

Um Processo Para Avaliação Quantitativa de Refatorações de Software

Luiza Figueiredo Pagliari, Dalto José Nunes

¹Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{lfpagliari,daltro}@inf.ufrgs.br

Resumo

Several software refactorings have been proposed on the last years for different programming paradigms. Their benefits are often given based on qualitative assessments, without quantitative arguments to prove the existence of these benefits. This paper presents a process to get a quantitative assessment of refactorings using software metrics. It splits the refactoring steps into atomic changes and measures the effects of these changes on some metrics. As result, we have the impact of the refactoring on each of the metric chosen, and know the consequences of the refactoring on the code before applying it.

1. Introdução

Programas devem evoluir constantemente para se adaptar às mudanças dos requisitos ocorridas durante seu ciclo de vida. Essas mudanças são inevitáveis e tornam o código mais complexo e o distancia de seu projeto original, o que acaba diminuindo a qualidade do software [Mens and Tourwé, 2004]. Para contornar este problema, devem ser feitas periodicamente reorganizações internas do programa para facilitar sua compreensão e futura manutenção, porém sem alterar seu comportamento. Essas reorganizações são chamadas de refatoração de software.

A literatura apresenta uma grande quantidade de refatorações definidas para diversos paradigmas de programação e com o objetivo de melhorar diferentes atributos de qualidade. Algumas aumentam o reuso, outras diminuem a complexidade, existem ainda as que melhoram a modularidade do software, etc. No entanto, a comprovação de suas melhorias é frequentemente feita através de avaliações qualitativas das mesmas, sendo poucos os trabalhos que as avaliam quantitativamente para comprovar a existência de seus benefícios.

Este trabalho apresenta um processo para obter uma avaliação quantitativa de refatorações através do uso de métricas de software. Ele é definido de forma a não ser específico para um determinado paradigma de programação, o que permite seu uso para avaliar refatorações para diferentes tipos de programa. A aplicação do processo permite conhecer quais os impactos de uma refatoração em diversos atributos de qualidade, sejam eles previstos em sua descrição, sejam eles efeitos colaterais de seus passos. De posse dessas informações, o desenvolvedor pode decidir se a refatoração deve ser usada antes de executá-la, evitando assim fazer modificações que piorem o software e ter de desfazer as alterações caso os resultados não sejam adequados.

O restante do artigo está organizado da seguinte maneira: a seção 2 apresenta conceitos básicos de refatoração de software; na seção 3 é descrito o processo de avaliação; a seção 4 traz um exemplo de aplicação do processo; a seção 5 mostra as vantagens e limitações da proposta, bem como descreve oportunidades para aplicar o processo; na seção 6 são mostrados alguns dos trabalhos relacionados a este artigo; e a seção 7 apresenta as conclusões do trabalho.

2. Refatoração de Código

O termo “refatoração” foi originalmente usado por William Opdyke [Opdyke, 1992] e significa “uma alteração feita na estrutura interna do software para torná-lo mais fácil de ser entendido e menos custoso de ser modificado, sem alterar seu comportamento observável” [Fowler et al., 1999]. Refatorações são aplicadas quando se deseja melhorar algum atributo de qualidade do software, como modularidade, reusabilidade, complexidade, manutenibilidade, etc. [Mens and Tourwé, 2004]. A descrição da refatoração costuma explicar a forma que o software será melhorado com sua aplicação, porém em alguns casos existem efeitos colaterais (benéficos ou não) que não são citados por não fazerem parte do objetivo principal da refatoração. Uma avaliação mais profunda de diferentes atributos de qualidade pode esclarecer todos os efeitos que a refatoração terá.

Existem muitos trabalhos com descrições de refatorações para diferentes paradigmas de programação, como imperativo [Garrido and Johnson, 2002], orientado a objetos [Kerievsky, 2004] e orientado a aspectos [Monteiro and Fernandes, 2005]. Uma importante contribuição foi o catálogo apresentado por Fowler et al. [Fowler et al., 1999], com 72 refatorações orientadas a objetos. Ele reuniu pela primeira vez uma grande quantidade de refatorações, organizou-as em categorias e estabeleceu um formato para descrevê-las. Este formato divide as explicações da refatoração em cinco partes:

Nome: importante para futuras referências à refatoração;

Resumo: descreve a situação em que a modificação se torna necessária e qual ação deve ser executada;

Motivação: mostra as razões para executar a refatoração e quando ela não deve ser aplicada;

Mecânica: descreve passo-a-passo como executar a refatoração, o que pode incluir a aplicação de outra refatoração em algum dos passos;

Exemplos: mostra um uso bem simples para ilustrar como a refatoração funciona.

