Oportunidades de refatoração

Prof. André Luiz Peron Martins Lanna

Agenda

- Introdução
- Maus cheiros de código
  - Código duplicado
  - Método longo
  - Classe inchada
  - Lista de parâmetros longa demais
  - Mudanças divergentes
  - Cirurgia com rifle
  - Inveja de recursos
  - Aglomerados de dados
  - Obsessão primitiva
  - Instruções switcj
  - Hierarquias de herança
  - paralelas
  - Classe preguiçosa
  - Generalização especulativa
  - Campo temporário
  - Cadeias de mensagens
  - Homem do meio
  - Intimidade inapropriada
  - Classes alternativas com diferentes interfaces
  - Biblioteca de classes incompletas
  - Classes de dados
  - Herança recusada
  - Comentários
- Conclusões

Introdução

- Saber refatorar (executar cada operação de refatoração) não te faz um bom projetista refatorador. É preciso saber quando (início e término).
- “Maus cheiros” de código são pontos em que há possibilidades para aplicação de refatoração.
  - Como será visto adiante, maus cheiros são pontos em que princípios de bom projeto não são considerados ou podem ser melhorados.

Introdução (cont.)

- Entretanto estes pontos de melhorias não são precisos: o julgamento do refatorador é quem vai decidir quais operações podem e devem ser aplicadas.
- Este julgamento vem na medida em que o projetista ganha familiaridade com as operações de refatoração.

MAUS CHEIROS DE CÓDIGO

Código duplicado

- Se o mesmo trecho de código aparecer em vários pontos do projeto, saiba que sua solução será melhor se você conseguir unificá-los.
- Operações aplicáveis (sugestões):
  - Extrair método: quando a mesma expressão encontra-se em dois métodos na mesma classe.
  - Extrair método / puxar para cima: quando a mesma expressão está em classes irmãs.
  - Extrair método / método template: extrair comportamento comum de dois métodos e implementar a variabilidade em subclasses.
    - Substituir algoritmo: em casos de substituir algoritmo por um mais claro.

Método longo

- Quanto maior for o método, mais difícil é de entendê-lo. Vários métodos curtos (e a delegação entre eles) é preferível.
- Vários métodos não torna o projeto difícil de entender? Não se os métodos forem nomeados devidamente.
- Vários métodos ajuda a aumentar coesão e diminuir acoplamento (por meio de indireção).
- (!) Sempre que sentir necessidade de escrever um comentário sobre um trecho de código, tente extraí-lo como um método.
- (!) Sempre que encontrar um trecho de código comentado, ainda que uma única linha apenas, vale a pena representá-lo por meio de uma operação. Refatore, portanto.
- Expressões condicionais e loops também são candidatas a sofrerem refatorações.

Método longo (cont.)

- Operações aplicáveis (sugestões):
  - Extrair método: sempre que necessário diminuir o tamanho de um método.
  - Extrair método / Substituir temporário por query: sempre que ao usar extrair método, o novo método apresentar muitos parâmetros na assinatura ou variáveis temporárias.
  - Introduzir objeto-parâmetro / preservar objeto inteiro: quando necessário reduzir a lista de parâmetros em uma assinatura de métodos.
  - Substituir método por método-objeto: quando mesmo aplicando as combinações acima a assinatura do método ainda possui muitos parâmetros.
  - Decompor condicional: sempre que lidar com expressões condicionais e loops.

Classe grande

- Quando uma classe apresenta muitas variáveis de instâncias é indício de que ela está com coesão baixa (fazendo mais do que deveria).
- Operações aplicáveis (sugestões):
  - Extrair classe: para agrupar um número de variáveis que juntas fazem algum sentido para o projeto.
  - Extrair subclasse: para agrupar um número de variáves que fazem sentido como uma subclasse da classe em que estão.
  - Ambas operações apresentadas acima são indicadas para agrupar variáveis de instância que são pouco utilizadas pelo código em refatoração.

Classe grande (cont.)

- Classes com muito código indica que há grandes chances de haver código duplicado.
- Operações aplicáveis (sugestão):
  - Extrair classe ou extrair subclasse: para reduzir o tamanho dos métodos da classe.
  - Extrair interface: ao avaliar como os clientes utilizam a classe, extraia interfaces para cada um dos tipos de uso.

