Relatório de Detecção de Bad Smells e Refatoração Segura

Disciplina: Teste de Software

Trabalho: Trabalho Test Pattern

Aluno: Matheus Hoske Aguiar

Matrícula: 728000

Repositório GitHub: https://github.com/matheushoske/bad-smells-js-refactoring

1. ANÁLISE DE SMELLS

Durante a análise manual e automática do código em `src/ReportGenerator.js`, foram identificados três Bad Smells principais que comprometem a manutenibilidade e testabilidade do código:

1.1 Método Longo (Long Method)

O método `generateReport` possui 68 linhas e concentra múltiplas responsabilidades: geração de cabeçalho, processamento de itens, formatação de linhas e geração de rodapé. Este smell é problemático porque:

- **Dificulta a compreensão:** Um desenvolvedor precisa ler todo o método para entender qualquer parte da lógica
- **Dificulta os testes:** Testar cada responsabilidade isoladamente torna-se impossível, exigindo testes complexos que verificam múltiplos comportamentos simultaneamente
- **Aumenta o risco de bugs:** Mudanças em uma parte do método podem afetar outras partes não relacionadas
- **Reduz a reutilização:** Lógica útil (como formatação de linhas) não pode ser reutilizada em outros contextos

1.2 Complexidade Cognitiva Alta (High Cognitive Complexity)

O método `generateReport` apresenta complexidade cognitiva de 27, muito acima do limite recomendado de 15. A complexidade é causada por:

- Múltiplos níveis de aninhamento de condicionais (if/else dentro de if/else)
- Lógica duplicada para diferentes tipos de relatório (CSV e HTML)
- Condicionais aninhadas para diferentes roles de usuário (ADMIN e USER)
- Condicionais adicionais para filtragem de itens

Este smell é crítico porque:

- **Dificulta a manutenção:** Adicionar novos tipos de relatório ou roles requer modificar múltiplas partes do código
- **Aumenta a probabilidade de erros:** A lógica complexa torna fácil introduzir bugs ao fazer modificações
- **Dificulta os testes:** Cada combinação de condições precisa ser testada, aumentando exponencialmente o número de casos de teste necessários
- **Reduz a legibilidade:** O código se torna difícil de ler e entender, especialmente para novos desenvolvedores

1.3 Duplicação de Código (Code Duplication)

O código apresenta duplicação significativa na lógica de formatação de itens. A mesma estrutura de formatação é repetida para:

- Admin com CSV (linha 36)
- Admin com HTML (linha 40)
- User com CSV (linha 47)
- User com HTML (linha 50)

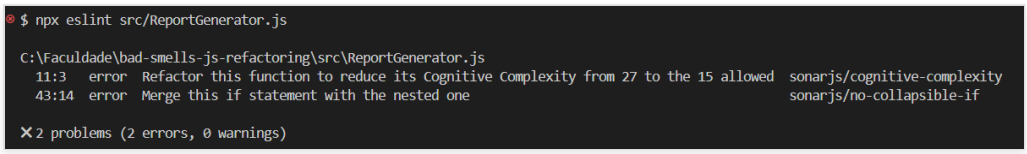
Além disso, a lógica de cálculo do total (`total += item.value`) é repetida em quatro lugares diferentes.

Este smell é problemático porque:

- **Violation do princípio DRY (Don't Repeat Yourself):** Mudanças na lógica de formatação precisam ser feitas em múltiplos lugares
- **Aumenta o risco de inconsistências:** É fácil esquecer de atualizar uma das cópias, resultando em comportamento inconsistente
- **Dificulta os testes:** Cada duplicação precisa ser testada separadamente, aumentando a carga de manutenção dos testes
- **Aumenta o tamanho do código:** Código duplicado aumenta desnecessariamente o tamanho da base de código

2. RELATÓRIO DA FERRAMENTA

A ferramenta ESLint com o plugin `eslint-plugin-sonarjs` foi configurada e executada no código original. O resultado da análise está apresentado abaixo:



Análise dos Resultados

O `eslint-plugin-sonarjs` identificou dois problemas críticos:

1. **Complexidade Cognitiva (27 vs 15):** A ferramenta detectou que a complexidade cognitiva do método `generateReport` é 27, quase o dobro do limite recomendado de 15. Esta métrica quantifica a dificuldade de entender o código, considerando estruturas de controle aninhadas e fluxos condicionais.
2. **If Colapsável (linha 43):** A ferramenta identificou que o `if` na linha 43 (`if (user.role === 'USER')`) contém um `if` aninhado (`if (item.value <= 500)`) que pode ser colapsado em uma única condição, reduzindo a complexidade.

