**Curso : Engenharia de Software**

**Disciplina : Técnicas de Programação em Plataformas Emergentes**

**Professor : André Luiz Peron Martins Lanna**

**Alunos :**

* Andŕe Aben-Athar de Freitas
* Lucas Ganda Carvalho
* Lucas Medeiros
* Wictor Bastos Girardi

**Trabalho 3 - Projeto de Código**

1. **Característica 1 - Avoid Duplication ou Ausência de Duplicidade**

* **Uma descrição da característica, mostrando claramente quais são os seus efeitos no código (em termo de estrutura, claridade, coesão, acoplamento dentre outros efeitos aplicáveis)**

De acordo com o Code Craft(GOODLIFFE, 2006), a duplicação é a inimiga do design simples e elegante, além de que redundâncias desnecessárias podem gerar um simples problema, achar um bug e corrigí-lo em um trecho do código, mas esquecer de corrigir outro, comprometendo a segurança do mesmo. Dessa forma, unificar trechos de código diferentes que fazem coisas similares, facilita na hora de corrigir problemas, além de tornar o código mais claro.

* **Uma relação da característica com os maus-cheiros de código definidos por Fowler.**

Essa característica está relacionada intrinsecamente com o mau-cheiro identificado por Fowler como Código Duplicado, pois esta define “Se o mesmo trecho de código aparecer em vários pontos do projeto, saiba que sua solução será melhor se você conseguir unificá-los.”, além de ter ligação também com mau-cheiros como Classe grande, que também é um indício de código duplicado.

* **Pelo menos uma operação de refatoração capaz de levar o projeto de código a ter a característica em análise.**

Uma das operações que ajuda a atingir essa característica é a de *Extrair Método.*

1. **Característica 2 - Modularity ou Modularidade**

* **Uma descrição da característica, mostrando claramente quais são os seus efeitos no código (em termo de estrutura, claridade, coesão, acoplamento dentre outros efeitos aplicáveis)**

Na solução de um determinado problema de design, naturalmente ele é dividido em em módulos ou componentes, assim cada parte é menos complexa que o problema original, mas juntos formam a solução completa. A qualidade dessa decomposição é medida quanto maior a coesão e menor o acoplamento.

Código ou trechos de código, como métodos, com baixa coesão é um sinal de má decomposição, pois cada módulo deve ter um objetivo definido e não um monte de funcionalidades não relacionadas. Assim como baixo acoplamento é um ponto positivo, pois quanto menor o acoplamento, menos as partes separadas terão dependência entre si.

* **Uma relação da característica com os maus-cheiros de código definidos por Fowler.**

Essa característica se relaciona com diversos maus-cheiros definidos por Fowler, por exemplo métodos longos, pois vários métodos ajudam a aumentar a coesão e diminuir acoplamento, classe grande, pois uma classe com muitas variáveis de insta}ĉnasi é um indício de coesão baixa.

* **Pelo menos uma operação de refatoração capaz de levar o projeto de código a ter a característica em análise.**

Algumas das operações que ajudam a atingir essa característica são *Extrair Método e Extrair Classe.*

1. **Característica 3 - Simplicity ou Simplicidade**

* **Uma descrição da característica, mostrando claramente quais são os seus efeitos no código (em termo de estrutura, claridade, coesão, acoplamento dentre outros efeitos aplicáveis)**

Uma das características mais importantes de um código bem escrito, a simplicidade é algo que pode ser fácil de entender. Basicamente se trata de um código simples, coerente e consistente, escrito de forma enxuta e de fácil legibilidade. Isso exige algum esforço, como o matemático Blaise Pascal apreciou: “Sinto muito pela extensão da minha carta, mas não tive tempo de escrever uma curta”. Calcule cuidadosamente quão pouco código é necessário, e então escreva exatamente isso.

