Universidade de São Paulo
Brasil

Aula 10 – Refatoração de Software

Se feder, está na hora de trocar

MAC0321 - Laboratório de Programação Orientada a Objetos

Professor: Marcelo Finger **(mfinger@ime.usp.br)**

Departamento de Ciência da Computação
Instituto de Matemática e Estatística

Tópicos

1. Conceitos
2. Mal cheiro do código
3. Refatorações mais comuns

Contexto

Conceitos

- **Refatoração** é o processo de *mudança do design* uma aplicação sem modificar o seu comportamento original [Opdyke, 1992 apud Mens e Tourwé, 2004].
- Uma [pequena] modificação no sistema que não altera o seu comportamento funcional
- Mas que melhora alguma qualidade não-funcional.

Objetivos e Aplicações

- Prevenir o envelhecimento do design e garantir a flexibilidade adequada para evoluções
- Permitir a integração tranquila de futuras extensões/alterações
- Código legado
- Métodos ágeis incorporam como uma prática – Extreme Programming

Qualidades não-funcionais

- Possíveis aspectos de qualidade não-funcional
  - simplicidade
  - flexibilidade
  - clareza
  - desempenho

Extensão da Refatoração

- Cada refatoração tão simples que leva alguns segundos ou poucos minutos.
- É uma operação sistemática e óbvia (ovo de Colombo).
- Parece não ajudar muito.
- Mas quando se juntam 50 refatorações, bem escolhidas, em sequência, ***o código melhora radicalmente***.
- O segredo está em ter um bom repertório de refatorações e saber aplicá-las criteriosa e sistematicamente

Oportunidades de Refatoração

- Mudança do nome de variáveis
- Mudanças nas interfaces dos objetos
- Pequenas mudanças arquiteturais
- Encapsular código repetido em um novo método
- Generalização de métodos

raizQuadrada(float x) $\Rightarrow$ raiz(float x, int n)

Por que sistematizar?

- Refatorações sempre existiram
- Mas não tinham nome, eram implícitas e *ad hoc*.
- A novidade está em criar um vocabulário e catálogo
- Sistematizar para aprender novas técnicas, e ensinar aos outros.

Mal cheiro de código

Quando usar?

- Sempre há duas possibilidades:
  1. Melhorar o código existente.
  2. Jogar fora e começar do 0.
- É sua responsabilidade avaliar a situação e decidir quando é a hora de optar por um ou por outro.
- Quando NÃO refatorar?
  - Quando é tão ruim que reescrever é melhor
  - Quando você está próximo de um prazo!

Quando Usar? Mau cheiro

- Cunhada por Beck [Fowler, 2000], a expressão *bad smell* refere-se às estruturas no código que sugerem (e às vezes gritam pela) possibilidade de refatoração.
- A partir da identificação de um mau cheiro é possível propor refatorações adequadas, que podem reduzir ou mesmo eliminar o mau-cheiro.

Histórico

- Surgiu na comunidade de Smalltalk nos anos 80/90.
- Desenvolveu-se formalmente na Universidade de Illinois em Urbana-Champaign.
- Grupo do Prof. Ralph Johnson.
  - Tese de PhD de William Opdyke (1992).
  - John Brant e Don Roberts: The Refactoring Browser Tool
- Kent Beck (XP) na indústria.

Estado Atual

- Hoje em dia a refatoração é um dos preceitos básicos de Programação eXtrema (XP).
- Mas não está limitado a XP, pode ser usado em qualquer contexto
- Não é limitado a Smalltalk.
- Pode ser usado em qualquer linguagem, mesmo não orientada a objetos

Passos para aplicação de uma refatoração

- Detectar oportunidade de refatoração
- Identificar refatorações apropriadas
- Realizar (automaticamente) as refatorações.

