UNIVERSIDADE DE BRASÍLIA

RELATÓRIO DO TRABALHO FINAL

Disciplina: Técnicas de Programação em Plataformas Emergentes

Integrantes: Gustavo Martins Ribeiro, Pablo Christianno Silva Guedes, Philipe de Sousa Barros e Deivid Alves de Carvalho

PERGUNTAS

1. Para cada um dos princípios de bom projeto de código mencionados acima, apresente sua definição e relacione-o com os maus-cheiros de código apresentados por Fowler em sua obra.

De acordo com os princípios evidenciados no enunciado, um bom projeto de código deve primar pela **simplicidade**, evitando a complexidade acidental e eliminando código morto, pela **elegância**, prevenindo a dispersão de responsabilidades (Shotgun Surgery) e o excesso de comentários e pela modularidade, dividindo o sistema em componentes menores para evitar classes e métodos grandes. Logo em seguida, boas **interfaces** são essenciais para garantir a clareza e prevenir nomes pouco descritivos e a invocação de métodos irrelevantes. Já a **extensibilidade** do código deve ser assegurada, evitando a divergência de funcionalidades e a rigidez, enquanto a **duplicação** deve ser minimizada, eliminando código duplicado e agrupamentos de dados que possam redundar em erros. Além disso, a **portabilidade** é crucial, garantindo que o código não dependa excessivamente de plataformas específicas ou contenha condicionalidades excessivas. Por fim, um código **idiomático e bem documentado** deve seguir as convenções da linguagem, evitando confusões e problemas de manutenção decorrentes de nomes pouco descritivos e estruturas confusas. Esses princípios, quando observados, ajudam a identificar e mitigar os maus-cheiros de código, resultando em um sistema mais **fácil de manter** (melhor manutenibilidade) e **escalável** (melhor escalabilidade).

2. Identifique quais são os maus-cheiros que persistem no trabalho prático 2 do grupo, indicando quais os princípios de bom projeto ainda estão sendo violados e indique quais as operações de refatoração são aplicáveis. Atenção: não é necessário aplicar as operações de refatoração, apenas indicar os princípios violados e operações possíveis de serem aplicadas.

Em relação ao trabalho prático 2, observa-se que o projeto foi estruturado em projeto em 3 pacotes, sendo um para comportar tipos de dados, um para comportar os testes e, por fim, um para comportar as classes principais do sistema. Sobre este último, observa-se que há 6 classes distintas, sendo as classes **CALCULADORADETOTAL** e **ENDERECO** resultantes, respectivamente, das operações de refatoração substituir método por objeto-método e extrair classe. Por outro lado, as classes **CLIENTE** e **PRODUTO** não serão alvos de busca por maus-cheiros. Isso se deve à simplicidade atual do sistema, pois tais classes possuem apenas **getters** e **setters**, mas em um eventual futuro elas poderão incorporar novos comportamentos provenientes de novas features do projeto. Por consequência, somente as classes **VENDA** e **CARRINHO** se mostram robustas o suficiente para a análise em questão.

Sobre o código da classe **VENDA**, persistem maus-cheiros como classes grandes, métodos longos, código duplicado, divergência de funcionalidade e código borrado, indicando violações dos princípios de simplicidade, modularidade, evitar duplicação, extensibilidade e boas interfaces. Sendo assim, essa classe concentra muitas responsabilidades, tornando-a difícil de entender e manter, enquanto métodos longos e duplicação de lógica complicam a legibilidade e a manutenção. Além disso, mudanças em uma funcionalidade específica dessa classe podem exigir alterações em várias partes do código, o que dificulta a extensibilidade e aumenta o risco de introdução de erros. Por fim, para resolver esses problemas, algumas operações de refatoração aplicáveis incluem Extrair Método, Extrair Classe e Substituir Método por Objeto-Método, os quais melhorariam a estrutura do código, facilitando sua manutenção e evolução.

Por sua vez, o código da classe **CARRINHO** apresenta maus-cheiros como a presença de métodos longos e a utilização de coleções paralelas, que violam os princípios de simplicidade, modularidade e evitar duplicação. O método calcularTotal, por exemplo, percorre listas de produtos e quantidades de forma paralela, o que não apenas complica a lógica, mas também pode introduzir erros quando há alterações nas coleções. Além disso, a manipulação direta de listas em métodos como adicionarProduto e removerProduto pode ser melhorada através da criação de classes ou métodos específicos para encapsular essa lógica, facilitando a manutenção e a evolução do código. As operações de refatoração recomendadas incluem Extrair Método, para melhorar a clareza e modularidade dos métodos, e Extrair Classe, para encapsular a lógica relacionada a produtos e quantidades, eliminando a dependência de coleções paralelas e simplificando o código.

REFERÊNCIAS BIBLIOGRÁFICAS

- Martin Fowler. Refactoring: Improving the design of Existing Code. Addison-Wesley Professional, 1999.
- Pete Goodliffe. Code Craft: The practice of Writing Excellent Code. No Starch Press, 2006.