* **Uma relação da característica com os maus-cheiros de código definidos por Fowler.**

Essa característica se relaciona com diversos maus-cheiros definidos por Fowler, por exemplo o de código muito longo(Bloated) que em sua teoria são códigos que cresceram demais, na maioria dos casos porque suas responsabilidades não foram bem delimitadas durante o planejamento da aplicação. Métodos e classes muito longas, bem como longas listas de parâmetros são alguns exemplos de Code Smells nesta categoria.

* **Pelo menos uma operação de refatoração capaz de levar o projeto de código a ter a característica em análise.**

Uma das operações que ajudam a atingir essa característica é a de *Extrair Método* pelo fato de simplificar a maneira que o código é escrito.

1. **Característica 4 - Elegancy ou Elegância**

* **Uma descrição da característica, mostrando claramente quais são os seus efeitos no código (em termo de estrutura, claridade, coesão, acoplamento dentre outros efeitos aplicáveis)**

A elegância incorpora os aspectos estéticos do design e muitas vezes anda de mãos dadas com a simplicidade. Basicamente significa que o código escrito não é confuso ou excessivamente complexo. São códigos bem escritos e que possuem uma estrutura digna de uma certa "Beleza e elegância". Tais códigos podem ser identificados por algumas características de acordo com Goodliffe, Pete. *Code Craft : The Practice of Writing Excellent Code*, No Starch Press, Incorporated, 2006. *ProQuest Ebook Central*,:

* Uma única operação não passa por todos os módulos, convertendo o formato de seu parâmetro entre 16 representações diferentes, antes de finalmente ignorá-lo.
* Cada parte complementa as outras, adicionando algo distinto e valioso.
* O design não está repleto de casos especiais.
* Ele associa coisas semelhantes.
* Não há surpresas desagradáveis ​​à espreita ao virar da esquina.
* Há uma pequena localidade de mudança: uma única e simples mudança em um lugar não leva a modificações do código em muitos outros lugares.

Elegância e simplicidade sustentam a maioria das características restantes nesta lista.

* **Uma relação da característica com os maus-cheiros de código definidos por Fowler.**

Essa característica se relaciona com diversos maus-cheiros definidos por Fowler, por exemplo o mau cheiro de cirurgia com rifle, comumente chamado também inibidores de modificação ou da sua forma mais famosa, código espaguete. O código espaguete é aquele no qual, para alterar um ponto, precisamos também fazer alterações em diversos outros, o que torna a manutenção uma dor de cabeça. Uma vez que fazer pequenas correções no código é uma tarefa corriqueira, ela também deve ser de fácil realização e, portanto, códigos com esse sintoma devem ser reescritos a fim de tornar o crescimento da aplicação possível.

* **Pelo menos uma operação de refatoração capaz de levar o projeto de código a ter a característica em análise.**

Algumas das operações que ajudam a atingir essa característica são *Extrair Método e Parametrizar Método.*

1. **Característica 5 - Good Interfaces ou Boas interfaces**

* **Uma descrição da característica, mostrando claramente quais são os seus efeitos no código (em termo de estrutura, claridade, coesão, acoplamento dentre outros efeitos aplicáveis)**

Uma interface é nada mais que um "um contrato que quando assumido por uma classe deve ser implementado" ou seja a fachada pública atrás da qual esconde uma implementação interna, portanto uma boa interface é fundamental para um código bem escrito ao ponto de boa parte das estruturas utilizadas na ciência da computação serem interfaces, como bibliotecas, classes, funções, protocolos e estruturas de dados.

* **Uma relação da característica com os maus-cheiros de código definidos por Fowler.**

Essa característica se relaciona com diversos maus-cheiros definidos por Fowler, por exemplo classes alternativas com interfaces diferentes. Dificultando assim a manutenção e reutilização do código.

* **Pelo menos uma operação de refatoração capaz de levar o projeto de código a ter a característica em análise.**

Uma das operações que ajudam a atingir essa característica é o de *Renomear Método.*