

Refatoração:

Melhorando a Qualidade
de Código Pré-Existente



Cursos de Verão 2007 – IME/USP

www.agilcoop.org.br

Alexandre Freire & Paulo Cheque

Refatoração

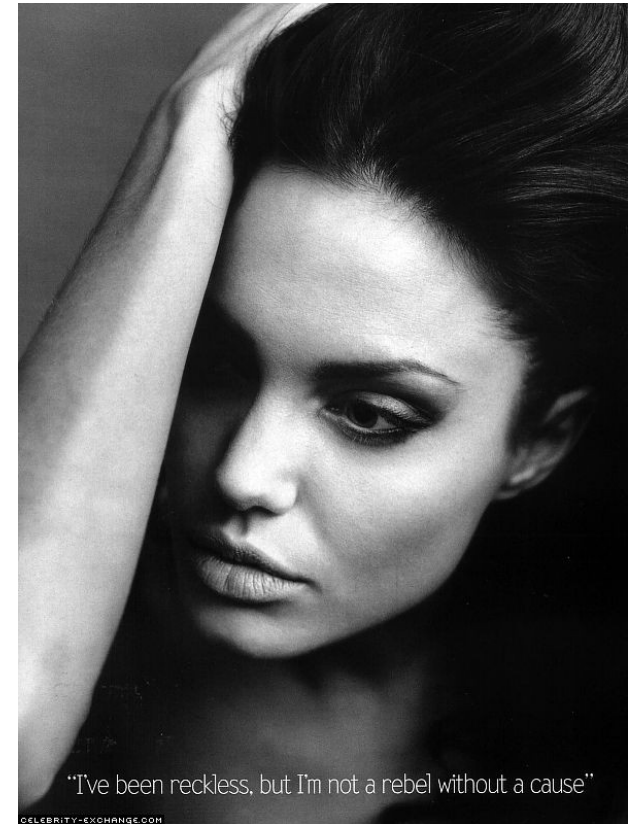
- Uma [pequena] modificação no sistema que não altera o seu comportamento funcional,
- mas que torna o código mais fácil de ser entendido e menos custoso de ser alterado:
 - simplicidade
 - flexibilidade
 - clareza
- O desempenho pode piorar, otimizar não é o objetivo da refatoração

O espírito da Refatoração



Antes

→
(refatoração)



Depois

Exemplos de refatoração

- Alterar nomes de variáveis, métodos e objetos para melhorar legibilidade do código
 - `String calc_imp_rnd_str(int a, TxS tx, int inc) -> String calculaImpostoDeRenda(int ano, Imposto taxas, int rendimento)`
- Mudar parâmetros de métodos para que fiquem mais claros
- Melhorar o design da arquitetura
- Eliminar duplicação de código
- Flexibilizar métodos para novos usos

Refatoração sempre existiu...

- Melhorar o código sempre foi uma prática implícita
- Com sua popularização (e aceitação acadêmica) surge um vocabulário de refatorações comuns e um catálogo compartilhado
- Assim podemos utilizá-las mais sistematicamente.
- Podemos aprender novas técnicas, ensinar uns aos outros.

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.

Catálogo de Refatorações

- [Fowler, 2000] “Refatoração”, contém 72 refatorações.
- Análogo aos padrões de desenho orientado a objetos [Gamma et al. 1995] (GoF).
- Vale a pena gastar algumas horas com este catálogo. (GoF é obrigatório, não tem opção).
- [Kerievsky, 2004] “Refactoring to patterns”, catálogo recente com 27 refatorações que aplicam padrões!

Por que refatorar?

- Não resolve todos os problemas
- Melhora o projeto do software
- Torna o software mais fácil de entender
- Ajuda a encontrar falhas
- Ajuda a programar mais rapidamente

Quando Refatorar

- 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.

Sintomas que o código merece ser refatorado

- Quando você encontra código antigo que não entende de primeira
- Ao ler código feito por outros programadores, e perceber que ele não está claro (durante revisões por exemplo)
- Quando precisa consertar uma falha, ou adicionar funcionalidade
- Regra dos 3, ou mais eXtremo, *Once and Only Once*

O Primeiro Passo em Qualquer 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.
 - porém, como são feitas em pequenos passos, é fácil recuperar-se de uma falha
- Os testes vão ajudá-lo a detectar erros se eles forem criados.

Pequenos passos

- Em geral, um *passo de refatoração* é tão simples que parece que ele não vai ajudar muito.
 - Cada passo é trivial.
 - Demora alguns segundos ou alguns poucos minutos para ser realizado.
 - É uma operação sistemática e óbvia.
- Mas quando se juntam 50 passos, bem escolhidos, em seqüência, o código melhora radicalmente.

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

- **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:**

- *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

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

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 ()) {
        Pedido cada = (Pedido) e.proximoElemento ();
        divida += cada.valor ();
    }
    // imprime detalhes
    System.out.println ("nome: " + _nome);
    System.out.println ("divida total: " + divida);
}
```

Extrair Método

exemplo sem variáveis locais

```
void imprimeDivida () {
    Enumerate e = _pedidos.elementos ();
    double divida = 0.0;
    imprimeCabecalho ();
    // calcula dívidas
    while (e.temMaisElementos ()) {
        Pedido cada = (Pedido) 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

exemplo com variáveis locais

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

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

Extrair Método

exemplo com atribuição

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

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

Extrair Método

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.

