Universidade de Brasília Faculdade UnB Gama

Universidade de Brasília

Técnicas de Programação em Plataformas Emergentes - 2022/1

Trabalho Prático 3

Grupo 1

Aline Lermen - 18/0011961
João Vitor Farias -18/0020251
Lucas Melo - 18/0125885
Nilvan Peres - 17/0122468
Nilo Mendonça - 16/0037522

SUMÁRIO

1. AUSÊNCIA DE DUPLICIDADES

1.1 Descrição da característica

Em um bom projeto de código não deve existir código duplicado. Duplicação de código pode apenas complicar o projeto e reduzir sua elegância e simplicidade. Um exemplo de problema que pode ser causado por isso é o caso em que dois pedaços de código muito parecidos, em que apenas um detalhe os diferencia, possui um "bug" que deve ser resolvido. Após ser resolvido em um dos pedaços, o outro pode ficar esquecido e continuar com o "bug". Não apenas isso, mas só o fato de ser preciso resolver esse "bug" duas vezes (por estar nos dois pedaços de código) já seria um trabalho extra desnecessário. Com a duplicidade, a estrutura do código pode acabar ficando desnecessariamente mais longa e até mais complexa de se entender, enquanto que seria possível trazer mais clareza unindo pedaços de código semelhantes em apenas uma função com um nome que deixe claro qual a sua responsabilidade. A característica de duplicidade de código não se aplica somente a pedaços aleatórios de código e funções, mas também pode ser útil em classes que possuem alguma funcionalidade semelhante e que poderia ser passada para uma superclasse.

1.2 Relação com os maus-cheiros de código definidos por Fowler

Essa característica pode estar relacionada com alguns mau-cheiros de código: código duplicado, classe grande, instruções switch, hierarquias de herança paralelas. Em relação ao primeiro mau-cheiro, "código duplicado", é intuitivo. O mau-cheiro de "instruções switch" é a repetição da estrutura switch-case em diferentes partes do projeto, o que resulta em duplicação de código. Os outros dois mau-cheiros citados, "classe grande" e "hierarquias de herança paralelas" são os que indicam que pode haver código duplicado, pois uma classe com muito código possui altas chances de possuir duplicação. Ou então, quando em uma herança, ao criar uma subclasse é necessário criar também outra subclasse, há um início de duplicação de código.

1.3 Possível operação de refatoração

Para eliminar os possíveis maus-cheiros e alcançar a característica "ausência de duplicidades" de um bom projeto, pode-se utilizar a operação de refatoração "extrair método" que pode resolver alguns dos maus-cheiros citados anteriormente.

2. MODULARIDADE

2.1 Descrição da característica

A modularidade na Engenharia de Software consiste em uma técnica que visa separar os trechos de código em módulos, que por sua vez funcionarão como blocos de instruções que individualmente serão responsáveis por implementar determinada funcionalidade. Essa técnica gera inúmeros benefícios, entre eles facilita na manutenção e evolução do código uma vez que cada módulo pode ser editado sem interferir nos demais, permite o desenvolvimento de diferentes partes do programa de forma paralela entre um ou mais desenvolvedores, diminuindo o tempo necessário para o desenvolvimento, além de facilitar no entendimento, claridade, desenvolvimento de testes, reutilização e diminuir o acoplamento.

2.2 Relação com os maus-cheiros de código definidos por Fowler

Entre os possíveis maus-cheiros que podem estar relacionados a essa característica tem a classe inchada, uma vez que um código pouco modularizado tende a agrupar um grande número de variáveis e funções dentro de uma mesma classe, além de conter funções extensas e de difícil entendimento.

2.3 Possível operação de refatoração

Entre as operações de refatoração que podem auxiliar na obtenção de uma maior modularidade se encontra a operação de extrair classe, uma vez que ajuda no agrupamento de variáveis que possuem sentido em ficarem juntas, além da operação de extrair subclasse.

3. EXTENSIBILIDADE

3.1 Descrição da característica

Um código bem projetado permite que funcionalidades extras sejam inseridas em locais apropriados, quando for necessário. Ao tentar lidar com qualquer futura modificação em potencial, há o risco de gerar um excesso de complexidade no código. A extensibilidade pode ser estruturada por meio de software scaffolding: plugins carregados dinamicamente, hierarquias de classe cuidadosamente escolhidas com interfaces abstratas no topo, o

fornecimento de funções de retorno de chamada úteis e até mesmo uma estrutura de código lógica e maleável.