Longa lista de parâmetros

- Listas de parâmetros longas são difíceis de entender, pois se tornam inconsistentes e difíceis de usar.
  - Listas longas de parâmetros são muito suscetíveis a mudanças.
  - Antes da programação OO era muito utilizado. Com a OO o número de parâmetros cai drasticamente, pois basta passar os objetos necessários e acessar os elementos desejados.
- Operações aplicáveis:
  - Substituir parâmetro por método: para obter o dado de um parâmetro, realize uma chamada a um objeto que possua o dado.

Longa lista de parâmetros (cont.)

- Operações aplicáveis (cont.):
  - Preservar o objeto inteiro: para substituir um conjunto de dados vindos de um objeto e substituí-los pelo próprio objeto.
  - Introduzir objeto-parâmetro: para substituir um conjunto de dados que não estão logicamente relacionados por um objeto de dados.

Mudanças divergentes

- Idealmente mudanças em uma classe devem ser realizadas de modo pontual.
- Contudo de acordo com o tipo de mudança uma classe pode sofrer diferentes alterações. Ex.:
  - Mudanças em regras de negócio *vs.* persistência.
- Operações aplicáveis:
  - Extrair classe: para acomodar os elementos de cada variação em um único lugar.

Cirurgia com rifle

- É o oposto da mudança com rifle: ao realizar uma mudança várias classes devem sofrer modificações.
- É difícil de encontrar todos os lugares afetados, e fácil de esquecer alguma modificação.
- Operações aplicáveis:
  - Mover método ou mover campo: para colocar todas as variações em uma única classe.
  - Incorporar classe: para agrupar um conjunto de comportamentos em uma única estrutura (aumentar a coesão, caso ela tenha sido afetada com a movimentação de métodos e campos).

- (!) Mudanças divergentes: uma classe que sofre várias mudanças.
  (!) Cirurgia com rifle: uma mudança que afeta várias classes.

Inveja de recursos

- Quando métodos de uma classe estão mais interessados nos recursos de outras classes (geralmente atributos).
  - Utilização excessiva de métodos get de uma classe é indício de inveja de recursos.
- Métodos aplicáveis:
  - Mover método: caso mais simples, traz o método para a classe devida.
  - Extrair método / mover método: para casos em que a inveja ocorre apenas em um trecho do código da outra classe.

- Trazer métodos para dentro de uma classe pode diminuir sua coesão. Avalie a possibilidade de extrair vários métodos e levá-los para as outras classes.

Aglomerados de dados

- É comum que alguns itens de dados apareçam em conjunto em alguns pedaços do código de diversas formas possíveis.
- Por que não transformá-los em um objeto?
- Operações aplicáveis:
  - Extrair classe: para tornar os campos em um objeto.
  - Introduzir objeto parâmetro: para transformar os parâmetros de um método em um objeto, simplificando assim sua assinatura.

Obsessão primitiva

- Tipos de dados primitivos são oferecidos por todas as linguagens: int, float, boolean, etc...
- À partir destes tipos de dados primitivos são formadas estruturas maiores, tais como registros e classes.
- Operações aplicáveis:
  - Trocar dado por objeto: para substituir dados individuais.
  - Trocar tipo código por classe: quando os dados representam um código.
  - Trocar tipo código por subclasses ou Trocar tipo código com State / Strategy: quando os dados são utilizados em comandos condicionais.
  - Extrair classe: quando um grupo de dados deve ser considerado em conjunto.
  - Introduzir parâmetro objeto: quando um conjunto de dados aparece em uma assinatura de um método.

Instruções switch

- Em OO é comum ver a mesma instrução switch ... case em diversos pontos do projeto (duplicação de código).
- Solução mais elegante para este comando é o uso de polimorfismo.
- Operações aplicáveis:
  - Extrair método / Mover método: para extrair todo o comando switch e movê-lo para onde o polimorfismo é necessário.
  - Trocar tipo por subclasse ou trocar tipo por State / Strategy: uma vez que o método encontra-se no local devido altere-o de moro a usar o polimorfismo.

Hierarquias de herança paralelas

- Caso especial de cirurgia com rifle:
  - Sempre que adicionar uma subclasse em um ramo, terá que adicionar a mesma subclasse no outro ramo.
  - É também indício de duplicação de código.
- Elimine a duplicação certificando que instâncias de uma hierarquia referenciam instâncias da outra.
- Operações aplicáveis:
  - Mover método e mover campo: de modo a retirar as dependências entre as classes.

Classe preguiçosa

- Classes que não possuem comportamento suficiente para ser mantida no projeto, ou seja, pequena demais deve ser eliminada do projeto.
- Operações aplicáveis:
  - Mover método e mover atributo: de modo a levar os elementos presentes na classe para outras classes em que são mais indicados.

