



Auditoria e Qualidade de Sistemas

Prof. Edgard Davidson C. Cardoso



Qualidade de Sistemas

REFATORAÇÃO - MELHORANDO A QUALIDADE DE CÓDIGO

Refatoração



Kent Beck

“Não sou um grande programador; sou apenas um bom programador com ótimos hábitos. Refatorar me ajuda a ser muito mais efetivo na escrita de código robusto”



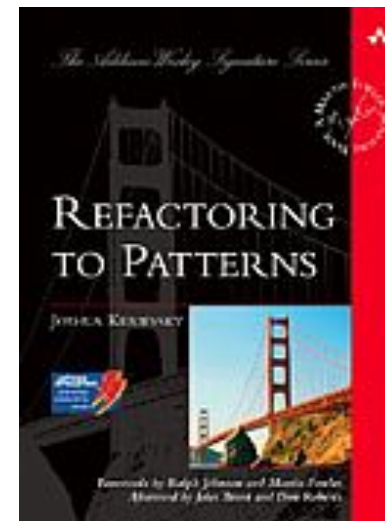
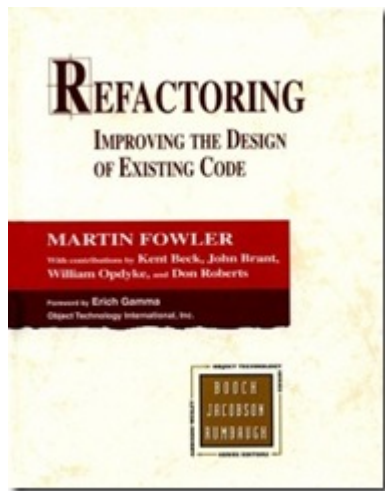
O que é Refatoração?

- **Refatoração (Substantivo):** *“uma alteração feita na estrutura interna do software para torná-lo mais fácil de ser entendido e menos de ser modificado sem alterar seu comportamento observável.”*
- **Refatoração (verbo):** *“reestruturar software aplicando uma série de Refatorações sem alterar seu comportamento observável.”*

Referências

- Livros:

- Martin Fowler et al, “Refactoring: Improving the Design of Existing Code”, Addison-Wesley Professional, 1999
- Joshua Kerievsky, “Refactoring to Patterns”, Addison- Wesley, 2004





Referências

- OnLine:
 - www.refactoring.com
 - <http://www.industriallogic.com/xp/refactoring/catalog.html>

O espírito da refatoração



Antes

→
(refatoração)



Depois



Refatoração

- Uma modificação no sistema que **não altera o seu comportamento funcional**, mas **melhora sua estrutura interna**.
- Limpa o código minimizando as chances de introduzir erros.
- Melhorar o *design* depois que o código foi escrito.



Por que você deve Refatorar?

- Refatorar melhora o projeto de software
- Refatorar torna o software mais fácil de entender
- Refatorar ajuda a encontrar falhas
- Refatorar ajuda a programar mais rapidamente



Quando você deve Refatorar?

- A regra de três
- Refatore quando acrescentar funções
- Refatore quando precisar consertar uma falha
- Refatore enquanto revisa o código



“Mau cheiros” no código

“If it stinks, change it.” (Se cheirar mal, troque-o)
Avó do Kent beck falando como cuidar de bebês



Alguns maus cheiros

- Código duplicado
- Método muito longo
- Classe muito grande
- Muitos parâmetros
- Alteração Divergente
- Nomes de variáveis obscuros
- Intimidade inapropriada
- Comentários
- Existem mais...



Primeiros Passos - Teste

- 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 têm que ser automáticos e ser capazes de se auto-verificarem.



Problemas com Refatoração

- Refatoração de sistemas grandes ou enormes
- Refatoração com bancos de dados
- Refatoração de APIs públicas
- Quando NÃO refatorar?
 - Quando é tão ruim que reescrever é melhor
 - Quando você está próximo de um prazo!