Internalizar Método

- **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 *Internalizar Método* em todos eles seguido de uns bons *Extrair Métodos*.

Internalizar Método

- **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.

Internalizar Método

- **Exemplo:**

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

- **Refatoração:**

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

Substituir variável temporária por consulta

- **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.

Substituir variável temporária por consulta

- **Mecânica:**

- Encontre variáveis locais que são atribuídas uma única vez
 - Se *temp* é atribuída mais do que uma vez use *Dividir Variável Temporária*
- Declare *temp* como final
- Compile (para ter certeza)
- Extraia a expressão
 - Método privado - efeitos colaterais
- Compile e teste

Substituir variável temporária por consulta

```
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;
}
```

Substituir variável temporária por consulta

```
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;
}
```

Substituir variável temporária por consulta

```
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;
}
```

Substituir variável temporária por consulta

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

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

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

Substituir Herança por Delegação

- **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.*

Substituir Herança por Delegação

- **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

Substituir Herança por Delegação

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;  
    }  
}
```

Substituir Herança por Delegação

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;  
    }  
}
```

Substituir Herança por Delegação

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;  
    }  
}
```

Substituir Herança por Delegação

Crio os métodos de delegação que serão necessários:

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

Condensar Hierarquia

- **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.*

Condensar Hierarquia

- **Mecânica:**
 - Escolha que classe será eliminada: a superclasse ou a subclasse
 - Utilize uma refatoração para mover todo o comportamento e dados da classe a ser eliminada (e.g.: Mover Campo, Mover Método...)
 - 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

Substituir comando condicional por polimorfismo

```
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!");  
    }  
}
```

Substituir comando condicional por polimorfismo

```
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; }  
}
```

Introduzir Objeto Nulo

```
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;  
}
```

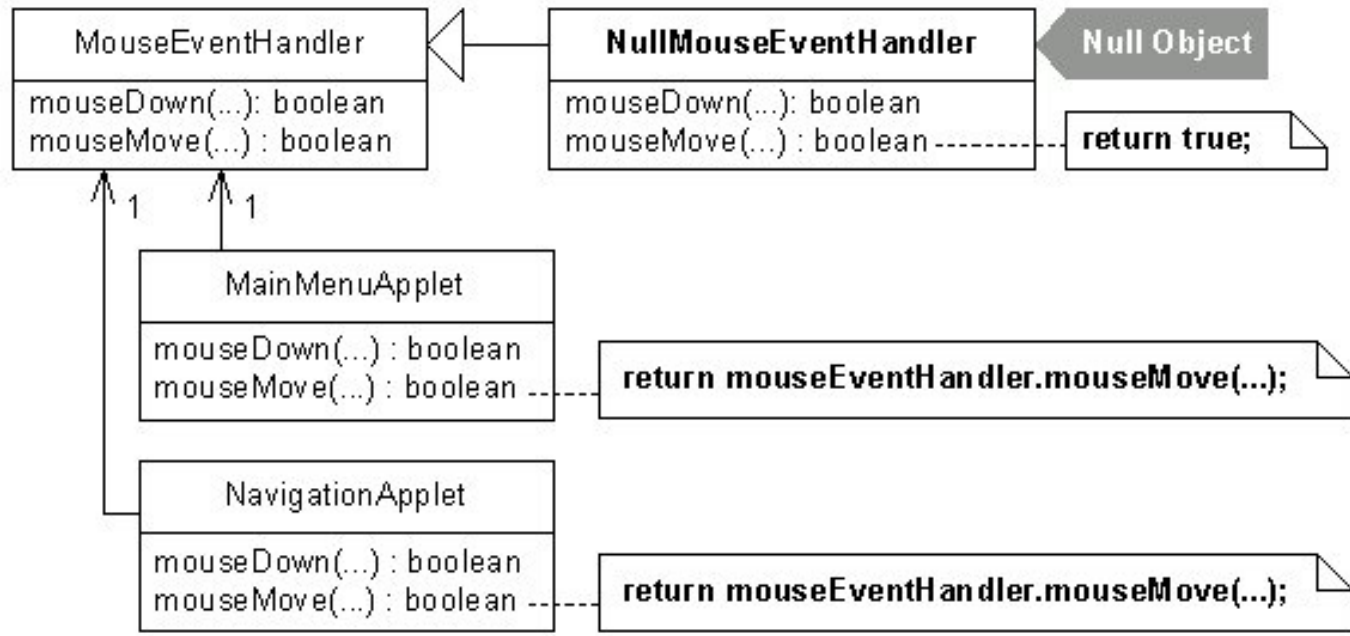
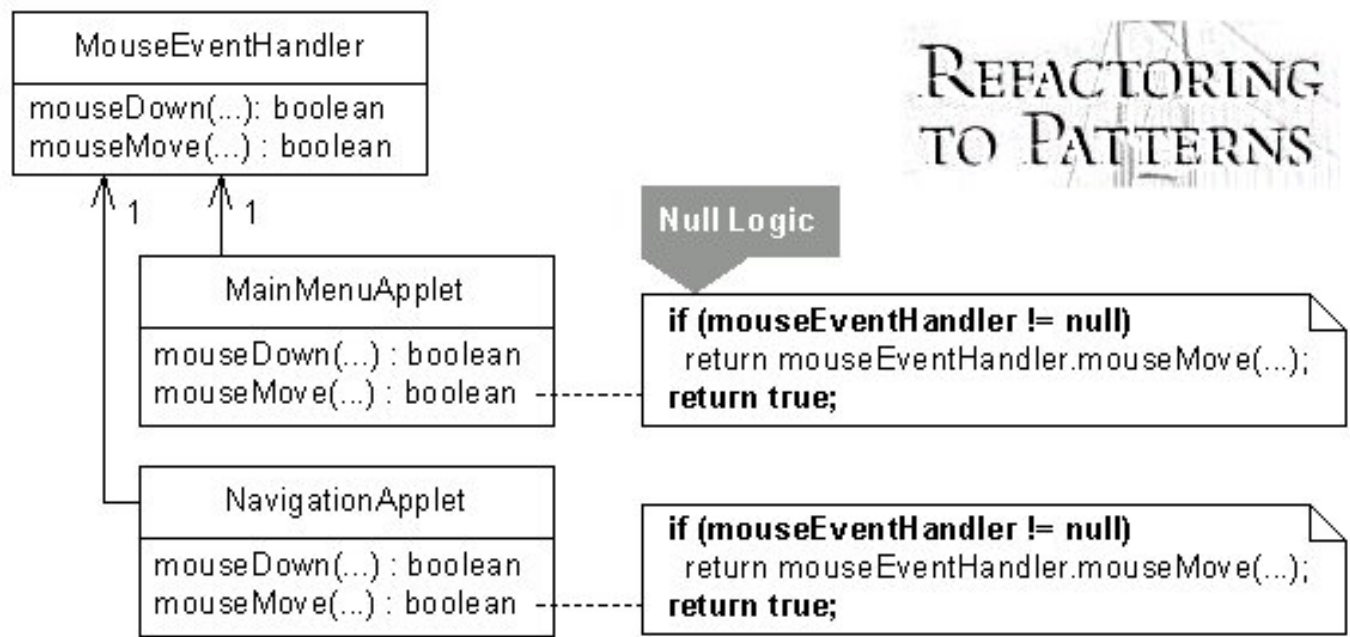
Introduzir Objeto Nulo

- 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 extends Pre_Interceptor {  
    void chamada () {}  
}
```

```
class MeuObjetoNulo extends MeuObjeto {  
    MetodoCORBA metodo () { return MetodoCORBANulo; }  
}
```



