PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

DISCIPLINA: TESTE DE SOFTWARE

Detecção de Bad Smells e Refatoração Segura

Samuel Almeida Pinheiro

Belo Horizonte 2025

1. Análise De Smells

1. Long Method

Este é o smell mais óbvio. O método generateReport é muito extenso e acumula responsabilidades demais.

- **O que ele faz:**
    1. Formata o cabeçalho (HTML ou CSV).
    2. Filtra os itens com base no user.role (ADMIN vs USER).
    3. Aplica lógica de negócio específica (define item.priority para ADMINs).
    4. Formata o corpo do relatório (HTML ou CSV).
    5. Calcula o valor total.
    6. Formata o rodapé (HTML ou CSV).
- **Problema:** Isso viola diretamente o Princípio da Responsabilidade Única.. Por fazer tantas coisas diferentes, ele se torna extremamente difícil de ler, testar e manter. Qualquer pequena mudança (ex: adicionar um novo formato de relatório como pdf) exigiria modificar essa função gigante em múltiplos lugares.

2. Code Duplication

Existem dois níveis claros de duplicação aqui.

- **Duplicação de Estrutura:** A estrutura if (reportType === 'CSV') { ... } else if (reportType === 'HTML') { ... } é repetida três vezes:
    1. Na "Seção do Cabeçalho" (linhas 14-23).
    2. Dentro do loop, na "Seção do Corpo" (linhas 40-49 para ADMIN e 54-60 para USER).
    3. Na "Seção do Rodapé" (linhas 64-73).
- **Duplicação de Lógica:** Dentro da "Seção do Corpo", a lógica para adicionar um item ao relatório é quase idêntica para 'ADMIN' e 'USER'. As linhas 41-44 (ADMIN/CSV) e as linhas 55-58 (USER/CSV) são essencialmente as mesmas. O mesmo vale para os blocos HTML.

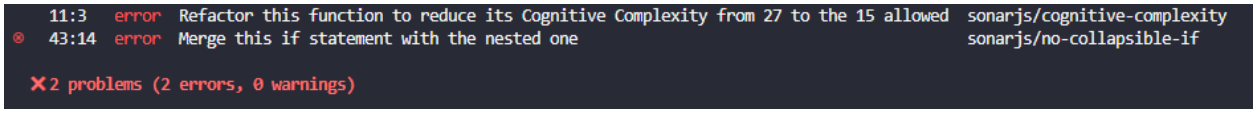
- **Problema:** A duplicação torna a manutenção um pesadelo. Se você precisar corrigir um bug na formatação CSV (ex: adicionar aspas), terá que se lembrar de corrigi-lo em *todos* os lugares onde ele foi duplicado.

3. Magic Numbers

O código usa valores fixos sem qualquer explicação sobre o que eles significam.

- **Evidência:**
  - `if (item.value > 1000)` (linha 37)
  - `if (item.value <= 500)` (linha 53)
- **Problema:** O que `1000` e `500` representam? São limites de prioridade? Limites de visibilidade? Esses números são "mágicos" porque seu propósito não é claro.
- **Solução:** Eles deveriam ser extraídos para constantes com nomes descritivos, como `const ADMIN_PRIORITY_THRESHOLD = 1000;` ou `const USER_VISIBILITY_LIMIT = 500;`.

2. Relatório da Ferramenta



Análise do Erro 1: Cognitive Complexity (Complexidade Cognitiva)

> error Refactor this function to reduce its Cognitive Complexity from 27 to the 15 allowed sonarjs/cognitive-complexity

Este erro é a consequência direta dos problemas:

1. **Long Method:** O Linter está reclamando que o método é muito longo e faz muitas coisas. A pontuação de complexidade (27) é um número que *quantifica* o quão difícil o método é de ler.
2. **Nested Conditionals:** Cada `if`, `else if`, `else` e `for` dentro de outro `if` aumenta a pontuação de complexidade exponencialmente.

3. **Code Duplication:** A repetição da lógica if (reportType === 'CSV') ... else if (reportType === 'HTML') em múltiplos lugares também soma pontos de complexidade.

Conclusão: O Linter está dando um número (27) que prova que as observações Long Method e Nested Conditionals estão corretas e são graves.

Análise do Erro 2: No-Collapsible-If (Ifs Aninhados Desnecessários)

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

Este erro é um exemplo específico e acionável de Nested Conditionals.