Catálogo de Refatorações

- [Fowler, 2000] “Refatoração”, contém 72 refatorações.
- Análogo aos padrões de desenho orientado a objetos [Gamma et al. 1995] (GoF).
- [Kerievsky, 2004] “Refactoring to patterns”, catálogo com 27 refatorações que aplicam padrões!



Formato de refatorações nos catálogos

- **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.



Extrair Método (100)

- **Nome:** *Extrair Método*
- **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.*
- **Motivação:** *é um dos refatoramentos 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.*



Extrair Método (100)

Mecânica:

- Crie um novo método e escolha um nome que explicita 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 usados 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.**



Extrair Método (100)

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);  
}
```



Extrair Método (100)

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 ("*****");
}
```



Extrair Método (100)

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 (divida)  
{  
    System.out.println("nome: " + _nome);  
    System.out.println("divida total: " + divida);  
}
```



Extrair Método (100)

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;  
}
```



Extrair Método (100)

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;  
}
```



Extrair Método (100)

e se o código fosse

```
void imprimeDivida (double dividaAntiga) {
    Enumerate e = _pedidos.elementos ();
    double divida = dividaAntiga * 1.2;
    imprimeCabecalho ();
    // calcula dívidas
    while (e.temMaisElementos ()) {
        Order cada = (Order) e.proximoElemento ();
        divida += cada.valor ();
    }
    imprimeDetalhes (divida);
}

void imprimeDetalhes (divida)
{
    System.out.println("nome: " + _nome);
    System.out.println("divida total: " + divida);
}
```




Extrair Método (100)

solução que recebe e retorna

```
void imprimeDivida (double dividaAntiga) {  
    imprimeCabecalho ();  
    double divida = calculaDivida (dividaAntiga * 1.2);  
    imprimeDetalhes (divida);  
}
```

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



Extrair Método (100)

- E se mais do que um valor deve ser retornado ?
 - Escolha um subconjunto do código;
 - Pode-se passar por referência (evitar);
- O quê fazer com variáveis temporárias?
 - Usar Replace Temp with Query (120)
 - Ou trocar método por método objeto (135)



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 mau organizados, aplique *Inline Method* em todos eles seguido de uns bons *Extract Methods*.



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.

- **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 - 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)

```
// finalmente
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)

- **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.*
- **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)

- **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)

Crio 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)

Removo 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)

Crio 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 desenho. 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 a classe que será eliminada
 - isto afeta: declarações, tipos de parâmetros e construtores.
 - Remove a classe vazia
 - Compile e teste



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;  
    }  
}
```



Introduce Null Object (260)

```
Result meuORBCorba (String parametros[])
{
    Result r;
    if (pre_interceptor != NULL)
        pre_interceptor.chamada ();
    if (meuObjeto != NULL && meuObjeto.metodo != NULL)
        r = meuObjeto.metodo.invoke (parametros);
    if (pos_interceptor != NULL)
        r = pos_interceptor.chamada (r);
    return r;
}
```



Introduce Null Object (260)

- Substitua o valor NULL por um objeto do tipo Nulo.

```
Result meuORBCorba (String parametros[])  
{  
    pre_interceptor.chamada ();  
    Result r = meuObjeto.metodo.invoke (parametros);  
    return pos_interceptor.chamada (r);  
}
```

```
class Pre_InterceptorNulo {  
    void chamada () {}  
}  
class MeuObjetoNulo {  
    MetodoCORBA metodo () { return MetodoCORBANulo; }  
}
```




Princípio Básico

Quando o código cheira mau, refatore-o!

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



Princípio Básico

Quando o código cheira mau, refatore-o!

Cheiro	Refatoramento a ser aplicado
Comentários (desodorante 😊)	Extract Method (110) Introduce Assertion (267)
Muitos parâmetros	Replace Parameter with Method (292) Preseve Whole Object (288) Introduce Parameter Object (295)



Outros Princípios Básicos

- Refatoramento, 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.