Teste de Software

Refatoração, Maus Cheiros e Catálogo de Refatoração

PUC Minas

Instituto de Ciências Exatas
e Informática

Prof. Lesandro Ponciano

Departamento de Engenharia de Software
e Sistemas de Informação (DES)

Refatoração

- *"a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior"*
  - uma mudança feita na estrutura interna de um software para tornar mais fácil de entender e mais barato de modificar sem mudar o seu comportamento observável

- *"to restructure software by applying a series of refactorings without changing its observable behavior"*
  - reestruturar o software aplicando uma série de refatorações sem mudar seu comportamento observável

Maus Cheiros

- Um mau cheiro é uma indicação de problema
- Detectar um mau cheiro requer
  - intuição
  - percepção
  - experiência
- Um mau cheiro é a indicação da necessidade de uma refatoração
  - Vamos discutir um catálogo com 21 situações

1 – Código Duplicado

- Código duplicado em classes não relacionadas
- Mesma expressão em
  - vários métodos
  - classes irmãs
- Algoritmos diferentes que fazem a mesma coisa

2 – Método Longo

- Quanto mais código em um método, mais difícil de entender o método
- Mistura de diferentes comportamentos

3 – Classes Grandes

- Ocorrência de variáveis em excesso
  - Possível quebra de padrão especialista na informação
  - Mistura de atributos de diferentes camadas (apresentação, domínio e dados)

- Código longo
  - Possível código duplicado
  - Mistura de operações de diferentes camadas (apresentação, domínio e dados)

4 – Lista Longa de Parâmetros

- Métodos que recebem uma grande quantidade de parâmetros

- Talvez deve-se usar passagem de objetos ao invés de valores isolados

5 – Alterações por Motivos Divergentes

- Uma classe muito suscetível a alterações
  - alterações por diferentes distintas

- Talvez deve-se fazer o particionamento em outras classes

6 – Cirurgia com Rifle

- Sentido oposto à alterações por motivos divergentes
  - uma alteração gera mudanças em muitas classes

- Possíveis soluções
  - Mover métodos ou variáveis
  - Criar novas classes agregadoras

7 - Inveja dos Dados

- Método que parece mais interessado em outra classe que na sua
  - Usa mais dados da outra classe do que seus próprios dados
  - Inveja dos dados contidos em outra classe
  - Ocasiona alto acoplamento

- Possível solução
  - Mover método para a outra classe

8 - Agrupamento de Dados

- Dados que aparecem frequentemente juntos, mas não estão logicamente ligados

- Possíveis soluções
  - Criação de classes (Talvez empregando o padrão GRASP de invenção pura)
  - Transformação de parâmetros em objetos

9 – Obsessão por Tipos Primitivos

- Dados primitivos isolados no programa
- Por exemplo, usar *string* para representar pelo menos diferentes dados como nome, CPF e id

10 - *Switches* numerosos ou duplicados

- Uso de *switch* vs. Polimorfismo

- Se o polimorfismo é um exagero
  - substituição de parâmetros por métodos explícitos

11 - Hierarquias paralelas de herança

- A adição de uma subclasse leva à adição de outra subclasse em outra classe
- Caso especial de cirurgia com rifle

12 - Classes ociosas

- Classes pouco usadas
  - Cada classe custa dinheiro para manter e compreender
- Classes que simplesmente não são úteis o bastante

13 - Generalidade especulativa

- Criar estruturas adicionais para algo que nunca será utilizado

- Classes abstratas desnecessárias, delegações em excesso

14 - Atributos temporários

- Só são úteis durante um tempo de vida muito limitado da classe
- Uma variável de instância recebe um valor apenas em determinadas circunstâncias
  - Você espera que o objeto precisa de todas as suas variáveis, mas isso não ocorre
- Possível solução
  - extrair uma classe para variáveis órfãos

15 – Cadeia de Mensagens

- O usuário é obrigado a encadear várias chamadas a métodos para realizar uma tarefa
- Pode-se ocultar a delegação para tornar a tarefa mais clara
- Se for o caso, mover métodos

16 - Intermediários

- Encapsulamento leva à delegação

- Intermediários cumprem um papel importante em programação OO por causa da delegação de métodos, mas excessos devem ser evitados

17 - Intimidade inadequada

- Classes que utilizam as partes ‘privadas’ de outras

- Possíveis soluções
  - Transformar associações bidirecionais em unidirecionais
  - Trocar herança por delegação
  - Nova classe intermediária

18 – Classes Alternativas com Interfaces Diferentes

- Métodos que fazem a mesma coisa mas têm assinaturas diferentes

19 - Biblioteca de classes incompleta

- Dificuldades em inserir novos comportamentos em uma biblioteca

20 – Classes de Dados

- Apenas atributos, sem comportamentos
- São "depósitos burros de dados"
- São iguais crianças
  - bom ponto de partida
  - mas a longo prazo precisam adquirir um pouco de responsabilidade

21 - Herança Recusada

- Quando classes não precisam do que herdaram
- Hierarquia mal planejada

Catálogo de Refatorações

- Esse catálogo de 21 maus cheiros que levam a refatoração não é exaustivo
- Ele é um ponto de partida para o trabalho de refatoração
- Divisão das refatorações por categorias

Exemplo de Categorias

1) Composição de métodos
2) Movendo recursos entre objetos
3) Organizando dados
4) Simplificando expressões condicionais
5) Tornando as chamadas de métodos mais simples
6) Lidando com generalizações
7) Refatorações grandes

Estrutura de um Item do Catálogo

- Nome
- Sumário da situação e da refatoração
- Motivação
- Mecânica da refatoração
- Exemplo(s)

Referências

SOMMERVILLE, Ian. Engenharia de Software - 9a edição. Pearson ISBN 9788579361081.

PRESSMAN, Roger. Engenharia de software. 8. Porto Alegre ISBN 9788580555349.

BECK, Kent. TDD desenvolvimento guiado por testes. Bookman Editora, 2009.

FOWLER, Martin. Refatoração: Aperfeiçoamento e Projeto. Bookman Editora, 2009.

MARTIN, Robert; MARTIN, Micah. Princípios, padrões e práticas ágeis em C# (Seção II, Capítulo 7)

Caram, João. Notas de aula. PUC Minas.