Existem pares de refatorações que executam modificações inversas no código, como *Extrair Método* e *Internalizar Método* ou *Subir Campo na Hierarquia* e *Descer Campo na Hierarquia* [Fowler et al., 1999]. Elas são chamadas de “refatorações opostas”, e são muito importantes por permitirem ao desenvolvedor desfazer mais facilmente uma refatoração quando os resultados não forem adequados [Monteiro, 2005]. Caso uma refatoração não tenha uma oposta definida, ela até poderá ser desfeita, porém será um processo muito mais complicado e propenso à inserção de erros no programa. Avaliar uma refatoração que não tenha uma oposta permite conhecer seus impactos em atributos de qualidade antes de aplicá-la, e evita a possibilidade de introdução de erros caso ela tenha que ser desfeita.

O processo de refatoração, porém, não se resume apenas a aplicar um conjunto de passos em seqüência. Mens e Tourwé [Mens and Tourwé, 2004] identificam outras cinco atividades deste processo, no qual a aplicação dos passos é apenas uma etapa intermediária:

1. Identificar uma oportunidade de refatoração;
2. Determinar quais refatorações devem ser aplicadas;
3. Garantir que as refatorações não alterarão o comportamento;
4. Aplicar a refatoração;
5. Avaliar os efeitos da refatoração em características de qualidade do software (ex.: complexidade, manutenibilidade) ou do processo (ex.: produtividade, custos);
6. Manter a consistência entre o código refatorado e outros artefatos de software (documentação, especificação de requisitos, testes, etc.).

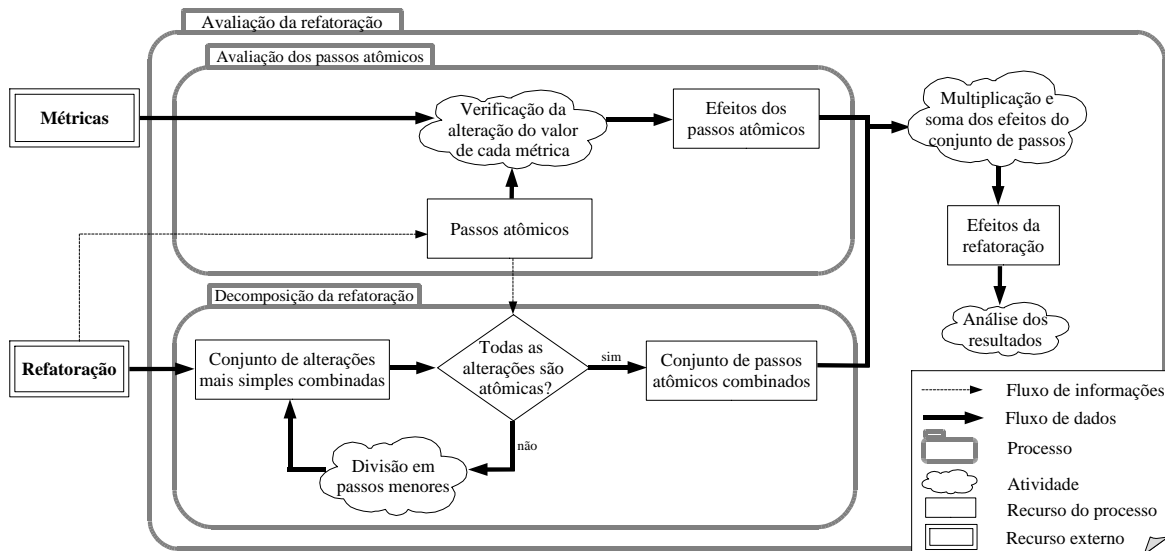


Figura 1: O processo de avaliação proposto

O processo de avaliação de refatorações proposto neste trabalho está relacionado a duas das atividades descritas acima: à 5ª, porque a aplicação do processo permite obter o impacto das refatorações em atributos de qualidade de software; e à 2ª, pois caso a oportunidade de refatoração seja obtida através de medições de atributos de qualidade, será possível escolher as refatorações a serem aplicadas tomando como base seus impactos nesses atributos.

3. Processo de Avaliação

Nesta seção será apresentada a proposta para um processo de avaliação de refatorações de software, representado na figura 1. O processo proposto possui dois processos menores que dividem a avaliação em duas frentes: a análise dos efeitos de pequenos passos no valor das métricas e a decomposição da refatoração em passos cada vez menores.

Antes de iniciar a execução do processo, é necessário definir a refatoração a ser avaliada e com quais métricas será feita esta avaliação. Tanto a refatoração quanto as métricas podem ser definidas pelo próprio avaliador ou aproveitadas de outros trabalhos. De posse dessas informações, é possível iniciar a avaliação da refatoração usando como critério as métricas escolhidas.

3.1. Processo de avaliação dos passos atômicos

O objetivo deste processo é verificar como o valor das métricas escolhidas é alterado quando forem feitas pequenas alterações no software. Essas pequenas alterações foram chamadas de “passos atômicos” porque devem ser passos tão simples que não permitam sua divisão em passos menores. Eles devem ser fáceis de avaliar e genéricos o suficientes para permitir sua composição para formar alterações mais complexas. Exemplos de passos atômicos são: “criar classe vazia”, “adicionar implementação de interface a componente”, “remover atributo” e “mover código entre operações do mesmo componente”.