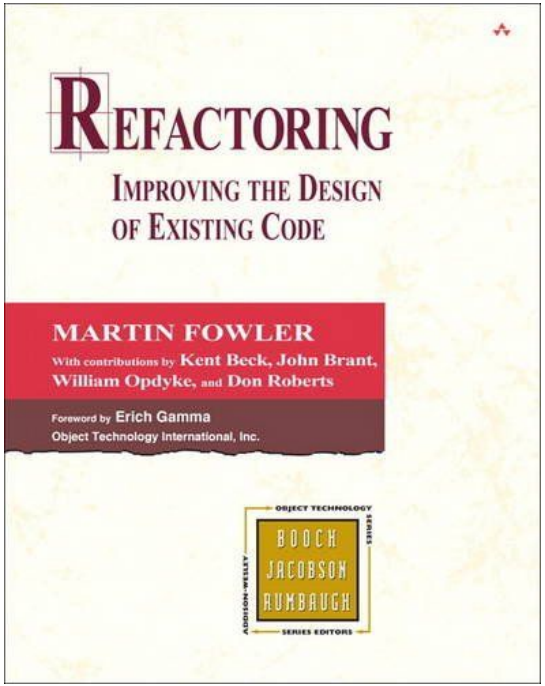
Testes para refatoração

- Antes de refatorar, tenha um conjunto sólido de testes para garantir que o comportamento não seja alterado
- Refatorações podem adicionar erros.
  - Porém, como são feitas em pequenos passos, é fácil recuperar-se de uma falha
- Os testes ajudam a detectar erros se eles forem criados
- Testes automáticos são capazes de se auto-verificarem
- Cuidado! Testes também fedem!

Refatorações mais comuns

Catálogo de Refatorações

- [Fowler, 2000] contém 72 refatorações.
- Vale a pena gastar algumas horas com [Fowler, 2000].



Algumas Refatorações

Cheiro	Refatoração a ser aplicada
Código duplicado	Extract Method (110) Substitute Algorithm (139)
Método muito longo	Extract Method (110) Replace Temp With Query (120) Introduce Parameter Object (295)
Classe muito grande	Extract Class (149) Extract Subclass (330) Extract Interface (341) Duplicate Observed Data (189)
Intimidade inapropriada	Move Method (142) Move Field (146) Replace Inheritance with Delegation(352)

Formato de Entradas no Catálogo

- **Nome** da refatoração.
- **Resumo** da situação na qual ela é necessáriae o que ela faz.
- **Motivação** para usá-la (e quando não usá-la).
- **Mecânica**, i.e., descrição passo a passo.
- **Exemplos** para ilustrar o uso.

Nome: Extract Method

- **Resumo:** Há um fragmento de código que poderia ser agrupado.
- **Mude o fragmento** para um novo método e escolha um nome que explique o que ele faz.
- **Motivação:** é uma das refatorações mais comuns. Se um método é longo demais ou difícil de entender e exige muitos comentários, extraia trechos do método e crie novos métodos para eles. Isso vai melhorar as chances de reutilização do código e vai fazer com que os métodos que o chamam fiquem mais fáceis de entender. O código fica parecendo comentário.

Extract Method

- **Mecânica:** Crie um novo método e escolha um nome que explicite a sua intenção (o nome deve dizer o que ele faz, não como ele faz).
- Copie o código do método original para o novo.
- Procure por variáveis locais e parâmetros utilizados pelo código extraído.
  - Se variáveis locais forem usadas apenas pelo código extraído, passe-as para o novo método
  - Caso contrário, veja se o seu valor é apenas atualizado pelo código. Neste caso substitua o código por uma atribuição
  - Se é tanto lido quando atualizado, passe-a como parâmetro
- Compile e teste.

Exemplos: extração de métodos

```
public class ExtractMethodExample0 {
     List<Double> pedidos;
     String nome;
     void imprimeDivida () {
          ListIterator<Double> it = pedidos.listIterator();
          double divida = 0.0;
          // imprime cabeçalho
          System.out.println ("*****************************");
          System.out.println ("*** Dívidas do Cliente ****");
          System.out.println ("*****************************");
          // calcula dívidas
          while (it.hasNext()){
               double valor = it.next();
               divida += valor;
          }
          // imprime detalhes
          System.out.println ("nome: " + nome);
          System.out.println ("divida total: " + divida);
          }
}
```

Exemplos: extração de métodos

```
public class ExtractMethodExample1 {
	List<Double> pedidos;
	String nome;
	void imprimeDivida () {
		ListIterator<Double> it = pedidos.listIterator();
		double divida = 0.0;
		imprimeCabecalho(); // novo !
		// calcula dívidas
		while (it.hasNext()){
			double valor = it.next();
			divida += valor;
		}
		// imprime detalhes
		System.out.println ("nome: " + nome);
		System.out.println ("divida total: " + divida);
	}
	void imprimeCabecalho() {
		System.out.println ("**************************");
		System.out.println ("*** Dívidas do Cliente ****");
		System.out.println ("**************************");
	}
}
```