Contribuição da Ferramenta

O `eslint-plugin-sonarjs` foi fundamental para identificar problemas que a análise manual poderia ter perdido:

- **Métrica objetiva de complexidade:** A análise manual pode subjetivamente considerar o código "complexo", mas a ferramenta fornece uma métrica quantitativa (27) que permite comparação objetiva
- **Detecção de padrões específicos:** A regra `no-collapsible-if` identifica padrões específicos de refatoração que podem não ser óbvios na análise manual

- **Validação contínua:** A ferramenta pode ser integrada em pipelines de CI/CD para prevenir a introdução de novos smells

A ferramenta complementou a análise manual ao fornecer métricas objetivas e identificar padrões específicos de refatoração, tornando o processo de detecção de smells mais sistemático e confiável.

3. PROCESSO DE REFATORAÇÃO

O smell mais crítico identificado foi a **Complexidade Cognitiva Alta (27)**, que engloba os outros problemas identificados (Método Longo e Duplicação de Código). Para resolver este problema, foram aplicadas as seguintes técnicas de refatoração:

3.1 Técnicas Aplicadas

1. **Extract Method:** O método longo foi quebrado em métodos menores e mais focados:
    - `generateHeader()`: Responsável apenas pela geração do cabeçalho
    - `processItems()`: Processa todos os itens e retorna conteúdo e total
    - `shouldIncludeItem()`: Determina se um item deve ser incluído no relatório
    - `markPriorityIfNeeded()`: Marca prioridade para itens de admin acima do threshold
    - `formatItemLine()`: Formata uma linha de item para CSV ou HTML
    - `generateFooter()`: Gera o rodapé do relatório
2. **Extract Constant:** Números mágicos foram extraídos para constantes:
    - `MAX_USER_ITEM_VALUE = 500`: Limite de valor para usuários comuns
    - `ADMIN_PRIORITY_THRESHOLD = 1000`: Threshold para marcar prioridade em admins
3. **Decompose Conditional:** Condicionais complexas foram simplificadas através da extração de métodos com nomes descritivos que expressam a intenção do código.
4. **Eliminação de Duplicação:** A lógica de formatação foi unificada no método `formatItemLine()`, eliminando a repetição de código.

3.2 Antes e Depois

Código Original (Antes)

```
export class ReportGenerator {
  constructor(database) {
    this.db = database;
  }

  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 += '\n';
      report += '

                    Relatório


\n';
      report += `

Usuário: ${user.name}

\n`;
      report += '\n';
      report += '\n';
    }

    // --- Seção do Corpo (Alta Complexidade) ---
    for (const item of items) {
      if (user.role === 'ADMIN') {
        // Admins veem todos os itens
        if (item.value > 1000) {
          // Lógica bônus para admins
          item.priority = true;
        }
```

```
        if (reportType === 'CSV') {
          report += `${item.id},${item.name},${item.value},${user.nam
          total += item.value;
        } else if (reportType === 'HTML') {
          const style = item.priority ? ' style="font-weight:bold;"'
          report += `\n`;
          total += item.value;
        }
      } else if (user.role === 'USER') {
        // Users comuns só veem itens de valor baixo
        if (item.value <= 500) {
          if (reportType === 'CSV') {
            report += `${item.id},${item.name},${item.value},${user.r
            total += item.value;
          } else if (reportType === 'HTML') {
            report += `\n`;
            total += item.value;
          }
        }
      }
    }

    // --- Seção do Rodapé ---
    if (reportType === 'CSV') {
      report += '\nTotal,,\n';
      report += `${total},,\n`;
    } else if (reportType === 'HTML') {
      report += '
    ID         Nome         Valor
${item.id} ${item.name} ${item.value}
${item.id} ${item.name} ${item.value}
\n';
      report += `
Total: ${total}
\n`;
      report += '\n';
    }

    return report.trim();
```

```
    }
}
```