Encadeando Construtores

- **Resumo:** *Seus construtores duplicam código*
- **Mecânica:**
 - *Encadear os construtores em sequência para evitar ao máximo código repetido*

```
public Pessoa(String nome, int idade){  
    this.nome = nome;this.idade = idade;  
}
```

```
public Pessoa(String nome, int idade, Endereco  
    endereco){  
    this.nome = nome;this.idade = idade;  
    this.endereco = endereco;  
}
```

Encadeando Construtores

```
public Pessoa(String nome, int idade){
    this.nome = nome;this.idade = idade;
}

public Pessoa(String nome, int idade, Endereco
endereco) {
    this(nome, idade);
    this.endereco = endereco;
}

public Pessoa(String nome, int idade, Endereco
endereco, Profissao profissao){
    this(nome, idade, endereco);
    this.profissao = profissao;
}
```

Sintoma básico

- Quando o código cheira mal, refatore-o!

Cheiro	Refatoração a ser aplicada
Código duplicado	Extrair Método Substituir o Algoritmo
Método muito longo	Extrair Método Substituir variável temporária por consulta Introduzir Objeto Parâmetro
Classe muito grande	Extrair Classe Extrair Subclasse Extrair Interface Duplicar Dados Observados
Intimidade inapropriada	Mover Método Mover Campo Substituir Herança por Delegação

Sintoma básico

Comentários	Extrair Método Introduzir Asserção
Muitos parâmetros	Substituir Parâmetro por Método Preservar o Objeto Inteiro Introduzir Objeto Parâmetro

Mais princípios básicos

- Qualquer um pode escrever código que o computador consegue entender. Bons programadores escrevem código que pessoas conseguem entender.
- Quando você sente que é preciso escrever um comentário para explicar o código melhor, tente refatorar primeiro.
- testes tem que ser automáticos e ser capazes de se auto-verificarem.

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 várias para C++ e Java.
- Iniciativas acadêmicas (Ralph@UIUC).
- Agora, integrado no Eclipse e no Visual Works.

Navegador de Refatorações

- "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.
- É importante superar o "Refactoring Rubicon"
 - Extrair Método

Referências

- www.refactoring.com
- Martin Fowler. *Refactoring: improving the design of existing code.*
- Kerievsky. *Refactoring to patterns*