Exemplos: extração de métodos

```
public class ExtractMethodExample2 {
	List<Double> pedidos;
	String nome;
	void imprimeDivida () {
		ListIterator<Double> it = pedidos.listIterator();
		double divida = 0.0;
		imprimeCabecalho();
		// calcula dívidas
		while (it.hasNext()){
			double valor = it.next();
			divida += valor;
		}
		imprimeDetalhes(divida); // novo !
	}
	void imprimeDetalhes(double divida) {
		System.out.println ("nome: " + nome);
		System.out.println ("divida total: " + divida);
	}
	void imprimeCabecalho() {
		System.out.println ("**************************");
		System.out.println ("*** Dívidas do Cliente ****");
		System.out.println ("**************************");
	}
}
```

Exemplos: extração de métodos

```
public class ExtractMethodExample3 {
	List<Double> pedidos;
	String nome;
	void imprimeDivida () {
		imprimeCabecalho();
		double divida = calculaDivida();
		imprimeDetalhes( divida); // novo !
	}
	private double calculaDivida() {
		ListIterator<Double> it = pedidos.listIterator();
		double divida = 0.0;
		while (it.hasNext()){
			double valor = it.next();
			divida += valor;
		}
		return divida;
	}
	private void imprimeDetalhes( double divida) {
		System.out.println ("nome: " + nome);
		System.out.println ("divida total: " + divida);
	}
	private void imprimeCabecalho() {
		System.out.println ("**************************" );
		System.out.println ("*** Dívidas do Cliente ****" );
		System.out.println ("**************************" );
	}
}
```

Exemplos: extração de métodos

Dá para ficar mais curto ainda:

```
void imprimeDivida () {
        imprimeCabecalho ();
        imprimeDetalhes (calculaDivida ());
}
```

Mas não é necessariamente melhor pois é um pouco menos claro.

Substituir temporário por chamada

Substitui o uso de um variável por uma chamada a um método que realiza as operações.

Motivação:

– Variáveis temporárias incentivam seu uso prolongado por terem um escopo limitado

Exemplo: delegação

```
public class ReplaceTempExample {
      private double quantidade;
      private double precoItem;
      double getPreco() {
            double precoBase = quantidade * precoItem;
            double fatorDesconto;
            if (precoBase > 1000)
                  fatorDesconto = 0.95;
            else fatorDesconto = 0.98;
            return precoBase * fatorDesconto;
      }

      double getPrecoRefatorado() {
            return precoBase() * fatorDesconto();
      }
      private double precoBase() {
            return quantidade * precoItem;
      }
      private double fatorDesconto() {
            return (precoBase() > 1000) ? 0.95 : 0.98;
      }
}
```

Substituir Herança por Delegação

Uma classe herda de uma outra classe, mas usa muito pouco da superclasse. Operações da superclasse não se aplicam à subclasse: delegação é mais apropriada.

Motivação:

– Variáveis temporárias incentivam seu uso prolongado por terem um escopo limitado

Exemplo: delegação

```
class HerancaPura<T> extends Vector<T> {
      public void push (T element) {
            insertElementAt(element, 0);
      }
      public T pop () {
            T result = firstElement ();
            removeElementAt (0);
            return result;
      }
}
public class HerancaPorDelegacao<T>  {
      Vector<T> vetorLocal = new Vector<T>(0);
      public void push (T element) {
            vetorLocal.insertElementAt(element, 0);
      }
      public T pop () {
            T result = vetorLocal.firstElement ();
            vetorLocal.removeElementAt (0);
            return result;
      }
}
```

Outras Refatorações

- **Colapso de hierarquia:** A superclasse e a subclasse não são muito diferentes. Combine-as em apenas uma classe
- **Subir atributo:** Todas subclasses têm os mesmos atributos. Mova os atributos repetidos para a superclasse
- **Extração de interface:** Subconjunto da interface é muito usado, ou várias classes têm partes de suas interfaces em comum. Classes passam a implementar a nova interface

Futuro das Refatorações

- Evitar refatorações manuais
- Preferir o uso de ferramentas para refatorações
- Em pesquisa: Identificação de mal-cheiro por técnicas de processamento de língua natural (PLN/NLP)
- Em pesquisa: sugestão de refatorações por IA

LLMs e Refatorações

- LLMs (como chatGPT) são treinadas em programas na internet
- A maioria dos programas na internet são de baixa qualidade (fedem!)
- Ao usar LLM para gerar código, este obrigatoriamente deve ser refatorado
- Se o código não funciona, melhor nem perder tempo e reescrevê-lo

Lista de exercícios

No computador com o Eclipse

Entrega até o final do dia

MAC321
Lab POO

- Professor: Marcelo Finger

  E-mail: mfinger@ime.usp.br