Refatoração de Software

Prof. Dr Marcos L. Chaim

EACH -- USP

Material elaborado a partir de slides dos professores Dr. Alfredo Goldman e Dr. Fábio Kon – DCC/IME/USP

Atualização: Eliane Martins
Nov/2010

Conceito

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

Conceito

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

Objetivos & aplicações

- Prevenir o envelhecimento do *design* e garantir a flexibilidade adequada para permitir a integração *tranqüila* de futuras extensões/alterações [Mens eTourwé, 2004]
- Código legado.
- Métodos ágeis incorporam como uma prática – Extreme Programming.

Exemplos 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)` ⇒ `raiz(float x, int n)`

Refatoração

- Em geral, uma *refatoração* é tão simples que parece que não vai ajudar muito.
- Mas quando se juntam 50 refatorações, bem escolhidas, em seqüência, o código melhora radicalmente.

Refatoração

- Cada refatoração é simples.
- Demora alguns segundos ou alguns poucos minutos para ser realizado.
- É uma operação sistemática e óbvia (ovo de Colombo).
- O segredo está em ter um bom vocabulário de *refatorações* e saber aplicá-las criteriosa e sistematicamente.

Refatoração Sempre Existiu

- Mas não tinha um nome.
- Estava implícito, *ad hoc*.
- A novidade está em criar um vocabulário comum e em catalogá-las.
- Assim podemos utilizá-las mais sistematicamente.
- Podemos aprender novas técnicas, ensinar uns aos outros.

Quando Usar Refatoração

- 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.

Origens

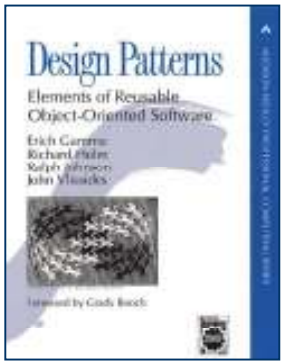
- Surgiu na comunidade de Smalltalk nos anos 80/90.
- Desenvolveu-se formalmente na Universidade de Illinois em Urbana-Champaign.
- Grupo do Prof. Ralph Johnson.
  - Tese de PhD de William Opdyke (1992).
  - John Brant e Don Roberts:
    - ***The Refactoring Browser Tool***
- Kent Beck (XP) na indústria.

Estado Atual

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

Catálogo de Refatorações

- [Fowler, 2000] contém 72 refatorações.
- Vale a pena gastar algumas horas com [Fowler, 2000].
- Análogo aos padrões de projeto orientado a objetos [Gamma et al. 2000] (GoF).



Passos para aplicação de uma refatoração

- Detectar quando uma aplicação precisa ser refatorada.
- Identificar quais *refatorações* devem ser aplicadas e onde.
- Realizar (automaticamente) as refatorações. [Tourwé e Mens, 2003].

Mau cheiro

- Cunhada por Beck [Fowler, 2000], o termo *mau cheiro* (*bad smell*) refere-se às estruturas no código que sugerem (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.

Mau Cheiro

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)*

Aplicando uma refatoração

- Antes de começar a refatoração, verifique se você tem um conjunto sólido de testes para verificar a funcionalidade do código a ser refatorado.
- Refatorações podem adicionar erros.
- Os testes vão ajudá-lo a detectar erros se eles forem criados.

Formato de Cada Entrada no Catálogo

- **Nome** da refatoração.
- **Resumo** da situação na qual ela é necessária e 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.

Extract Method (110)

- **Nome**: *Extract Method*
- **Resumo:** *Você tem 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.*

Extract Method (110)

- **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 (110)

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.

Extract Method (110)

Mecânica (cont.):

- 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.

