

# Refatoração - Uma abordagem simples e direta

Rafael Alfredo Capucho

22 de junho de 2009

Universidade Federal de São Paulo - UNIFESP

Versão 1.1

<http://blog.rafaelcapucho.com>

# Sumário

<b>1</b>	<b>Introdução</b>	<b>3</b>
1.1	Quando utilizar . . . . .	3
1.2	Metodologias Ágeis . . . . .	3
1.3	Desenvolvimento Orientado a Testes . . . . .	3
1.3.1	Relação entre Refatoração e TDD . . . . .	4
<b>2</b>	<b>Técnicas de Refatoração</b>	<b>4</b>
2.1	Replace Temp with Query . . . . .	4
2.2	Split Temporary Variable . . . . .	5
2.3	Extract Method . . . . .	6
<b>3</b>		<b>7</b>
<b>4</b>	<b>Bibliografia</b>	<b>7</b>

# 1 Introdução

O termo refatoração remete ao processo de reconstrução de uma determinada parte do código-fonte de um sistema sem que sua funcionalidade seja alterada. Refatoração não deve ser usada para adicionar novas funcionalidades nem corrigir problemas funcionais de um sistema e sim na melhoria da arquitetura de software pré-existente, Aplicando determinados padrões conseguimos eliminar códigos desnecessários tornando a arquitetura mais legível e organizada, aumentando a manutenibilidade do código-fonte em questão.

## 1.1 Quando utilizar

Não existe uma definição precisa de quando devemos refatorar um bloco de código, porém com certa experiência conseguimos perceber quando um código deve ser refatorado, segundo Kent um dos criadores do <sup>1</sup>Extreme Programming a refatoração de código se torna necessária quando o código cheira mal (<sup>2</sup>Bad Smells) ou seja, quando percebemos que existem maneiras mais diretas e mais claras de escrever o mesmo código.

## 1.2 Metodologias Ágeis

Metodologias ágeis é um conjunto de metodologias que podem ser empregadas por equipes de desenvolvimento com a intenção de melhorar o processo de desenvolvimento. Essas metodologias definem diversos padrões de desenvolvimento o que permite no final do processo ter um desenvolvimento mais rápido e seguro.

## 1.3 Desenvolvimento Orientado a Testes

Desenvolvimento Orientado a Testes ou TDD (Test-Driven Development) é uma técnica de desenvolvimento de software que consiste em garantir a integridade funcional do sistema por meio de testes contínuos que podem apontar possíveis problemas estruturais e funcionais. Cada teste é feito de maneira automatizada de acordo com regras pré-definidas pelo programador. Existem diversas frameworks para teste de unidade disponíveis o mais usado

---

<sup>1</sup>Extreme Programming ou XP é uma metodologia de desenvolvimento ágil para pequenas equipes na qual o processo é feito em pequenas iterações junto ao cliente podendo sofrer alterações durante o desenvolvimento do software

<sup>2</sup>Bad Smells ou cheirar mal é uma expressão usada para definir que um determinado código deve ser refatorado.

para Java atualmente é o framework JUnit criado por Kent Beck e Erich Gamma.

### 1.3.1 Relação entre Refatoração e TDD

Na <sup>3</sup>introdução citamos que o processo de refatoração implica em uma mudança estrutural do código sem que seja alterada a funcionalidade externa. Utilizando TDD junto com a Refatoração de código podemos comprovar que a sua funcionalidade não foi prejudicada após aplicada a refatoração. Tal comprovação será o resultado proveniente dos testes automatizados previamente estipulados.

## 2 Técnicas de Refatoração

Segundo Martin Fowler, sempre que for necessário refatorar o código-fonte os passos são sempre os mesmos, precisa-se construir uma base sólida com testes de unidade para todas as seções do código. Os testes são essenciais para evitar a inserção de possíveis bugs no seu código, Fowler diz também que continua sendo humano e como todo humano continua cometendo erros. Então precisamos de uma base sólida de testes. Não será abordado neste documento o uso de testes de unidade assim como metodologias ágeis, pois são assuntos complexos que merecem atenção, deixaremos para tratar sobre esses assuntos separadamente. No capítulo seguinte será apresentado as principais técnicas de refatoração criadas por Martin Fowler.