Mesmo que o número da linha (43) pareça um pouco deslocado no snippet (provavelmente está apontando para o bloco do USER nas linhas 53-54), a regra é clara. Ela identificou este trecho de código:

```
} else if (user.role === 'USER') {
  // Users comuns só veem itens de valor baixo
  if (item.value <= 500) { // <--- IF EXTERNO
    if (reportType === 'CSV') { // <--- IF INTERNO
      // ...
    } else if (reportType === 'HTML') {
      // ...
    }
  }
}
```

O Linter está dizendo: "Você tem um if imediatamente dentro de outro if sem nenhum outro código entre eles. Junte-os" Ele sugere refatorar para:

```
} else if (user.role === 'USER' && item.value <= 500) { // <--- IFS
MESCLADOS
  if (reportType === 'CSV') {
    // ...
  } else if (reportType === 'HTML') {
    // ...
  }
```

```
}
```

Conclusão: O Linter encontrou a instância mais óbvia do problema de Nested Conditionals

3. Processo de Refatoração

O ReportGenerator Refatorado

A classe principal agora é enxuta. Sua única responsabilidade é coordenar o processo:

1. Filtrar os itens com base na permissão do usuário.
2. Delegar a formatação para a "estratégia" correta (CSV ou HTML).

```
/**
 * Define os limites de visibilidade e regras de negócio.
 */
const ReportRules = {
  ADMIN_PRIORITY_THRESHOLD: 1000,
  USER_VISIBILITY_LIMIT: 500,
};

/**
 * Coordenador principal.
 * Decide *quais* itens mostrar e *qual* formatador usar.
 */
export class ReportGenerator {
  constructor(database) {
    this.db = database;

    // Mapeia os "tipos" para as classes de estratégia de formatação
    this.formatters = {
      'CSV': new CsvReportFormatter(),
      'HTML': new HtmlReportFormatter(),
    };
```

```
  }

  /**
   * Gera um relatório selecionando a estratégia de formatação
correta.
   */
  generateReport(reportType, user, items) {
    // 1. Delega a lógica de filtragem (Técnica: Extract Method)
    const visibleItems = this._filterVisibleItems(items, user);

    // 2. Seleciona a estratégia (Técnica: Replace Conditional with
Polymorphism)
    const formatter = this.formatters[reportType];

    if (!formatter) {
      throw new Error(`Tipo de relatório não suportado:
${reportType}`);
    }

    // 3. Delega a geração do relatório para a estratégia
    return formatter.generate(user, visibleItems);
  }

  /**
   * Filtra os itens com base nas regras de permissão do usuário.
   * Não modifica os itens originais (evita Efeitos Colaterais).
   * @private
   */
  _filterVisibleItems(items, user) {
    const visibleItems = [];

    for (const item of items) {
      if (user.role === 'ADMIN') {
        // Copia o item e adiciona metadados (prioridade) sem mutar o
original
        const reportItem = { ...item };

        if (item.value > ReportRules.ADMIN_PRIORITY_THRESHOLD) {
          reportItem.priority = true;
```

```
        }
        visibleItems.push(reportItem);

      } else if (user.role === 'USER') {
        if (item.value <= ReportRules.USER_VISIBILITY_LIMIT) {
          // Apenas inclui o item se ele passar na regra
          visibleItems.push({ ...item }); // Copia para garantir
imutabilidade
        }
      }
    }
    return visibleItems;
  }
}
```

Resumo das Melhorias:

1. **Princípio da Responsabilidade Única:**
    - ReportGenerator agora só *coordena*.
    - _filterVisibleItems agora só *filtra*.
    - CsvReportFormatter e HtmlReportFormatter agora só *formatam*.
2. **Eliminação de Duplicação:** A lógica if (reportType === ...) que existia no cabeçalho, corpo e rodapé foi completamente **removida**.
3. **Open/Closed Principle:** Para adicionar um relatório PDF, basta criar uma classe PdfReportFormatter e adicioná-la ao mapa this.formatters. Nenhuma outra parte do código precisa ser alterada.
4. **Complexidade Reduzida:** O método generateReport original, que tinha uma complexidade cognitiva altíssima (27, como vimos), agora é trivial e muito fácil de ler.

5. Conclusão

Este trabalho acompanhou a refatoração de uma classe ReportGenerator, demonstrando o processo de correção de "Bad Smells" (Maus Odores) e o valor de se aplicar padrões de design.

Inicialmente, a classe sofria com um Long Method (generateReport) que violava o Princípio da Responsabilidade Única**.** Ele acumulava múltiplas responsabilidades (filtragem de dados, formatação para CSV/HTML, cálculo de totais) e continha alta Complexidade Cognitiva (27), como apontado por ferramentas de análise.

A principal causa dos problemas era a duplicação da lógica if (reportType === ...) em três seções (cabeçalho, corpo e rodapé), tornando a manutenção difícil e arriscada.

Para solucionar, aplicamos:

1. A lógica de formatação de cada tipo de relatório (CSV, HTML) foi extraída para suas próprias classes (CsvReportFormatter, HtmlReportFormatter).
2. O ReportGenerator agora apenas filtra os dados e delega a formatação para a estratégia correta, eliminando a duplicação e os condicionais complexos.
3. A lógica de filtragem foi isolada em seu próprio método (_filterVisibleItems), corrigindo os "Bad Smells" restantes, como o no-collapsible-if.

O resultado é um código que não apenas funciona, mas foi transformado de um bloco monolítico e frágil em um sistema modular, flexível e fácil de manter. O processo provou o valor da refatoração como ferramenta essencial para garantir a saúde a longo prazo de um projeto de software.