Extract Method (110)
Exemplo Sem Variáveis Locais

```
void imprimeDivida () {
   Enumerate e = _pedidos.elementos ();
   double divida = 0.0;
   // imprime cabeçalho
   System.out.println ("**************************");
   System.out.println ("*** Dívidas do Cliente ****");
   System.out.println ("**************************");
   // calcula dívidas
   while (e.temMaisElementos ()){
        Order cada = (Order) e.proximoElemento ();
        divida += cada.valor ();
   }
   // imprime detalhes
   System.out.println ("nome: " + _nome);
   System.out.println ("divida total: " + divida);
}
```

Extract Method (110)
Exemplo Sem Variáveis Locais

```
void imprimeDivida () {
    Enumerate e = _pedidos.elementos ();
    double divida = 0.0;
    imprimeCabecalho ();
    // calcula dívidas
    while (e.temMaisElementos ()){
          Order cada = (Order) e.proximoElemento ();
          divida += cada.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 ("***************************");
}
```

Extract Method (110)
Exemplo COM Variáveis Locais

```
void imprimeDivida () {
   Enumerate e = _pedidos.elementos ();
   double divida = 0.0;
   imprimeCabecalho ();
   // calcula dívidas
   while (e.temMaisElementos ()){
        Order cada = (Order) e.proximoElemento ();
        divida += cada.valor ();
   }
   imprimeDetalhes (divida);
}
void imprimeDetalhes (double divida)
{
   System.out.println("nome: " + _nome);
   System.out.println("divida total: " + divida);
}
```

Extract Method (110)
com atribuição

```
void imprimeDivida () {
   imprimeCabecalho ();
   double divida = calculaDivida ();
   imprimeDetalhes (divida);
}

double calculaDivida ()
{
   Enumerate e = _pedidos.elementos ();
   double divida = 0.0;
   while (e.temMaisElementos ()){
        Order cada = (Order) e.proximoElemento ();
        divida += cada.valor ();
    }
   return divida;
}
```

Extract Method (110)
depois de compilar e testar

```
void imprimeDivida () {
   imprimeCabecalho ();
   double divida = calculaDivida ();
   imprimeDetalhes (divida);
}

double calculaDivida ()
{
   Enumerate e = _pedidos.elementos ();
   double resultado = 0.0;
   while (e.temMaisElementos ()){
        Order cada = (Order) e.proximoElemento ();
        resultado += cada.valor ();
   }
   return resultado;
}
```

Extract Method (110)
depois de compilar e testar

- Dá para ficar mais curto ainda:

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

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

Inline Method (117)

- **Nome**: *Inline Method*
- **Resumo:** a implementação de um método é tão clara quanto o nome do método. Substitua a chamada ao método pela sua implementação.
- **Motivação:** bom para eliminar indireção desnecessária. Se você tem um grupo de métodos mal organizados, aplique *Inline Method* em todos eles seguido de uns bons *Extract Method* s.

Inline Method (117)

- **Mecânica:**
  - Verifique se o método não é polimórfico ou se as suas subclasses o especializam.
  - Ache todas as chamadas e substitua pela implementação.
  - Compile e teste.
  - Remova a definição do método.
  - Dica: se for difícil -> não faça.

Inline Method (117)

- **Exemplo:**

```
int bandeiradaDoTaxi (int hora) {
   return (depoisDas22Horas (hora)) ? 2 : 1);
}
int depoisDas22Horas (int hora) {
   return hora > 22;
}
```

```
int bandeiradaDoTaxi (int hora) {
        return (hora > 22) ? 2 : 1);
}
```

Replace Temp with Query (120)

- **Nome**: *Replace Temp with Query*
- **Resumo:** Uma variável local está sendo usada para guardar o resultado de uma expressão. Troque as referências a esta expressão por um método.
- **Motivação:** Variáveis temporárias encorajam métodos longos (devido ao escopo). O código fica mais limpo e o método pode ser usado em outros locais.

