

A3 Gestão e Qualidade de Software

Projeto Anagrama Refatorado

Alunos:

- Arthur Salatine de Moraes, RA 822151203
- Guilherme Lima Machado, RA 822162693

Introdução

O presente relatório detalha a refatoração completa do projeto “Anagrama”, desenvolvido em Java com Swing. O código-fonte original, desenvolvido sob pressões de prazo e com foco exclusivo na funcionalidade, apresentava diversas deficiências que comprometiam sua manutenibilidade, legibilidade e extensibilidade.

A refatoração, baseada nos princípios do *Clean Code*, teve como objetivo transformar o código monolítico e altamente acoplado em uma solução robusta, modular e aderente às boas práticas de desenvolvimento de software.

1. Diagnóstico

A versão inicial do código-fonte serve como base para a identificação das deficiências que justificaram a necessidade de uma refatoração em larga escala.

Código original, antes da refatoração:

<https://github.com/salatinea/Anagrama/tree/7192b8e375f320dab01554b5f7570ab65b2be73b>

As principais deficiências identificadas foram:

i. Monolitismo e alto acoplamento

O código original estava concentrado em um único arquivo (AnagramaApp.java), que acumulava mais de 2600 linhas. Esta classe era responsável por inicializar a interface gráfica, gerenciar o estado do jogo (contadores, palavras, botões clicados) e executar toda a lógica de negócio. Essa concentração violava o Princípio da Responsabilidade Única (SRP - *Single Responsibility Principle*), tornando a manutenção e a localização de *bugs* extremamente complexas.

ii. Baixa legibilidade

O uso de variáveis de estado globais e ambíguas para controle de cada letra (botaoAClicado, contA, contB, etc.) resultava em um código com alto grau de complexidade e difícil de rastrear. Além disso, a lógica de tratamento de eventos dos botões utilizava grandes estruturas condicionais (switch-case), repetindo lógica para cada letra, o que violava o princípio DRY (*Don't Repeat Yourself*).

2. Aplicações

A refatoração foi guiada pelos princípios de *Clean Code*, resultando em um código modular, mais legível e com menor complexidade.

Código refatorado: <https://github.com/salatinea/Anagrama/tree/master>

i. Princípio da Responsabilidade Única (SRP)

O código foi reestruturado em módulos, garantindo que cada classe tenha uma única razão para mudar. Classes novas foram criadas com responsabilidades claras:

- **GameController:** Gerencia a orquestração do jogo, sendo o ponto central de controle da lógica de negócio.
- **EstadoJogo:** Isola o estado do jogo (quais botões foram clicados, a palavra atual em construção, etc.), desacoplando o estado da interface.
- **TratamentoPalavra:** Lida especificamente com a manipulação da palavra a ser adivinhada e sua validação.
- **ButtonFactory e BotoesLetrasEmbaralhadasPanel:** Modularizam a criação e a manipulação dos elementos visuais, eliminando a repetição de código no tratamento de botões.

ii. Eliminação da Repetição de Código (DRY)

A lógica repetitiva para cada letra do alfabeto foi substituída por uma lógica genérica e iterativa. A nova estrutura de painéis e *controllers* permite adicionar ou modificar o comportamento das letras sem a necessidade de reescrever grandes blocos de código condicional, aderindo ao princípio *Open/Closed Principle* (OCP).

iii. Melhoria da Legibilidade e Estrutura

Com a divisão das responsabilidades e o uso de nomes de classes, métodos e variáveis claros e significativos (ex: delete(), desfazer(), checkCorrect()), a legibilidade do código foi drasticamente melhorada.

3. Testes unitários

Para garantir que a refatoração não introduzisse *bugs* e que a lógica de negócio continuasse funcionando corretamente, foram implementados testes unitários focados nas classes de controle e lógica.

Código dos testes: <https://github.com/salatinea/Anagrama/tree/master/testes>

Os testes foram alocados na pasta `/testes` do repositório, conforme o critério de avaliação. Os principais focos dos testes foram:

- **testDelete() e testDesfazer():** Garantem que a manipulação da palavra digitada e o *undo* do último movimento (desfazer) funcionem de maneira determinística, controlando corretamente o estado dos botões.
- **testPalavraCompleta() e testPalavraCorreta():** Verificam se o jogo reconhece o estado final da palavra e se a checagem de correção é executada de forma precisa contra a palavra original.
- **Teste de estado de botões:** Asseguram que, após a interação, o estado dos botões (clicado/não-clicado) seja atualizado corretamente pelo `EstadoJogo` e refletido na interface.

4. Conclusão

A refatoração deste projeto serviu como uma demonstração prática da importância crítica do *Clean Code* na Engenharia de Software. O projeto original, embora funcional, era essencialmente inviável para qualquer extensão ou manutenção a longo prazo, sendo um exemplo de *código legado* devido à sua complexidade.

A aplicação de princípios como o SRP e o DRY transformou o projeto em um sistema manutenível. O esforço inicial da refatoração compensa exponencialmente o tempo que seria gasto na depuração e na introdução de novas funcionalidades no código original. Conclui-se que o *Clean Code* é um investimento essencial que reduz o custo de propriedade do software ao longo do tempo, garantindo a sustentabilidade e a qualidade de qualquer aplicação.