Generalidade especulativa

- Projetos que são genéricos demais, em que o projetista fez previsões de funcionalidades que algum dia poderiam vir a ser implementadas.
- São projetos difíceis de entender e manter.
- Operações aplicáveis:
  - Diminuir hierarquia: quando há classes abstratas que não fazem muita coisa.
  - Incorporar classe: quando há delegações desnecessárias.
  - Remover parâmetro: para os parâmetros que não são utilizados em um método.
- Um bom indício deste “mau cheiro” é quando o método ou classe é usado apenas por métodos de teste.

Campo temporário

- Ocorre quando algumas variáveis de instância de um objeto são utilizadas apenas em algum contexto específico (e não a maior parte dela).
- Situações típicas: quando ao invés de usar uma longa lista de parâmetros são utilizadas várias variáveis temporárias.
- Operações aplicáveis:
  - Extrair classe: para criar um local onde as variáveis temporárias serão armazenadas.

Cadeias de mensagens

- Ocorre quando um objeto chama outro, que chama outro, que chama outro e assim sucessivamente:
  - Geralmente ocorre por meio de métodos Get!
  - Indício de alto acoplamento: o projeto está ligado à navegação dos métodos.
    - Uma alteração nesta estrutura impacta mudanças no cliente.
- Operações aplicáveis:
  - Ocultar delegação: pode causar indireção indesejada.
  - Extrair método / mover método: extrair o trecho de código que é utilizado pelo cliente e movê-lo para pontos iniciais da cadeia.

Homem do meio

- Caso em que a indireção é indesejável. Os métodos de uma classe simplesmente delegam a execução para métodos de outras classes.
- Operações aplicáveis:
  - Remover homem do meio: para retirar a indireção e chamar o método diretamente.
  - Introduzir método: quando métodos não fazem muita coisa, traga-os para o método que o chama.
  - Trocar delegação com herança: quando há comportamento adicional, transforme a classe de indireção em uma subclasse do objeto real.

Intimidade inapropriada

- Algumas vezes uma classe torna muito íntima das demais, o que permite que ela investigue as partes privadas das outras classes. O ideal é que essa relação seja a mais restrita possível.
- Operações aplicáveis:
  - Mover método e mover campo: para reduzir a intimidade inapropriada.
  - Mudar associação bidirecional para unidirecional: para reduzir o acoplamento entre duas classes.
  - Extrair classe: para extrair o comportamento comum a duas classes em uma terceira classe.
  - Ocultar delegação: para não permitir que o cliente de uma classe conheça o método que executa a delagação de uma tarefa.

Classes alternativas com interfaces diferentes

- Métodos que fazem a mesma coisa mas que apresentam assinaturas diferentes, conforme as classes às quais estão ligadas.
- Operações aplicáveis:
  - Renomear método: para alterar a assinatura dos métodos divergentes.
  - Mover método: para mover comportamento entre as classes.

Biblioteca de classes incompleta

- Muitas vezes é comum encontrar bibliotecas de classes que não oferecem todos os recursos dos quais precisamos. Além disso, não conseguimos ter acesso ao seu fonte para modificá-lo.
- Operações aplicáveis:
  - Introduzir método estrangeiro: para adicionar um comportamento ainda não implementado.
  - Introduzir extensão local: para adicionar várias funcionalidades ainda não implementadas pela biblioteca.

Classe de dados

- Classes que possuem apenas campos de dados, métodos *set* e *get*. São classes simples demais e que certamente são muito utilizadas pelas demais classes da aplicação.
- Operações aplicáveis:
  - Encapsular campo / coleções: para ocultar dos clientes os detalhes de sua implementação.
  - Remover método *set*: para não permitir que nenhum campo seja alterado por alguem externo à classe.
  - Mover / extrair e ocultar método: essas três operações são muito utilizadas para lidar com métodos *set* e *get* muito utilizados por outras classes.

Herança negada

- Subclasses herdam métodos e dados de suas classes-pai, mas nem sempre gostariam de herdá-los. Nesses casos há um indício claro que a hierarquia está errada.
- Operações aplicáveis:
  - Descer método / campo: para mover os métodos e campos inutilizados para uma classe irmã daquela que está rejeitando a herança.
  - Substituir herança por delegação: quando a subclasse não quer herdar implementação de uma interface realizada por sua superclasse.

Referência bibliográfica

- Fowler, Martin. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999. [capítulo 3].