Replace Temp with Query (120)

- **Mecânica:**
  - Encontre variáveis locais que são atribuídas uma única vez
    - Se `temp` é atribuída mais do que uma vez use *Split Temporary Variable* (128)
  - Declare `temp` como `final`
  - Compile (para ter certeza)
  - Extraia a expressão
    - Método privado - efeitos colaterais
  - Compile e teste

Replace Temp with Query (120)

```
double getPreco() {
    int precoBase = _quantidade * _precoItem;
    double fatorDesconto;
    if (precoBase > 1000) fatorDesconto = 0.95;
    else fatorDesconto = 0.98;
    return precoBase * fatorDesconto;
}
```

```
double getPreco() {
    final int precoBase = _quantidade * _precoItem;
    final double fatorDesconto;
    if (precoBase > 1000) fatorDesconto = 0.95;
    else fatorDesconto = 0.98;
    return precoBase * fatorDesconto;
}
```

Replace Temp with Query (120)

```
double getPreco() {
    final int precoBase = precoBase();          // 1
    final double fatorDesconto;
    if (precoBase > 1000) fatorDesconto = 0.95; //2
    else fatorDesconto = 0.98;
    return precoBase * fatorDesconto;
}

private int precoBase() {
    return _quantidade * _precoItem;
}
```

Replace Temp with Query (120)

```
double getPreco() {
    final double fatorDesconto;
    if (precoBase() > 1000) fatorDesconto = 0.95; //2
    else fatorDesconto = 0.98;
    return precoBase() * fatorDesconto;
}

private int precoBase() {
    return _quantidade * _precoItem;
}
```

Replace Temp with Query (120)

```
double getPreco() {
    final double fatorDesconto;
    if (precoBase() > 1000) fatorDesconto = 0.95; //2
    else fatorDesconto = 0.98;
    return precoBase() * fatorDesconto;
}

private int fatorDesconto() {
    if (precoBase() > 1000)
        return 0.95;
    return 0.98;
}

private int precoBase() {
    return _quantidade * _precoItem;
}
```

Replace Temp with Query (120)

```
double getPreco() {
    final double fatorDesconto = fatorDesconto();
    return precoBase() * fatorDesconto;
}

private int fatorDesconto() {
    if (precoBase() > 1000)
        return 0.95;
    return 0.98;
}

private int precoBase() {
    return _quantidade * _precoItem;
}
```

Replace Temp with Query (120)
resultado final…

```
double getPreco() {
    return precoBase() * fatorDesconto();
}

private int fatorDesconto() {
    if (precoBase() > 1000)
        return 0.95;
    return 0.98;
}

private int precoBase() {
    return _quantidade * _precoItem;
}
```

Replace Inheritance With Delegation (352)

- **Motivação:** *herança é uma técnica excelente, mas muitas vezes, não é exatamente o que você quer. Às vezes, nós começamos herdando de uma outra classe mas daí descobrimos que precisamos herdar muito pouco da superclasse. Descobrimos que muitas das operações da superclasse não se aplicam à subclasse. Neste caso, delegação é mais apropriado.*

Replace Inheritance With Delegation (352)

- **Resumo:** *Quando uma subclasse só usa parte da funcionalidade da superclasse ou não precisa herdar dados: na subclasse, crie um campo para a superclasse, ajuste os métodos apropriados para delegar para a ex-superclasse e remova a herança.*

Replace Inheritance With Delegation (352)

- **Mecânica:**
  - Crie um campo na subclasse que se refere a uma instância da superclasse, inicialize-o com `this`
  - Mude cada método na subclasse para que use o campo delegado
  - Compile e teste após mudar cada método
    - Cuidado com as chamadas a `super`
  - Remova a herança e crie um novo objeto da superclasse
  - Para cada método da superclasse utilizado, adicione um método delegado
  - Compile e teste

Replace Inheritance With Delegation (352)

Exemplo: pilha subclasse de vetor.

```
Class MyStack extends Vector {

   public void push (Object element) {
        insertElementAt (element, 0);
}

   public Object pop () {
        Object result = firstElement ();
        removeElementAt (0);
        return result;
}
```

Replace Inheritance With Delegation (352)

Crie campo para superclasse.

```
Class MyStack extends Vector {
   private Vector _vector = this;
   public void push (Object element) {
        _vector.insertElementAt (element, 0);
}

   public Object pop () {
        Object result = _vector.firstElement ();
        _vector.removeElementAt (0);
        return result;
}
```

Replace Inheritance With Delegation (352)

Remova herança.

```
Class MyStack extends Vector {
   private Vector _vector = this; new Vector ();
   public void push (Object element) {
        _vector.insertElementAt (element, 0);
   }

   public Object pop () {
        Object result = _vector.firstElement ();
        _vector.removeElementAt (0);
        return result;
   }
```