O primeiro passo do processo é definir o conjunto de passos atômicos que será avaliado. Deve-se levar em conta quais pequenas alterações são feitas durante a execução da refatoração que está sendo avaliada, pois este mesmo conjunto de passos atômicos será usado na decomposição da refatoração (seção 3.2).

Com os passos atômicos definidos, passa-se à análise dos efeitos de cada um desses passos no valor das métricas adotadas. Por exemplo, para um conjunto de métricas que meça quantas classes e quantas linhas de código existem no sistema, o passo atômico “criar classe vazia” aumenta o número de classes em uma unidade e o número de linhas de código em duas unidades – uma linha com a declaração da classe e outra para o fechamento de seu escopo (representado em linguagens como Java e AspectJ por “}”). Algumas métricas como “número de linhas de código” podem ter resultados diferentes de acordo com a linguagem ou estilo de programação adotados, por isso o avaliador deve levar em conta essas possíveis variações ao fazer a avaliação dos passos atômicos.

Após analisar os efeitos de todos os passos atômicos em cada uma das métricas, obtém-se um conjunto de avaliações atômicas que serão usadas na fase final do processo de avaliação da refatoração (seção 3.3).

3.2. Processo de decomposição da refatoração

O processo de decomposição da refatoração busca dividir continuamente a refatoração original em passos cada vez menores, até obter apenas passos atômicos. Sua primeira etapa, representada na figura 1 pela seta entre a refatoração e o conjunto de alterações mais simples combinadas, corresponde à divisão da refatoração em uma seqüência de passos, o que já é feito previamente pelos autores que usam o formato de descrição de refatorações adotado por Fowler et al. [Fowler et al., 1999]. Um exemplo desta etapa é a descrição da refatoração “Encapsular Campo” [Fowler et al., 1999, p. 206], que divide o processo de encapsulamento de um atributo em 5 passos:

1. Criar métodos de gravação e leitura para o atributo;
2. Encontrar todos os clientes fora da classe que referenciem o atributo. Se o cliente usar o valor, substituir a referência por uma chamada ao método de leitura. Se o cliente alterar o valor, substituir a referência por uma chamada ao método de gravação;
3. Compilar e testar após cada alteração;
4. Assim que todos os clientes forem alterados, declarar o atributo como privado;
5. Compilar e testar.

Depois desta primeira divisão da refatoração em uma seqüência de passos, inicia-se o ciclo de decomposição desses passos. Este ciclo começa verificando se todos os passos são atômicos, tendo como critério o conjunto de passos atômicos definidos no processo descrito na seção 3.1. Em caso positivo, o ciclo se encerra e com ele o processo de decomposição da refatoração. Em caso negativo, é preciso continuar a divisão dos passos em alterações cada vez mais simples: para cada passo não-atômico, deve-se dividi-lo em passos menores que quando combinados sejam equivalentes ao passo original. Feito isso, o ciclo reinicia. Quando não houver mais nenhum passo não-atômico no conjunto, o ciclo se encerra e a decomposição da refatoração terá terminado.

Um caso da evolução deste ciclo pode ser observado através da decomposição dos passos da refatoração “Encapsular Campo” vista anteriormente. Por limitações de espaço, apresentaremos

apenas a decomposição do passo 1. Suponha que foram definidos e avaliados previamente os seguintes passos atômicos:

- Criar método vazio sem parâmetros e com retorno vazio;
- Adicionar parâmetro a método;
- Trocar retorno vazio por tipo de retorno;
- Adicionar linha de atribuição de valor a atributo;
- Adicionar linha de leitura de valor de atributo.

O passo 1 diz que devem ser criados métodos de gravação e leitura para o atributo, o que não é um dos passos atômicos definidos previamente. Dessa forma, este passo deve ser dividido em passos menores. Uma possível divisão seria a seguinte:

1. Criar métodos de gravação e leitura para o atributo =
 - (a) Criar método de gravação de atributo;
 - (b) Criar método de leitura de atributo.

Esses dois passos fazem o mesmo que o passo original, porém ainda não são passos atômicos. Isso torna necessário prosseguir a divisão deles em alterações ainda mais simples, o que pode ser feito da seguinte forma:

1. Criar métodos de gravação e leitura para o atributo =
 - (a) Criar método de gravação de atributo =
 - i. Criar método vazio sem parâmetros e com retorno vazio;
 - ii. Adicionar parâmetro a método;
 - iii. Adicionar linha de atribuição de valor a atributo.
 - (b) Criar método de leitura de atributo =
 - i. Criar método vazio sem parâmetros e com retorno vazio;
 - ii. Trocar retorno vazio por tipo de retorno;
 - iii. Adicionar linha de leitura de valor de atributo.

Após esta divisão, todos os passos combinados pertencem ao conjunto de passos atômicos definido anteriormente, portanto o ciclo de decomposição se encerra.

