Detecção de Bad Smells e Refatoração Segura

1. Introdução e Contexto

Um Bad Smell (mau cheiro no código), popularizado por Martin Fowler, é um sintoma no código-fonte que indica um problema de design mais profundo. Smells como Método Longo, Classe Grande ou Código Duplicado tornam o software difícil de entender, modificar e, crucialmente, testar.

Neste trabalho, vocês atuarão como arquitetos de software. Usaremos uma suíte de testes já existente (e confiável) como nossa rede de segurança. O objetivo é detectar Bad Smells usando análise estática e refatorar o código de produção, provando que nossas melhorias não introduziram regressões.

2. Objetivos de Aprendizagem

Ao final deste trabalho, o aluno deverá ser capaz de:

- Identificar manualmente Bad Smells clássicos (ex: Método Longo, Complexidade Ciclomática Alta, Duplicação de Lógica).
- Configurar e utilizar ferramentas de análise estática para detecção de smells de código em JavaScript.
- **Proposta de Ferramenta:** Utilizar o **ESLint** em conjunto com o plugin **eslint-plugin-sonarjs**, focado em qualidade de código e detecção de "code smells".
- Interpretar relatórios de complexidade cognitiva, duplicação e manutenção.
- Aplicar técnicas de refatoração (ex: *Extract Method*, *Replace Conditional with Polymorphism*) para corrigir os smells.
- Utilizar uma suíte de testes existente como rede de segurança para validar o processo de refatoração.

3. O Projeto Base

Para este trabalho, utilizaremos um projeto que simula um serviço de geração de relatórios. O código de produção foi deliberadamente escrito com vários Bad Smells.

- **Repositório do Projeto:** https://github.com/CleitonSilvaT/bad-smells-js-refactoring
- **O repositório contém:**
  - Um diretório src/ com a lógica de negócio mal cheirosa (ex: ReportGenerator.js).
  - Um diretório __tests__/ com uma suíte de testes robusta e completa (ReportGenerator.test.js). Os testes não devem ser alterados! Eles são sua garantia de que a refatoração foi bem-sucedida.
  - Um arquivo package.json com as dependências (ex: Jest).

4. Etapas do Trabalho

Etapa 1: Preparação do Ambiente e Rede de Segurança

1. Faça um **fork** do repositório para sua conta do GitHub e clone-o.
2. Instale as dependências: npm install.
3. Execute a suíte de testes: npm test.
4. **Validação:** Confirme que 100% dos testes passam. Esta é sua rede de segurança. Se os testes quebrarem durante a Etapa 5, você saberá que introduziu um bug.

Etapa 2: Análise Manual (Caça aos Smells)

1. Abra o arquivo src/ReportGenerator.js.
2. Leia atentamente o código. Tente identificar manualmente pelo menos 3 tipos diferentes de Bad Smells (ex: Métodos longos, Muitos if/else ou switch, Números Mágicos, Duplicação de código).
3. Anote suas observações.

Etapa 3: Configuração da Ferramenta de Detecção (ESLint + SonarJS)

1. Adicione as dependências de linting ao projeto:
   `npm install --save-dev eslint eslint-plugin-sonarjs`

2. Crie o arquivo de configuração `.eslintrc.json` na raiz do projeto (similar ao trabalho anterior, mas com foco diferente).

3. Adicione o seguinte conteúdo, ativando as regras recomendadas do SonarJS:

```
{
  "env": {
    "es2021": true,
    "node": true,
    "jest": true
  },
  "extends": [
    "eslint:recommended",
    "plugin:sonarjs/recommended"
  ],
  "parserOptions": {
    "ecmaVersion": "latest",
    "sourceType": "module"
  },
  "plugins": [
    "sonarjs"
  ],
  "rules": {
    // O plugin "sonarjs/recommended" já ativa regras cruciais
```

```
    // como complexidade cognitiva e duplicação.
    "sonarjs/cognitive-complexity": ["error", 15],
    "sonarjs/no-identical-functions": "error"
  }
}
```

Etapa 4: Detecção Automática e Análise

1. Execute o ESLint no seu terminal para analisar o código de produção:
   `npx eslint src/`

2. Analise o resultado. O linter deve reportar vários problemas no arquivo src/ReportGenerator.js, como alta complexidade cognitiva (cognitive-complexity) ou duplicação de lógica.
3. Compare os problemas encontrados pela ferramenta com suas anotações manuais.