Replace Inheritance With Delegation (352)

Crie os métodos de delegação que serão necessários.

```
public int size () {
        return _vector.size ();
   }

public int isEmpty () {
        return _vector.isEmpty ();
   }

}// end of class MyStack
```

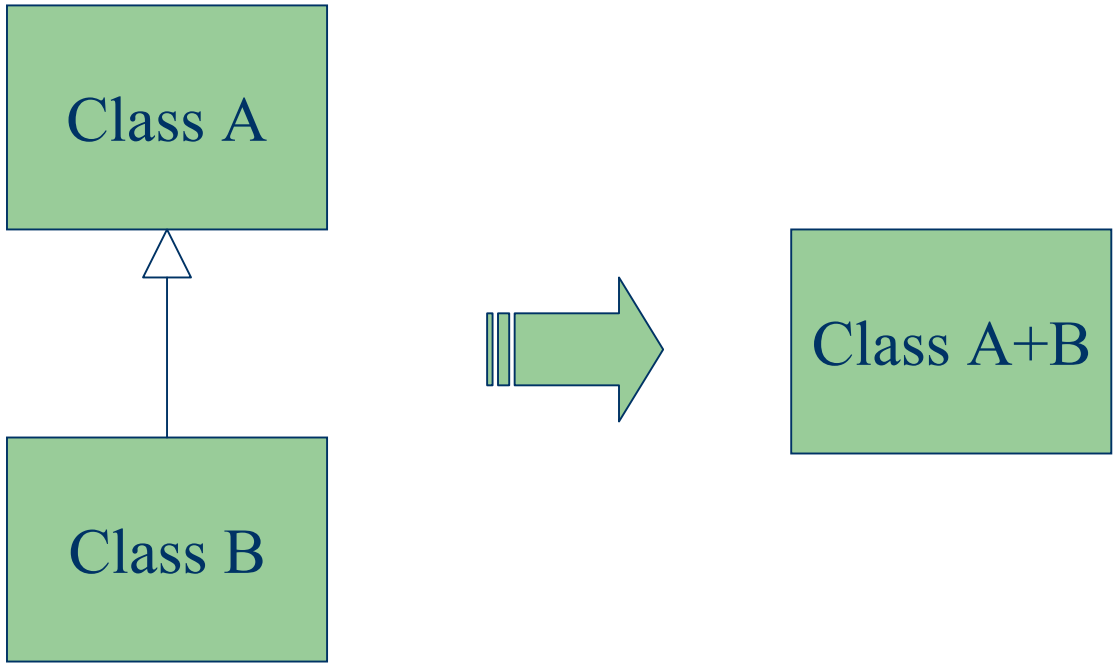
Collapse Hierarchy (344)

- **Resumo:** *A superclasse e a subclasse não são muito diferentes. Combine-as em apenas uma classe.*
- **Motivação:** *Depois de muito trabalhar com uma hierarquia de classes, ela pode se tornar muito complexa. Depois de refatorá-la movendo métodos e campos para cima e para baixo, você pode descobrir que uma subclasse não acrescenta nada ao seu projeto. Remova-a.*

Collapse Hierarchy (344)

- **Mecânica:**
  - Escolha que classe será eliminada: a superclasse ou a subclasse
  - Use Pull Up Field (320) and Pull Up Method (322) ou Push Down Method (328) e Push Down Field (329) para mover todo o comportamento e dados da classe a ser eliminada
  - Compile e teste a cada movimento
  - Ajuste as referências à classe que será eliminada
    - isto afeta: declarações, tipos de parâmetros e construtores.
  - Remova a classe vazia
  - Compile e teste

Collapse Hierarchy (344)
Esquema



Pull UP Field (320)

- **Resumo:** *A superclasse e a subclasse têm os mesmos atributos (field). Mova os atributos repetidos para a superclasse.*
- **Motivação:** *Subclasses desenvolvidas independentemente umas das outras, ou após refatoração, podem ter características (features) duplicadas. Atributos podem ser duplicados. Estes não necessariamente têm o mesmo nome em todas as subclasses. A forma de determinar se tal acontece é verificar como os atributos são usados por outros métodos. Se eles são usados da mesma forma, pode-se generalizá-los.*