3.3. Fase final

Após avaliar os passos atômicos e transformar a refatoração original em um conjunto de passos atômicos combinados, o processo principal entra em sua fase final. É nesta fase que as informações obtidas nos dois processos anteriores são combinadas para produzir uma descrição dos efeitos da refatoração nos valores das métricas escolhidas e é feita a análise dos resultados alcançados.

A primeira etapa desta fase é a multiplicação dos efeitos de cada passo atômico de acordo com o número de vezes que o passo foi usado na refatoração. No exemplo de “Encapsular Campo”, para criar os métodos de gravação e leitura o passo atômico “Criar método vazio sem parâmetros e com retorno vazio” é usado duas vezes, enquanto que os demais passos atômicos são usados apenas uma vez. Dessa forma, os efeitos desse passo nos valores das métricas devem ser multiplicados por dois e os efeitos dos demais por um.

Em seguida, os efeitos multiplicados devem ser somados para obter as variações totais dos valores de cada uma das métricas, ou seja, descobre-se quais são os efeitos da refatoração quando todos os seus passos forem executados sequencialmente. De posse desses efeitos, é possível analisar quais as conseqüências da aplicação da refatoração em um programa, como por exemplo que atributos de qualidade serão melhorados, quais os impactos negativos nas métricas adotadas, etc.

4. Exemplo de Aplicação

Nesta seção uma refatoração de software é avaliada através da execução do processo proposto, visando mostrar a aplicabilidade do mesmo. Para facilitar o entendimento deste exemplo, a refatoração escolhida não é muito extensa ou complexa, assim ficam mais claras a execução de cada etapa do processo e as conexões existentes entre elas. Além disso, ao longo da descrição do processo (seção 3) foram dados pequenos exemplos que complementam as explicações desta seção.

4.1. A refatoração avaliada

A refatoração usada neste exemplo foi apresentada por Monteiro e Fernandes [Monteiro and Fernandes, 2005] com o objetivo de ser aplicada em programas orientados a aspectos, mais especificamente nos que são implementados usando a linguagem AspectJ [Kiczales et al., 2001]. Ela foi chamada de “Trocar *implements* por *declare parents*” e, resumidamente, possui a seguinte descrição:

Situação típica: Algumas classes implementam uma interface relacionada a um interesse transversal secundário. A implementação da interface só é usada quando o interesse está presente no sistema.

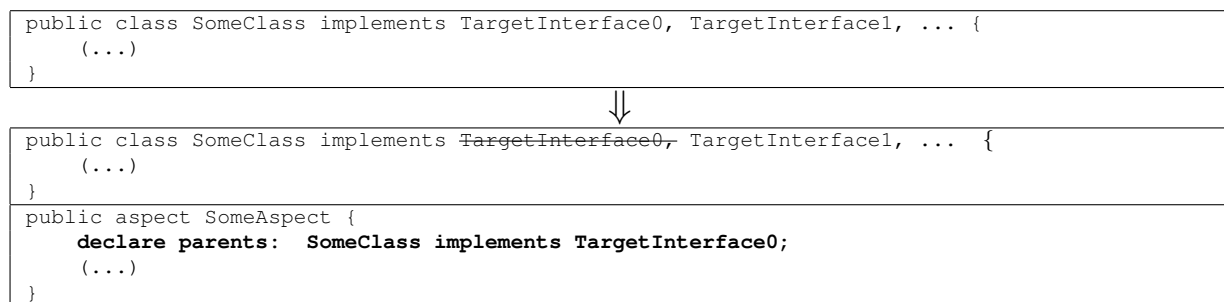
Ação recomendada: Trocar o `implements` da classe por um `declare parents` no aspecto.

Motivação: Interfaces são a forma padrão em Java de representar os vários papéis de uma classe, e AspectJ torna possível representar muitos desses papéis dentro de aspectos.

Mecânica :

1. Criar o `declare parents` adequado no aspecto;
2. Remover a declaração de `implements` referente à interface na classe que a implementa;
3. Compilar e testar.

Exemplo:



4.2. As métricas usadas

Como a refatoração avaliada é uma refatoração orientada a aspectos, as métricas a serem usadas devem permitir a sua aplicação em programas orientados a aspectos. Por causa disso, foram selecionadas as métricas apresentadas por Sant’Anna et al. [Sant’Anna et al., 2003], cujas definições estão descritas resumidamente na tabela 1. Essas métricas avaliam o software analisando quatro atributos diferentes: tamanho, coesão, acoplamento e separação de interesses.

Os primeiros três atributos são medidos por métricas definidas através do reuso e refinamento de métricas orientadas a objetos clássicas [Sant’Anna et al., 2003], estendendo as definições originais para contemplar as novas estruturas da orientação a aspectos. Já a separação de interesses é medida por novas métricas definidas por Sant’Anna et al.

