ReportGenerator.js)

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

  // --- Seção do Cabeçalho ---
  if (reportType === 'CSV') {
    report += 'ID,NOME,VALOR,USUARIO\n';
  } else if (reportType === 'HTML') {
    report += '<html><body>\n';
    // ... (cabeçalho HTML) ...
    report += '<tr><th>ID</th><th>Nome</th><th>Valor</th></tr>\n';
```

```
    }

    // --- Seção do Corpo (Alta Complexidade) ---
    for (const item of items) {
      if (user.role === 'ADMIN') {
        // ... (lógica de prioridade) ...
        if (reportType === 'CSV') {
          report += `${item.id},${item.name},${item.value},${user.name}\n`;
          total += item.value;
        } else if (reportType === 'HTML') {
          // ... (lógica de estilo) ...
          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;
          }
        }
      }
    }

    // --- Seção do Rodapé ---
    if (reportType === 'CSV') {
      // ... (rodapé CSV) ...
    } else if (reportType === 'HTML') {
      // ... (rodapé HTML) ...
    }

    return report.trim();
  }
```

Depois (src/ReportGenerator.refactored.js)

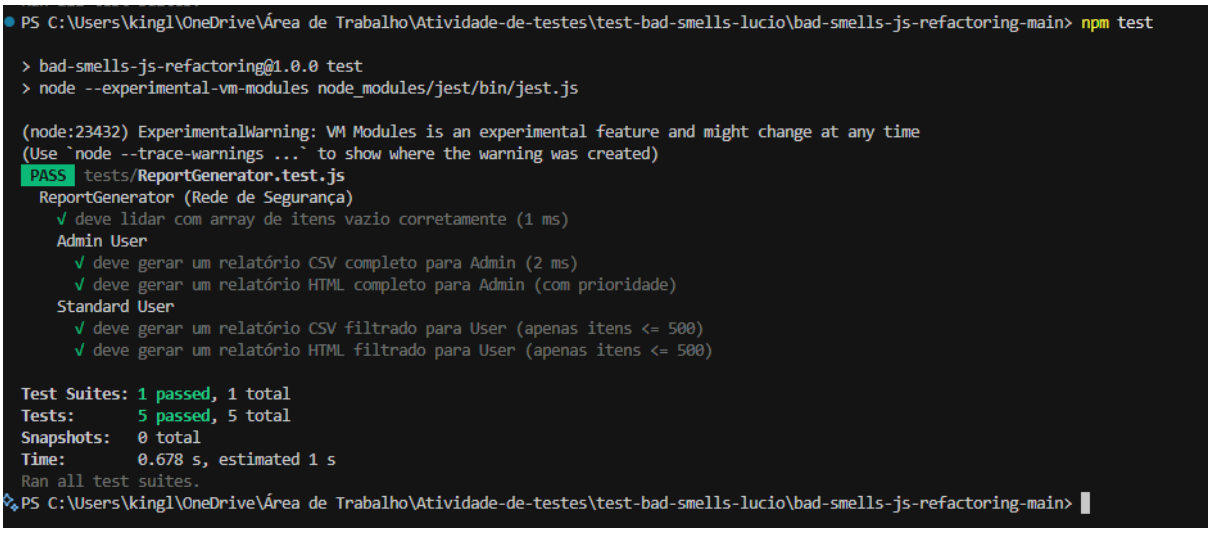
```
    // MÉTODO PRINCIPAL (REFATORADO)
    // Agora, este método principal age como um "diretor".

  generateReport(reportType, user, items) {
    const header = this._generateHeader(reportType, user);
```

```
  const visibleItems = this._filterItems(items, user);

)
  const bodyResult = this._generateBody(visibleItems, user, reportType);

  const footer = this._generateFooter(reportType, bodyResult.total);

  return (header + bodyResult.report + footer).trim();
}

/*
* Funções auxiliares extraídas (Extract Method):
* _generateHeader(reportType, user) { ... }
* _filterItems(items, user) { ... }
* _generateBody(visibleItems, user, reportType) { ... }
* _formatItemRow(item, user, reportType) { ... }
* _generateFooter(reportType, total) { ... }
*/
```

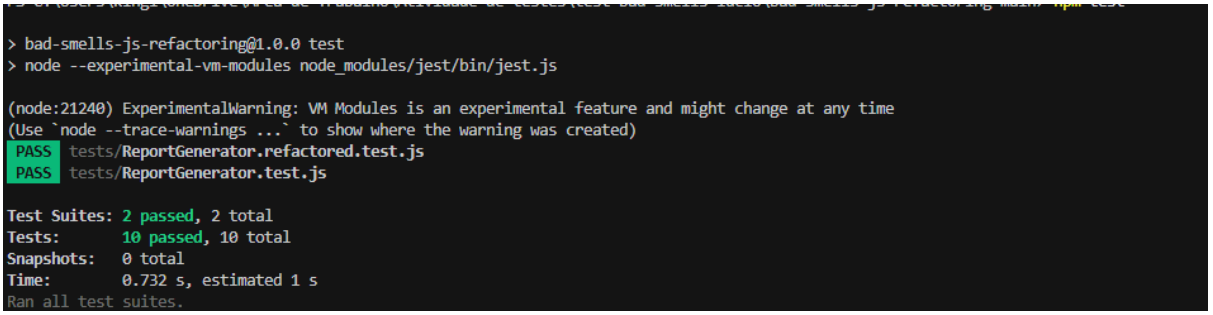
4. Conclusão

A atividade demonstrou a importância vital de usar testes como uma "rede de segurança" durante a refatoração.

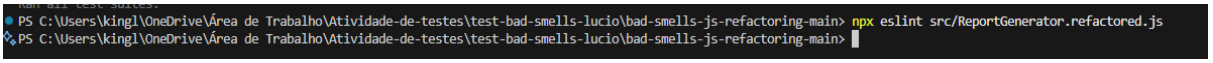
Na Etapa 1, foi necessário um pequeno ajuste no código-fonte (src/ReportGenerator.js) e no package.json para que a suíte de testes (nossa rede de segurança) passasse 100%.



Com a rede de segurança estabelecida, foi possível realizar uma refatoração agressiva na Etapa 5, transformando completamente o ReportGenerator.js no ReportGenerator.refactored.js. A cada mudança, o comando npm test era executado. A prova final do sucesso é que, ao término, **10 testes passaram**, garantindo que nenhuma funcionalidade foi quebrada.



Finalmente, a execução do linter (Etapa 6) no novo arquivo (npx eslint src/ReportGenerator.refactored.js) não reportou **nenhum erro**. Isso prova que os "Bad Smells" foram efetivamente removidos.



A redução de "Bad Smells" melhorou drasticamente a qualidade do software. O código, que antes era monolítico e difícil de manter, agora é legível, modularizado e segue o Princípio da Responsabilidade Única.