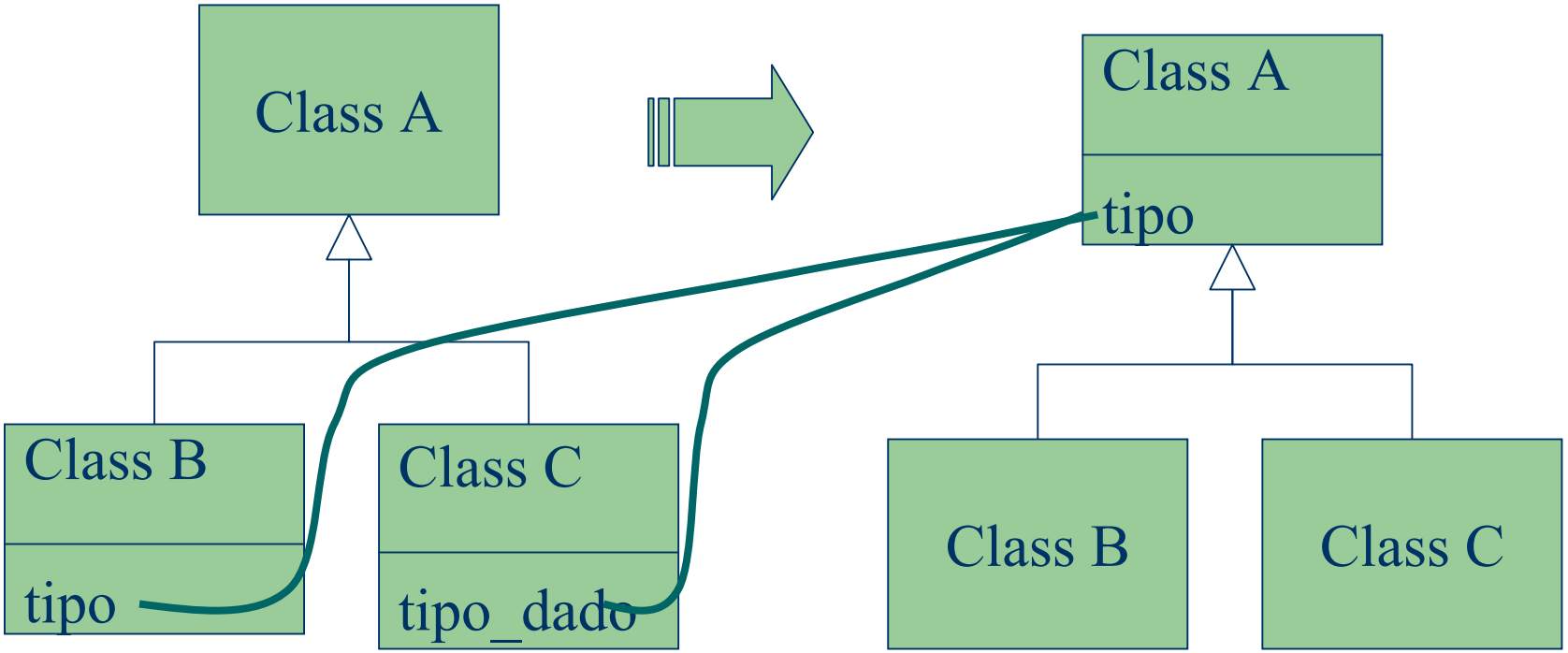
http://sourcemaking.com/refactoring/extract-interface

Pull UP Field (320)