Atributo	Métrica	Definição
Separação de Interesses	Difusão de Interesse por Componentes (CDC)	Conta o número de classes e aspectos cujo propósito principal é contribuir para a implementação de um interesse e o número de outras classes e aspectos que os acessam
	Difusão de Interesse por Operações (CDO)	Conta o número de métodos e adendos cujo propósito principal é contribuir para a implementação de um interesse e o número de outros métodos e adendos que os acessam
	Difusão de Interesse por Linhas de Código (CDLOC)	Conta o número de pontos transição de cada interesse pelas linhas de código. Pontos de transição são pontos no código em que ocorre uma "troca de interesse" de uma linha sombreada – relacionada a um interesse – para uma linha não-sombreada – sem relação com o interesse
Acoplamento	Acoplamento Entre Componentes (CBC)	Conta o número de outras classes e aspectos aos quais uma classe ou aspecto está acoplado
	Profundidade da Árvore de Herança (DIT)	Conta a profundidade na árvore de herança em que uma classe ou aspecto é declarado
Coesão	Falta de Coesão em Operações (LCOO)	Mede a falta de coesão de uma classe ou aspecto em termos da quantidade de pares de métodos e adendos que não acessam a mesma variável de instância
Tamanho	Tamanho do Vocabulário (VS)	Conta o número de classes e aspectos
	Linhas de Código (LOC)	Conta o número de linhas de código, descontadas linhas em branco e comentários
	Número de Atributos (NOA)	Conta o número de atributos de cada classe ou aspecto
	Operações Ponderadas por Componente (WOC)	Conta o número de métodos e adendos de cada classe ou aspecto e o número de seus parâmetros

Tabela 1: Conjunto de Métricas Usadas

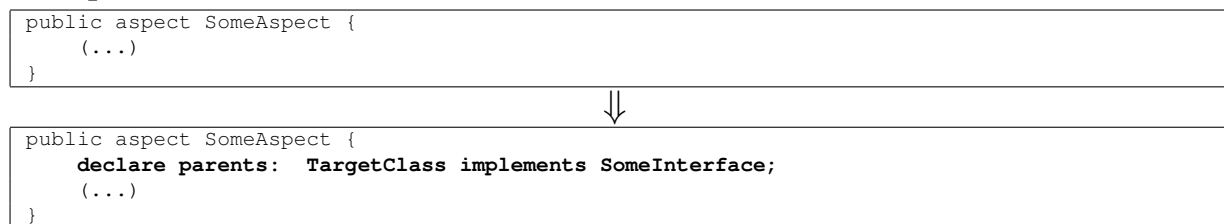
4.3. Avaliação dos passos atômicos

Para avaliar os passos atômicos, primeiro deve-se estabelecer quais são as alterações pontuais necessárias. Como a refatoração que está sendo avaliada é “Trocar *implements* por *declare parents*”, então os seguintes passos atômicos são suficientes:

Passo atômico 1 – Criar declaração intertipo de implementação em aspecto:

Descrição: adicionar a um aspecto uma declaração inter-tipo da espécie *declare parents* que faça com que um componente implemente uma interface do sistema.

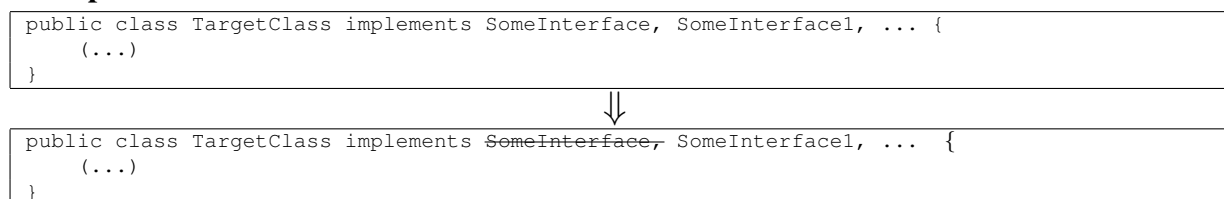
Exemplo:



Passo atômico 2 – Remover implementação de interface de classe:

Descrição: retirar uma interface da lista de interfaces implementadas de um componente.

Exemplo:



Passo atômico 3 – Compilar e testar:

Descrição: compilar o código e rodar os testes existentes.

Em seguida, deve-se verificar como o valor de cada métrica é alterado por cada um desses passos.

Passo atômico 1:

LOC: +1

Uma linha de código é adicionada ao aspecto: a linha com a declaração inter-tipo.

CBC: 0 a +2

A declaração inter-tipo possui acoplamento a dois componentes: a interface implementada (SomeInterface) e a classe-alvo que deve implementá-la (TargetClass). Após adicionar a declaração inter-tipo no aspecto, o número de acoplamentos do aspecto com outros componentes terá:

- aumentado duas unidades, se tanto a interface quanto a classe-alvo não forem previamente acopladas ao aspecto;
- aumentado uma unidade, se apenas a interface **ou** a classe-alvo não for previamente acoplada ao aspecto;
- se mantido igual, se tanto a interface quanto a classe-alvo já forem previamente acopladas ao aspecto.