Etapa 5: Refatoração Segura (O Desafio)

1. **Não modifique o arquivo original ainda.** Crie uma cópia: src/ReportGenerator.refactored.js.
2. Seu objetivo é refatorar o código neste *novo* arquivo para que ele fique limpo e sem os smells apontados.
   - Quebre métodos longos (Técnica: *Extract Method*).
   - Simplifique condicionais complexas (Técnica: *Decompose Conditional* ou *Replace Conditional with Polymorphism*).
   - Elimine duplicação.
3. **Validação Contínua:** Para validar seu novo arquivo, crie uma cópia do arquivo de teste (ex: __tests__/ReportGenerator.refactored.test.js) e mude apenas o import no topo do arquivo, para que ele teste o seu ReportGenerator.refactored.js.
4. Execute os testes (npm test) a cada refatoração. Eles devem continuar passando 100% do tempo.

Etapa 6: Validação Final

1. Execute o ESLint novamente, agora apontando para o seu arquivo refatorado:
   `npx eslint src/ReportGenerator.refactored.js`
2. O linter não deve apontar mais nenhum erro grave de smell (ou deve haver uma redução drástica).
3. Execute a suíte de testes completa (npm test) para garantir que ambos os arquivos (o antigo e o novo) passam, provando que sua refatoração não quebrou a funcionalidade.
4. Faça o commit e o push das suas alterações (incluindo os novos arquivos .refactored.js).

5. O Que Entregar

1. **Link do Repositório GitHub:** O link para o seu fork do projeto, contendo os novos arquivos src/ReportGenerator.refactored.js e __tests__/ReportGenerator.refactored.test.js.
2. **Relatório Escrito (PDF, 2-4 páginas):** Um documento contendo:
    - **Capa:** Nome da disciplina, nome do trabalho, seu nome completo e matrícula.
    - **Análise de Smells:** Descreva 3 Bad Smells que você encontrou no código original[6]. Explique por que são problemáticos para a manutenção e para os testes.
    - **Relatório da Ferramenta:** Inclua um *print* do resultado do eslint src/ (antes da refatoração). Comente como o eslint-plugin-sonarjs ajudou a identificar problemas que a análise manual poderia ter perdido (ex: complexidade cognitiva).
    - **Processo de Refatoração:** Escolha o *smell* mais crítico que você corrigiu. Mostre o Antes e o Depois do código. Explique a(s) técnica(s) de refatoração que você aplicou (ex: Usei Extract Method para...).
    - **Conclusão:** Uma breve reflexão sobre a importância de usar testes como rede de segurança durante a refatoração e como a redução de Bad Smells melhora a qualidade geral do software.

6. Critérios de Avaliação

- **Análise e Identificação de Smells (20%):** A capacidade de analisar e descrever corretamente os problemas no código de produção.
- **Configuração da Ferramenta (15%):** O eslint-plugin-sonarjs foi configurado e executado corretamente.
- **Qualidade da Refatoração (40%):** O código foi efetivamente melhorado, os smells foram removidos e o design está mais limpo e legível.
- **Uso da Rede de Segurança (15%):** O aluno demonstrou que os testes continuaram passando, provando uma refatoração segura.
- **Qualidade do Relatório Escrito (10%):** Clareza, objetividade e cumprimento de todos os itens solicitados.

7. Recursos Adicionais

- **Plugin eslint-plugin-sonarjs (Documentação):** https://github.com/SonarSource/eslint-plugin-sonarjs
- **Catálogo de Refatoração (Martin Fowler):** https://refactoring.com/catalog/
- **Livro (Fonte Canônica):** FOWLER, Martin. *Refatoração: Aperfeiçoando o Design de Códigos Existentes*. 2. ed. Novatec, 2019.