- **Mecânica:**
  - *Inspecione todos os atributos candidatos para ver se não são usados de forma similar.*
  - *Se os atributos não têm o mesmo nome, dê-lhe o nome que vai ser usado quando ele "subir" para a superclasse.*
  - *Compile e teste.*
  - *Crie um novo atributo na superclasse.*
  - ☞ *Se o atributo era privado nas subclasses, deve-se colocá-lo como protegido na superclasse.*
  - *Apague o atributo nas subclasses.*
  - *Compile e teste.*

http://sourcemaking.com/refactoring/extract-interface

Pull UP Field (320)
Esquema



Replace Conditional With Polymorphism (255)

```
class Viajante {
   double getBebida () {
        switch (_type) {
                case ALEMAO:
                        return cerveja;
                case BRASILEIRO:
                        return pinga + limao;
                case AMERICANO:
                        return coca_cola;
        }
        throw new RunTimeException ("Tipo desconhecido!");
   }
}
```

Replace Conditional With Polymorphism (255)

```
class Alemao extends Viajante {
    double getBebida () {
          return cerveja;
    }
}
class Brasileiro extends Viajante {
    double getBebida () {
          return pinga + limao;
    }
}
class Americano extends Viajante {
    double getBebida () {
          return coca_cola;
    }
}
```

Extract Interface

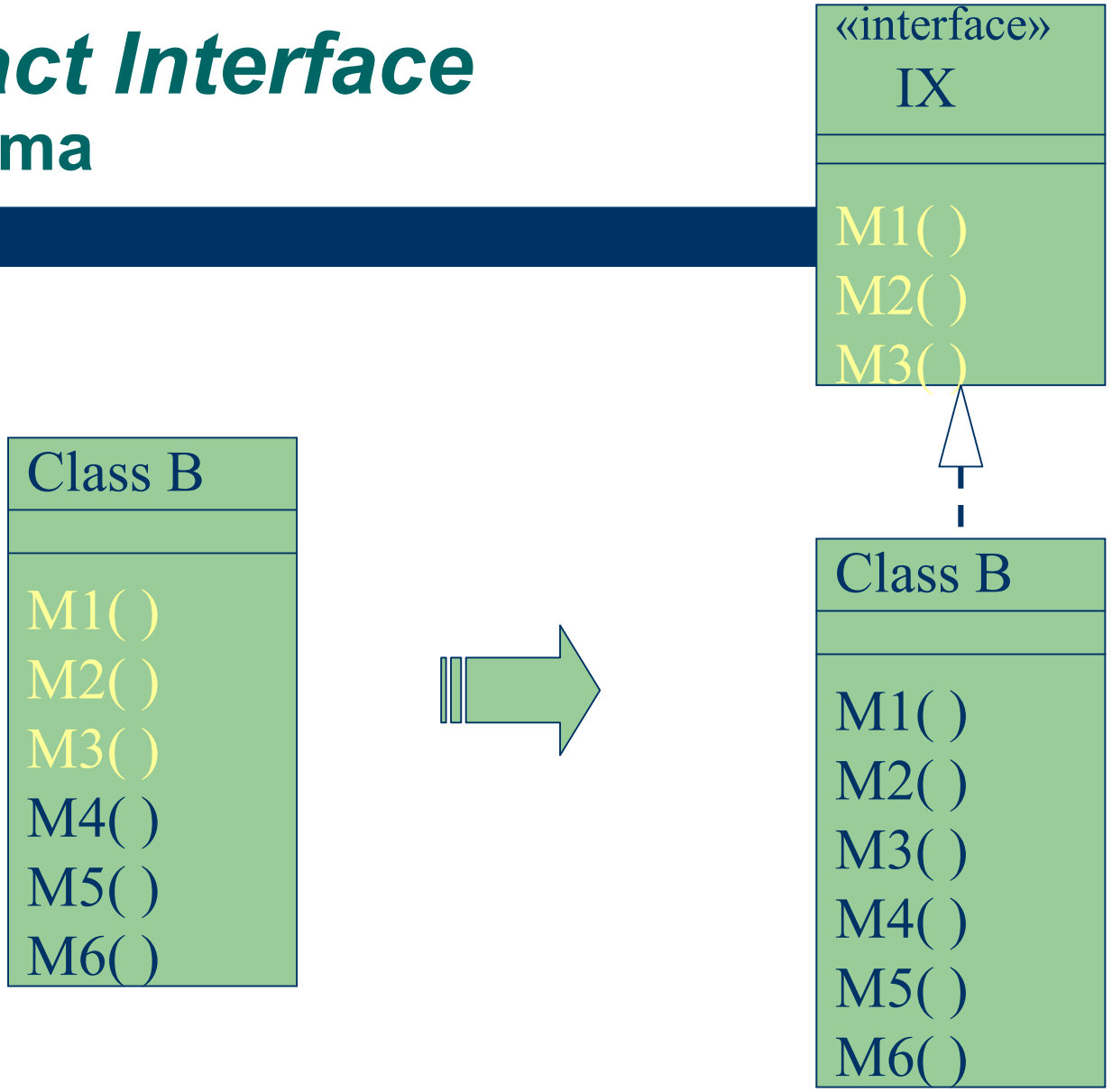
- **Resumo:** *Vários clientes usam o mesmo subconjunto da interface da classe, ou várias classes têm partes de suas interfaces em comum.*
- **Motivação:** *Classes usam umas às outras de várias formas. Pode-se ter, por exemplo, um grupo de clientes usando um subconjunto das responsabilidades da classe. E ainda, uma classe pode colaborar com qualquer classe que trate um certo grupo de requisições. Nestes casos, é útil transformar esse subconjunto de responsabilidades em algo com existência própria, para ficar claro e explícito seu uso no sistema. Dessa forma fica mais fácil de ver como as responsabilidades se dividem. E também, ao se inserir novas classes que implementem esse subconjunto, fica mais fácil de ver o que se encaixa nesse subconjunto.*