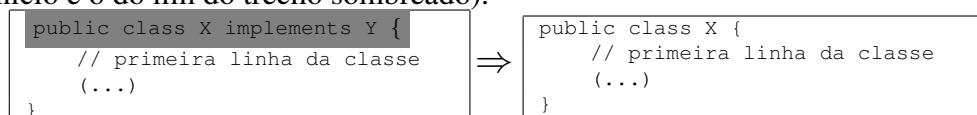
Passo atômico 2:

CDC: -1 a 0

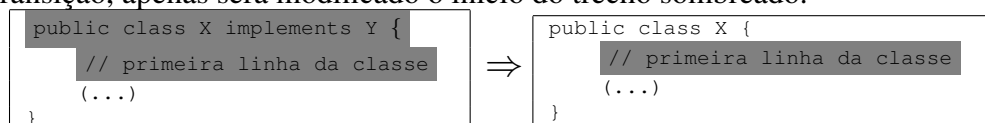
Se a interface estiver relacionada a um interesse transversal e se originalmente a única referência a esse interesse na classe for pela implementação da mesma, então após remover a interface essa referência será eliminada e a classe deixará de estar relacionado ao interesse, diminuindo assim o valor de CDC em uma unidade.

CDLOC: -2 a 0

Para haver alteração no valor de CDLOC, é necessário primeiro que a linha de declaração da classe deixe de ser sombreada. Além disso, também é necessário que imediatamente abaixo da declaração (no início do corpo da classe) não haja uma linha sombreada. Caso as duas condições sejam satisfeitas, então a remoção da interface eliminará dois pontos de transição do código na classe (o ponto do início e o do fim do trecho sombreado).



Caso a linha de declaração deixe de ser sombreada mas a primeira linha do corpo da classe também seja sombreada, não haverá alteração do número de pontos de transição, apenas será modificado o início do trecho sombreado.



CBC: -1 a 0

Se a classe só estiver acoplada à interface através da implementação da mesma (isto é, se no corpo da classe não houver referência à interface), então a remoção desta interface da lista de interfaces implementadas pela classe eliminará o acoplamento existente entre elas, o que diminui o valor de CBC da classe em uma unidade.

Passo atômico 3: Este passo, presente em praticamente todas as descrições de refatoração, apenas indica o momento que as alterações feitas pela refatoração devem ser testadas para verificar a não-alteração do comportamento do software. Ele não modifica o código de nenhuma maneira, portanto seu efeito sobre o conjunto de métricas é nulo.

Após cada um dos passos atômicos ter sido avaliado em função de todas as métricas escolhidas, tem-se as variações que os valores das métricas sofrerão quando os passos forem aplicados, mostradas na tabela 2. Isso quer dizer que se o valor de CBC for inicialmente 10, após remover a implementação de uma interface de uma classe o valor será 9 ou 10; se for originalmente 100, após aplicar esse passo atômico CBC passará a ser 99 ou 100.

Passo atômico	Separação de Interesses			Tamanho				Acoplamento		Coesão
	CDC	CDO	CDLOC	VS	LOC	NOA	WOC	CBC	DIT	LCOO
Criar declaração...	0	0	0	0	+1	0	0	0 a +2	0	0
Remover implementação...	-1 a 0	0	-2 a 0	0	0	0	0	-1 a 0	0	0
Compilar e testar	0	0	0	0	0	0	0	0	0	0

Tabela 2: Efeitos dos passos atômicos

4.4. Decomposição da refatoração

A decomposição de “Trocar *implements* por *declare parents*” se inicia dividindo a ação recomendada (“Trocar o *implements* da classe por um *declare parents* no aspecto”) em uma série de passos sequenciais, o que já foi previamente feito por seus autores na descrição da mecânica da refatoração. Assim, o ciclo de decomposição dos passos começa com o seguinte conjunto de alterações:

1. Criar o *declare parents* adequado no aspecto;
2. Remover a declaração de *implements* referente à interface na classe que a implementa;
3. Compilar e testar.

Agora deve-se verificar se todas essas alterações são atômicas. Ora, para executar o passo 1 basta usar o passo atômico “criar declaração intertipo de implementação em aspecto”; já o passo 2 pode ser feito aplicando o passo atômico “remover implementação de interface de classe”; e o passo 3 é exatamente o passo atômico “compilar e testar”. Desta forma, conclui-se que o conjunto de alterações é composto apenas por passos atômicos e encerra-se o ciclo de decomposição. A tabela 3 mostra o resultado deste processo: o conjunto de passos atômicos utilizados com as respectivas frequências.