Complexidade Cognitiva: 27
Problemas detectados pelo ESLint: 2 erros

Código Refatorado (Depois)

```
static MAX_USER_ITEM_VALUE = 500;
static ADMIN_PRIORITY_THRESHOLD = 1000;

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

  report += this.generateHeader(reportType, user);
  const result = this.processItems(reportType, user, items);
  report += result.content;
  total = result.total;
  report += this.generateFooter(reportType, total);

  return report.trim();
}

generateHeader(reportType, user) {
  if (reportType === 'CSV') {
    return 'ID,NOME,VALOR,USUARIO\n';
  }
  if (reportType === 'HTML') {
    return `\n

                    Relatório


\n

Usuário: ${user.name}

\n\n\n`;
  }
  return '';
}

processItems(reportType, user, items) {
  let content = '';
  let total = 0;
```

```
  for (const item of items) {
    if (this.shouldIncludeItem(user, item)) {
      this.markPriorityIfNeeded(user, item);
      content += this.formatItemLine(reportType, item, user);
      total += item.value;
    }
  }

  return { content, total };
}

shouldIncludeItem(user, item) {
  if (user.role === 'ADMIN') {
    return true;
  }
  if (user.role === 'USER') {
    return item.value <= ReportGenerator.MAX_USER_ITEM_VALUE;
  }
  return false;
}

markPriorityIfNeeded(user, item) {
  if (user.role === 'ADMIN' && item.value > ReportGenerator.ADMIN_PRI
    item.priority = true;
  }
}

formatItemLine(reportType, item, user) {
  if (reportType === 'CSV') {
    return `${item.id},${item.name},${item.value},${user.name}\n`;
  }
  if (reportType === 'HTML') {
    const style = item.priority ? ' style="font-weight:bold;"' : '';
    return `\n`;
  }
  return '';
}

generateFooter(reportType, total) {
  if (reportType === 'CSV') {
    return `\nTotal,,\n${total},,\n`;
  }
```

```
  if (reportType === 'HTML') {
    return `
    ID         Nome         Valor
${item.id} ${item.name} ${item.value}
\n

Total: ${total}

\n\n`;
  }
  return '';
}
```

Complexidade Cognitiva: < 15 (dentro do limite)
Problemas detectados pelo ESLint: 0 erros

3.3 Melhorias Alcançadas

- **Redução da Complexidade:** De 27 para abaixo de 15, uma redução de mais de 44%
- **Eliminação de Duplicação:** Lógica de formatação unificada em um único método
- **Melhor Legibilidade:** Métodos com nomes descritivos que expressam claramente sua intenção
- **Facilidade de Teste:** Cada método pode ser testado isoladamente
- **Manutenibilidade:** Mudanças futuras são mais simples e localizadas
- **Extensibilidade:** Adicionar novos tipos de relatório ou roles é mais simples

4. CONCLUSÃO

Este trabalho demonstrou a importância fundamental de utilizar testes como rede de segurança durante o processo de refatoração. A suíte de testes existente garantiu que todas as funcionalidades foram preservadas durante as mudanças estruturais do código. Todos os 5 testes originais continuaram passando após a refatoração, validando que o comportamento externo do sistema permaneceu inalterado.

A redução de Bad Smells no código trouxe benefícios significativos para a qualidade do software:

1. **Manutenibilidade:** O código refatorado é mais fácil de entender e modificar. Novos desenvolvedores podem compreender a estrutura mais rapidamente, e mudanças futuras são menos propensas a introduzir bugs.
2. **Testabilidade:** Métodos menores e mais focados facilitam a criação de testes unitários específicos. Cada método pode ser testado isoladamente, reduzindo a complexidade dos casos de teste.
3. **Extensibilidade:** A estrutura refatorada facilita a adição de novos recursos. Por exemplo, adicionar um novo tipo de relatório (como JSON) requer apenas a criação de novos métodos de formatação, sem modificar a lógica existente.
4. **Redução de Riscos:** Código com menor complexidade cognitiva é menos propenso a bugs. A eliminação de duplicação reduz o risco de inconsistências e facilita a correção de problemas quando eles ocorrem.
5. **Produtividade:** Desenvolvedores gastam menos tempo tentando entender o código e podem fazer mudanças com mais confiança, aumentando a produtividade da equipe.

A combinação de análise manual e ferramentas automatizadas (ESLint com SonarJS) provou ser eficaz na detecção sistemática de problemas. As métricas objetivas fornecidas pela ferramenta complementaram a análise qualitativa manual, resultando em uma identificação mais completa dos Bad Smells.

Em conclusão, a refatoração segura apoiada por testes robustos é uma prática essencial para manter a qualidade do código ao longo do tempo. A redução de Bad Smells não é apenas uma questão estética, mas uma necessidade prática para garantir que o software permaneça manutenível, testável e extensível à medida que evolui.

Repositório GitHub: https://github.com/matheushoske/bad-smells-js-refactoring

Data de Conclusão: 2024