É preciso que o design do código seja pensado em como o software será estendido. Polvilhar código aleatoriamente com ganchos para extensibilidade pode realmente degradar a qualidade. Deve-se equilibrar a funcionalidade necessária neste momento, o que será adicionado mais tarde e o que será necessário para determinar o quão extensível o design deve ser.

3.2 Relação com os maus-cheiros de código definidos por Fowler

Essa característica pode estar relacionada ao mau-cheiro de generalidade especulativa, ocasionado quando o projeto é modelado de forma genérica, tentando contemplar funcionalidades que talvez fossem implementadas um dia, gerando dificuldade no entendimento do projeto e na manutenção.

3.3 Possível operação de refatoração

No caso do mau-cheiro de generalidade especulativa, é possível utilizar algumas destas operações de refatoração: diminuir hierarquia, para quando classes abstratas possuírem comportamento reduzido; incorporar classes, quando atividades são delegadas para outras classes de forma desnecessária; e remover parâmetro, para quando há parâmetros que não estão sendo utilizados no método.

4. SIMPLICIDADE

4.1 Descrição da característica

Essa característica pode ser considerada a mais importante de um código que foi bem projetado. Pelo fato de ser simples, facilita a compreensão do código,além de ser coerente e consistente, não possui trechos desnecessários, e novas implementações são fáceis de serem adicionadas, e provavelmente não introduzirá um bug na aplicação. Um código simples é o menor possível, mas não o mais pequeno. Um design simples não é fácil de ser criado, leva tempo, e análise de como será o fluxo de informações, um código com o design simples parece óbvio, é só possui essa aparência graças a várias refatorações e análises para o tornar o algoritmo mais direto possível.

4.2 Relação com os maus-cheiros de código definidos por Fowler

Essa característica pode ser relacionada com vários maus-cheiros, porém serão listados os seguintes: herança negada, classe grande e método longo. Em relação ao primeiro, quando uma classe está herdando atributos ou métodos que não serão utilizados, acaba tornando o código mais complexo que deveria, pois, acabam não sendo utilizados pela classe filha. Já em relação ao segundo mau-cheiro, quando uma classe está grande demais além de indicar que a mesma está tendo mais de uma responsabilidade, também viola o princípio de simplicidade, porque acaba trazendo mais dificuldade na legibilidade do código, e novas implementações podem ser mais complicadas por conta do acoplamento da classe. Já em relação ao mau-cheiro de método longo, fere diretamente a objetividade do conceito de simplicidade para métodos, o qual define que: um método deve realizar apenas um único processo para facilitar a compreensão do mesmo.

4.3 Possível operação de refatoração

Para corrigir o mau-cheiro de método longo, podemos utilizar a seguinte refatoração: extração de método, e substituir método por método-objeto. Para classe grande, podemos utilizar as operações de extrair classe, na qual são agrupados em conjuntos de variáveis que juntas fazem sentido para o projeto, extrair como subclasse, quando esse agrupamento de variáveis fazem sentido como uma uma subclasse da classe em que estão. Para o mau-cheiro de herança negada, é possível utilizar a refatoração de descer método/campo, nesta técnica os métodos e campos inutilizados são movidos para uma classe irmã daquela que está rejeitando a herança.

5. BEM DOCUMENTADO

5.1 Descrição da característica

Apesar de um projeto com um bom design já facilitar a entrada de novos contribuidores na aplicação, a documentação é essencial para que o programador consiga entender a estrutura do projeto sem precisar olhar cada arquivo para ter uma ideia geral sobre o funcionamento da aplicação. Caso o design do código esteja em um nível satisfatório, a documentação será pequena, pois a estrutura do código é muito simples.

5.2 Relação com os maus-cheiros de código definidos por Fowler

Essa característica pode ser associada ao mau-cheiro de métodos, quando um método está longo e fazendo mais processos do que deveria, o programador sente a necessidade de adicionar comentários, com o intuito de documentar porque aquele trecho faz parte da função. Entretanto, um método já é uma documentação do código, e quando a mesma encontra-se em um design ideal, dificilmente será necessário a adição de blocos de comentários para a documentação.

5.3 Possível operação de refatoração

A documentação das especificações são essenciais no design arquitetural, os métodos por si só já são uma documentação de código. Também podem ser utilizados programas para a documentação dos endpoints de uma API.

REFERÊNCIAS BIBLIOGRÁFICAS

- Code Craft : The Practice of Writing Excellent Code, No Starch Press, Incorporated, 2006. ProQuest Ebook Central
- Oportunidades de refatoração. André Luiz Peron Martins Lanna. Disponível em <https://docs.google.com/presentation/d/1BG1DVjtOZeG-j3Fmj1cY1gz-4AW9FphX/edit#slide=id.p1>