4.5. Multiplicação e soma dos resultados

De posse das tabelas 2 e 3, é possível saber quais serão os efeitos totais nos valores das métricas quando os passos atômicos que compõem a refatoração forem executados sequencialmente. Primeiro é feita a multiplicação dos efeitos dos passos de acordo com a frequência de cada um deles descrita na tabela 3. No caso desta refatoração, a multiplicação não modifica nenhum efeito porque todos os passos são usados apenas uma vez. Depois, os efeitos multiplicados são somados, obtendo os resultados presentes na tabela 4. A última linha da tabela mostra o quanto cada métrica terá seu valor alterado ao aplicar a refatoração. É nesta linha que estão os dados a serem analisados para verificar possíveis vantagens e desvantagens de usar a refatoração.

4.6. Análise dos resultados

Na avaliação de exemplo, todas as métricas usadas consideram melhores os softwares cujos valores medidos são mais baixos, portanto diminuições nos valores das métricas significam benefícios trazidos pela refatoração. No caso de “Trocar *implements* por *declare parents*”, observa-se a possibilidade de melhorias na separação de interesses, pois tanto CDC quanto CDLOC podem diminuir ou não se alterar. No entanto, essas melhorias tem um custo: o aumento do tamanho do software através do incremento do número de linhas de código. Já o acoplamento dos componentes pode melhorar ou piorar, dependendo das características da classe e do aspecto envolvidos na refatoração. E, por fim, a coesão dos componentes não é modificada pela refatoração.

Com alterações tão diferentes para cada atributo do software, a decisão de usar ou não a refatoração depende da importância de cada atributo para o desenvolvedor. Por exemplo, se uma melhor separação de interesses for mais importante que um pequeno aumento no tamanho do software, a refatoração deve ser aplicada; se no entanto não for aceito que uma das métricas piore seus resultados, então esta refatoração deve ser evitada.

5. Discussões

O processo proposto neste trabalho permite avaliar refatorações de código através do uso de métricas de software. Com ele, é possível obter evidências quantitativas que comprovem

Passo atômico	Frequência
Criar declaração...	1
Remover implementação...	1
Compilar e testar	1

Tabela 3: Frequência de uso dos passos atômicos em “Trocar *implements* por *declare parents*”

Passo atômico	Separação de Interesses			Tamanho				Acoplamento		Coesão
	CDC	CDO	CDLOC	VS	LOC	NOA	WOC	CBC	DIT	LCOO
Criar declaração...	0	0	0	0	+1	0	0	0 a +2	0	0
Remover implementação...	-1 a 0	0	-2 a 0	0	0	0	0	-1 a 0	0	0
Compilar e testar	0	0	0	0	0	0	0	0	0	0
Total	-1 a 0	0	-2 a 0	0	+1	0	0	-1 a +2	0	0

Tabela 4: Efeitos totais de “Trocar *implements* por *declare parents*”

a existência de benefícios da refatoração levantados por avaliações qualitativas. Ele também permite conhecer os efeitos colaterais da refatoração, isto é, os impactos da mesma em atributos de software não previstos originalmente pela refatoração por não estarem relacionados com seu objetivo principal.

Saber quais são os efeitos de uma refatoração em atributos de software antes de aplicá-la permite ao desenvolvedor avaliar se é vantajoso usá-la e decidir por sua aplicação ou não. Isso é particularmente importante para as refatorações que não têm refatorações opostas definidas, pois caso elas sejam aplicadas e alterarem negativamente algum atributo de qualidade relevante, desfazer as modificações será muito mais complexo e propenso à inserção de erros no programa.

O processo de avaliação permite reaproveitar resultados à medida que novas avaliações forem feitas. A divisão do processo de avaliação em dois processos menores possibilita o reaproveitamento de resultados parciais obtidos em aplicações anteriores do processo. Por exemplo, para reavaliar uma refatoração usando um novo conjunto de métricas, reaproveita-se a definição dos passos atômicos e todo o processo de simplificação da refatoração, sendo necessário apenas obter os efeitos dos passos atômicos nas novas métricas e fazer a multiplicação e soma desses novos resultados. No caso inverso, em que as métricas são mantidas mas a refatoração avaliada é trocada, o reaproveitamento existe caso algum dos passos atômicos necessários já tenha sido avaliado anteriormente, pois seus efeitos já terão sido levantados e não será necessário reavaliá-lo.

Outra forma de reaproveitamento, desta vez dos resultados finais, ocorre na avaliação de refatorações complexas que usem refatorações mais simples em seus passos. Uma vez avaliada uma refatoração, ela passa a ser considerada um passo atômico. Dessa forma, toda vez que ela for utilizada por outra refatoração mais complexa seus resultados serão reaproveitados e não será necessário repetir sua divisão em passos atômicos ou o cálculo de seus efeitos.

O processo, no entanto, também possui algumas limitações. Em primeiro lugar, ele só pode ser usado com as chamadas “métricas estáticas” [Sommerville, 2007]. Isso ocorre porque as métricas dinâmicas, coletadas a partir de medições de programas em execução [Sommerville, 2007], não podem ser medidas analisando apenas alterações do código-fonte, o que torna impossível obter o impacto de passos atômicos nesse tipo de métrica. Esta limitação também está presente em outras propostas para obter os impactos de refatorações no código do software [Bois and Mens, 2003, Tahvildari and Kontogiannis, 2004], e não permite que o processo seja usado para medir características como desempenho ou confiabilidade do software.

