Relatório - TPPE - TP3

Data: 13/08/2024

Grupo 1
Eduardo Maia Rezende……180119231
José Luís Ramos Teixeira……..190057858
Pedro Vitor Augusto de Jesus……200073249
Felipe Alef Pereira Rodrigues……190042532

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.

1. Simplicidade

    Definição: A simplicidade em um projeto de software se refere à criação de soluções diretas e claras, evitando complexidade desnecessária. Um código simples é fácil de entender, manter e evoluir. Ele resolve o problema de maneira eficiente, sem introduzir abstrações ou lógica que não sejam necessárias.

    Relação com maus-cheiros de código:

    - "Complexidade Acidental": Quando o código é mais complicado do que o necessário para resolver o problema, muitas vezes devido a excesso de abstrações ou padrões desnecessários.

    - "Código Duplicado": Introduz complexidade, pois a mesma lógica é espalhada em vários lugares, dificultando a manutenção.

    - "Método Longo": Métodos que realizam muitas tarefas podem ser difíceis de entender e manter. Refatorar para métodos menores ajuda a simplificar o código.

2. Elegância

    Definição: A elegância em um projeto de software é a capacidade de criar soluções de forma inteligente, utilizando o mínimo de código necessário para alcançar o máximo efeito. Código elegante é conciso e utiliza boas práticas de design, tornando-se agradável de ler e fácil de seguir.

    Relação com maus-cheiros de código:

    - "Nomenclatura Pouco Expressiva": Nomes inadequados para variáveis, funções, ou classes tornam o código menos elegante. Refatorar para nomes que expressam claramente o propósito contribui para a elegância.

- "Código Morto": Código que não é mais necessário ou utilizado. Remover código morto torna o código mais elegante.

- "Inveja de Função": Quando um método utiliza mais dados de outra classe do que da própria, refatorar para mover o método para a classe adequada pode resultar em um design mais elegante.

3. Modularidade

Definição: Modularidade refere-se à divisão do código em partes menores e independentes (módulos) que podem ser desenvolvidas, testadas e mantidas de forma isolada. Um sistema modular facilita a compreensão, modificação e evolução do software, além de permitir a reutilização de componentes.

Relação com maus-cheiros de código:

- "Classe Grande": Classes que realizam muitas tarefas indicam baixa modularidade. Refatorar para dividir a classe em módulos menores melhora a modularidade.

- "Método Longo": Métodos longos e complexos indicam que a modularidade não está sendo respeitada. A decomposição em métodos menores e mais coesos aumenta a modularidade.

- "Divergência de Mudanças": Quando uma classe precisa ser alterada por diferentes motivos, ela pode estar violando o princípio de responsabilidade única, o que é contrário à modularidade.

4. Boas Interfaces

Definição: Boas interfaces em um projeto de software são aquelas que são claras, concisas e oferecem uma abstração adequada das funcionalidades do módulo ou componente. Elas devem ser intuitivas e permitir que outros desenvolvedores utilizem o módulo sem precisar entender sua implementação interna.

Relação com maus-cheiros de código:

- "Interface Inflada": Uma interface que oferece mais métodos do que o necessário torna-se difícil de entender e usar. Refatorar para simplificar a interface melhora sua qualidade.

- "Divergência de Mudanças": Se uma interface precisa ser modificada por várias razões diferentes, pode ser um sinal de que ela está mal projetada e precisa ser dividida em interfaces menores e mais coesas.

- "Interesse Exagerado em Dados": Se um método ou classe utiliza excessivamente os dados de outra classe, pode ser necessário ajustar a interface para encapsular melhor esses dados.

5. Extensibilidade

Definição: Extensibilidade é a capacidade de um sistema de software de incorporar novas funcionalidades ou modificar as existentes com o mínimo de impacto no código já existente. Um sistema extensível é preparado para futuras mudanças sem necessitar de grandes refatorações.

Relação com maus-cheiros de código:

- “Código Rígido”: Código que é difícil de alterar ou estender sem introduzir bugs. Refatorar para aplicar padrões de design como Factory ou Strategy pode aumentar a extensibilidade.

- "Classe Inchaço": Classes que acumulam muitas responsabilidades tornam difícil a extensão do sistema. Dividir a classe em várias especializadas melhora a extensibilidade.

- "Métodos Longos": Métodos que fazem muitas coisas são difíceis de estender. Refatorar para métodos menores facilita a adição de novas funcionalidades.

6. Evitar Duplicação

Definição: Evitar duplicação significa garantir que a lógica ou dados não sejam repetidos em diferentes partes do código. A duplicação aumenta o risco de inconsistências e dificulta a manutenção.

Relação com maus-cheiros de código:

- "Código Duplicado": A presença de código duplicado é um dos maus-cheiros mais reconhecidos e diretamente relacionado com esse princípio. Refatorar para eliminar duplicação torna o código mais limpo e fácil de manter.

- "Métodos Longos": Métodos que contêm lógica duplicada ou semelhantes em diferentes partes do sistema indicam a necessidade de refatoração.

- "Classe Grande": Se uma classe contém funcionalidades duplicadas que poderiam ser abstraídas, isso pode indicar a necessidade de extração para outras classes ou métodos.

7. Portabilidade

Definição: Portabilidade refere-se à facilidade com que o software pode ser transferido de um ambiente para outro. Um código portável deve ser independente de plataformas e facilmente adaptável para diferentes sistemas operacionais, hardware ou ambientes de execução.

Relação com maus-cheiros de código:

- "Acoplamento Externo": Código que depende fortemente de uma plataforma específica (como chamadas diretas a APIs de SO) reduz a portabilidade. Refatorar para utilizar abstrações ou camadas de adaptação melhora a portabilidade.

- "Dependência de Plataforma": Quando o código é muito específico a uma plataforma, o que dificulta a execução em outros ambientes. Refatorar para reduzir dependências diretas aumenta a portabilidade.

8. Código Idiomático e Bem Documentado

Definição: Código idiomático é aquele que segue as convenções e melhores práticas da linguagem de programação utilizada. Além disso, a boa documentação é essencial para que outros desenvolvedores possam entender rapidamente o propósito e funcionamento do código.

Relação com maus-cheiros de código:

- "Comentário Redundante": Comentários que explicam o óbvio são indicativos de que o código não é claro por si só. Refatorar para que o código seja autodescritivo pode eliminar a necessidade desses comentários.

- "Nomeclatura Pouco Expressiva": Nomes mal escolhidos tornam o código mais difícil de entender. Refatorar para usar nomes mais expressivos e idiomáticos melhora a legibilidade e compreensão.

- "Código Morto": Código que não é mais usado, mas ainda está presente, pode confundir e levar a erros de entendimento, violando o princípio de manter o código claro e documentado.

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.

Maus-Cheiros Identificados e Princípios Violados

1. Classe Grande

Princípio Violado: Modularidade

Operação de Refatoração: Extrair classes menores e mais coesas.

2. Método Longo

Princípio Violado: Simplicidade, Modularidade, Extensibilidade

Operação de Refatoração: Dividir métodos longos em métodos menores e mais específicos.

3. Código Duplicado

Princípio Violado: Evitar Duplicação, Simplicidade

Operação de Refatoração: Extrair métodos ou classes para eliminar duplicação.

4. Nomenclatura Pouco Expressiva

Princípio Violado: Elegância, Código Idiomático e Bem Documentado

Operação de Refatoração: Renomear variáveis, métodos e classes para nomes mais descritivos.