Trabalho Prático – Detecção de Bad Smells e Refatoração Segura

Aluno: Victor Reis Carlota **Disciplina:** Engenharia de Software **Professor:** *(preencher)* **Data:** Novembro/2025

---

1. Introdução

Este trabalho tem como objetivo aplicar práticas de **detecção e correção de *bad smells*** em código JavaScript, utilizando o **ESLint com o plugin SonarJS** e aplicando refatorações seguras sem alterar o comportamento do sistema.

O arquivo analisado foi `src/ReportGenerator.js`, responsável por gerar relatórios CSV e HTML com base em permissões de usuário e lista de itens. O comportamento foi validado através da suíte de testes automatizados fornecida no repositório.

---

2. Análise de Bad Smells

A análise foi realizada em duas etapas: **manual** (observação do código) e **automática** (usando ESLint + sonarjs).

2.1 Detecção Manual

Durante a inspeção do arquivo `ReportGenerator.js`, foram identificados os seguintes *bad smells*:

Tipo	Localização	Descrição
Método Longo (Long Method)	Função `generateReport()`	A função possui várias responsabilidades: gerar cabeçalho, iterar itens, aplicar filtros, somar valores e montar rodapé.
Condicionais Aninhadas (Nested Conditionals)	Dentro do loop `for`	Estruturas `if/else` duplicadas para perfis `ADMIN` e `USER`, tornando o código extenso e de difícil leitura.
Duplicação de Código (Duplicate Code)	Blocos para CSV e HTML	As mesmas operações são repetidas com pequenas variações de formato.

Esses smells tornam o código **difícil de entender, testar e manter**, além de aumentar a chance de erros ao realizar alterações futuras.

2.2 Detecção Automática com ESLint

A ferramenta de análise estática reportou os seguintes problemas:

```
C:\code\bad-smells-js-refactoring\src\ReportGenerator.js
  11:3   error  Refactor this function to reduce its Cognitive Complexity from 27 to the 5 a
  43:14  error  Merge this if statement with the nested one
```

O primeiro erro indica que a função principal possui uma **complexidade cognitiva** 5 vezes maior que o limite recomendado, enquanto o segundo sugere que há **ifs aninhados desnecessários**.

Essas evidências confirmam os smells encontrados manualmente.

---

3. Refatoração Segura

Com base nas evidências, foram aplicadas as seguintes técnicas de refatoração:

Após a refatoração, o arquivo ReportGenerator.refactored.js passou em 100% dos testes automatizados, enquanto o código original continuou apresentando uma falha de formatação esperada. Isso demonstra que a refatoração preservou o comportamento funcional e corrigiu o problema estrutural, melhorando a clareza e a manutenção do código sem alterar sua lógica de negócio.

- **Extract Method:** extração de partes do código em funções menores e nomeadas.
- **Replace Nested Conditional with Guard Clauses:** simplificação de blocos condicionais.