### 2.1 Replace Temp with Query

Essa técnica consiste em trocar variáveis temporárias que contêm expressões matemáticas por métodos, para que possam ser chamados em outros métodos aumentando a reusabilidade e manutenibilidade da expressão em questão.

```
public String whereIsTheObject(){
    double energiaPotencial = this.massa * this.gravitacional * this.altura;
    if(energiaPotencial == 0){
        return "O objeto está no chão, Energia Potencial: " + energiaPotencial;
    } else {
        return "O Objeto não está no chão, Energia Potencial: " + energiaPotencial;
    }
}
```

---

<sup>3</sup>Vide Seção 1

O método acima faz uso do cálculo da energia potencial armazenando o resultado em uma variável temporária de escopo local, dessa maneira não podemos calcular a energia potencial desse objeto em outro método sem declarar novamente a expressão matemática. Evitar o uso de variáveis temporárias pode te poupar várias linhas de código desnecessárias. Você provavelmente está se perguntando sobre performance nesses casos, tanto esse quanto outros problemas de otimização não serão tratados no momento. Você deverá resolver os problemas de otimização quando o seu código já estiver fatorado, o que implica em maior poder de otimização. Aplicando Replace Temp with Query iremos obter algo semelhante a isso:

```
public double energiaPotencial(){
    return getMassa() * getGravitacional() * getAltura();
}

public String whereIsTheObject(){
    if(energiaPotencial() == 0)
        return "O objeto está no chão, Energia Potencial: " + energiaPotencial();
    else
        return "O objeto não está no chão, Energia Potencial: " + energiaPotencial();
}
```

## 2.2 Split Temporary Variable

Caracteriza-se por ter uma ou várias variáveis temporárias com valores designados que são frequentemente alterados, exceto quando forem variáveis de loop ou coleções. O problema afeta diretamente a clareza do seu código, sendo que em um dado momento uma variável vale X e em outro a mesma variável poderá valer Y.

```
double temporario = getMassa() * getGravitacional() * getAltura();
System.out.println("Energia Potencial: " + temporario);
temporario = getMassa() * (getVelocidade()*getVelocidade()) / 2;
System.out.println("Energia Cinética: " + temporario);
```

A cada vez que a variável altera o seu valor, aumenta-se também a sua responsabilidade dentro do método. Qualquer variável com mais de uma responsabilidade deve ser refatorada de maneira que cada variável tenha apenas uma responsabilidade. Usar uma variável para designar várias coisas pode ser bastante confuso para o leitor do código. Lembre-se que a refatoração de código tem como princípio melhorar a organização do código para ser mais legível aos humanos, e não as máquinas. Para resolver nosso problema podemos criar variáveis temporárias como apenas uma responsabilidade e declará-las como sendo *final*. Aplicando Split Temporary Variable iremos

obter algo semelhante a isso:

```
final double energiaPotencial = getMassa() * getGravitacional() * getAltura();
System.out.println("Energia Potencial: " + temporario);
final double energiaCinetica = getMassa() * (getVelocidade()*getVelocidade()) / 2;
System.out.println("Energia Cinética: " + temporario);
```

## 2.3 Extract Method

O Método de extração representa o processo de agrupar blocos de código semelhantes em vários métodos em apenas um, criando uma referência de todos os métodos para o novo método, assim além de reduzir consideravelmente o número de linhas do seu código, você conseguirá em caso de manutenção, facilmente alterar esse método, que deve ser criado com o mínimo de responsabilidade possível.

```
public void imprimirRelatorio(){
    imprimirCadastrosPendentes();
    for(int i = 0; i < status.lenght(); i++){
        System.out.println("Status do relatorio " + i + " : " + status[i]);
    }
}

public void relatoriosPendentes(){
    imprimirHorarios();
    for(int i = 0; i < status.lenght(); i++){
        System.out.println("Status do relatorio " + i + " : " + status[i]);
    }
}
```

Dessa maneira estaremos replicando código, ao aplicar o método de extração chegaremos em algo como:

```
public void imprimirStatus(){
    for(int i = 0; i < status.lenght(); i++){
        System.out.println("Status do relatorio " + i + " : " + status[i]);
    }
}

public void imprimirRelatorio(){
    imprimirCadastrosPendentes();
    imprimirStatus();
}

public void relatoriosPendentes(){
    imprimirHorarios();
    imprimirStatus();
}
```

### 3

## 4 Bibliografia

- [1] BECK. Kent, *Extreme programing explained: embrace change*, Addison-Wesley.(2000)
- [2] FOWLER. Martin, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional.(1999)