Extract Interface

- **Mecânica:**
  - *Crie uma interface vazia.*
  - *Declare as operações comuns na interface.*
  - *Declare as classes relevantes que devem implementar a interface.*
  - *Ajuste as declarações de tipos nos clientes para usar a interface.*

http://sourcemaking.com/refactoring/extract-interface

Extract Interface
Esquema



Princípios Básicos

- Refatoração muda o programa em passos pequenos. Se você comete um erro, é fácil consertar.
- Qualquer um pode escrever código que o computador consegue entender. Bons programadores escrevem código que pessoas conseguem entender.
- Três repetições? Está na hora de refatorar.
- Quando você sente que é preciso escrever um comentário para explicar o código melhor, tente refatorar primeiro.

Ferramentas para Refatoração

- Refactoring Browser Tool.
- Dá suporte automatizado para uma série de refatorações.
- Pode melhorar em muito a produtividade.
- Existem há vários anos para Smalltalk.
- Já há vários para C++ e Java.
- Iniciativas acadêmicas (Ralph@UIUC).
- Agora, integrado no Eclipse e no Visual Works.

Refactoring Browser Tool

- *"It completely changes the way you think about programming".* "Now I use probably half [of the time] refactoring and half entering new code, all at the same speed". Kent Beck.

- A ferramenta torna a refatoração tão simples que nós mudamos a nossa prática de programação.

- `http://st-www.cs.uiuc.edu/~brant/RefactoringBrowser`

Conclusões

- Refatoração é uma técnica para tornar software mais simples, mais fáceis de serem entendidos → mais manutenível.
- Consiste de modificações simples, que não alteram a funcionalidade do código, mas que realizadas repetidamente melhoram muito o software.
- Refatorações foram catalogadas para uso sistemático e são apoiadas por ferramentas.

Bibliografia

- Erich Gamma et al. *Padrões de Projeto – Soluções reutilizáveis de software orientado a objetos, Bookman, 2000.*
- **Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley. 2000.**
- Tom Mens e Tom Tourwé, “A Survey of Software Refactoring”, IEEE Transactions on Software Engineering, 30(2): 126-139, 2004.
- Tom Tourwé e Tom Mens, “Identifying Refactoring Opportunities Using Logic Meta Programming”, Proceedings of the Seventh European Conference on Software Maintenance and Reengineering (CSMR ’03), 2003.
- William F. Opdyke, *Refactoring Object-oriented Frameworks*. Ph.D thesis, University of Illinois at Urbana-Champaign, 1992.

Informações

- www.refactoring.com
- www.ime.usp.br/~kon/presentations