Outros estudos do grupo têm mostrado que a forma escolhida para decompor a refatoração pode dificultar a avaliação de refatorações muito complexas. Isto ocorre porque esse tipo de refatoração precisa de muitos passos atômicos combinados de formas diferentes, transformando alterações relativamente simples (como “remover classe”) em uma seqüência de passos muito extensa (“remover atributo de classe”, “remover parâmetro de operação”, “trocar tipo de retorno por retorno vazio”, “remover implementação de interface de classe”, “remover hierarquia”, etc.). No entanto, por causa da possibilidade de reaproveitamento de resultados obtidos em avaliações anteriores, à medida que novas avaliações forem feitas a decomposição de refatorações se tornará mais simples e serão necessários menos esforços para completar a avaliação.

6. Trabalhos Relacionados

Diversos trabalhos apresentam refatorações para diferentes paradigmas de programação, como orientado a objetos [Fowler et al., 1999, Kerievsky, 2004], imperativo [Garrido and Johnson, 2002] e orientado a aspectos [Monteiro and Fernandes, 2005, Iwamoto and Zhao, 2003]. Estes trabalhos têm uma maior preocupação em descrever as etapas das refatorações apresentadas, não sendo feita em nenhum deles uma avaliação quantitativa que demonstre os impactos das refatorações em atributos de qualidade de software.

Alguns trabalhos apresentam diferentes formas de avaliar refatorações. Du Bois e Mens [Bois and Mens, 2003] propõem um formalismo para descrever o impacto de refatorações na estrutura de programas. No entanto, a forma escolhida pelos autores para representar a estrutura dos programas restringe a aplicabilidade de sua proposta a refatorações e métricas orientadas a objetos, enquanto o processo de avaliação proposto neste artigo pode ser usado com diferentes paradigmas de programação.

Tahvildari e Kontogiannis [Tahvildari and Kontogiannis, 2004] avaliam refatorações através da análise de meta-padrões de transformações de código. Esta proposta também está restrita a refatorações e métricas orientadas a objetos, além de só contemplar transformações que podem ser descritas pelos meta-padrões de manutenção de software definidos pelos autores.

Benn et al. [Benn et al., 2005] avaliam algumas refatorações para extrair aspectos de código orientado a objetos através de um estudo de caso. Após fazer medições antes e depois de aplicar as refatorações, e concluir que as transformações trouxeram conseqüências positivas para o caso escolhido, os autores afirmam ser necessário um estudo mais geral para determinar se as melhorias do caso específico também seriam alcançadas em outras situações semelhantes.

Figueiredo et al. [Figueiredo et al., 2005] propõem um método de avaliação quantitativa de artefatos a ser aplicado durante o desenvolvimento de software orientado a aspectos. Este método prevê uma etapa de refatorações do código para melhorar atributos de qualidade identificados como deficientes em medições feitas em etapas anteriores. Para escolher que refatorações aplicar, podem ser usados como critério os impactos que as refatorações têm nos atributos de qualidade medidos, impactos esses que são obtidos usando o processo de avaliação de refatorações definido no presente trabalho. Desta forma, o trabalho de Figueiredo et al. é considerado complementar a este.

7. Conclusão

Este trabalho propõe um processo para avaliar refatorações de software através do uso de métricas, de forma a conhecer os impactos que as alterações terão em diferentes atributos de qualidade. Com ele é possível obter evidências quantitativas dos benefícios trazidos pelas refatorações, bem como descobrir possíveis desvantagens de seu uso não previstas em sua definição.

O processo pode ser usado para avaliar refatorações para diferentes paradigmas de programação, e permite reaproveitar resultados alcançados em usos anteriores do processo, diminuindo gradativamente o esforço necessário para completar as avaliações. Os resultados obtidos com as avaliações auxiliam no desenvolvimento de software porque fornecem informações sobre os efeitos positivos e negativos da aplicação das refatorações avaliadas e permitem ao desenvolvedor prever as conseqüências de sua aplicação no código antes alterá-lo, evitando o trabalho de desfazê-la e possibilitando a escolha da refatoração que terá impactos

mais positivos.

Referências

- [Benn et al., 2005] Benn, J., Constantinides, C., Padda, H. K., Pedersen, K. H., Rioux, F., and Ye, X. (2005). Reasoning on software quality improvement with aspect-oriented refactoring: A case study. In Tsai, W. and Hamza, M., editors, Proceedings of the 2005 Software Engineering and Applications, Phoenix, AZ, USA.
- [Bois and Mens, 2003] Bois, B. D. and Mens, T. (2003). Describing the impact of refactoring on internal program quality. In Proceedings of the International Workshop on Evolution of Large-scale Industrial Software Applications, pages 37–48, Vrije Universiteit Brussel.
- [Figueiredo et al., 2005] Figueiredo, E., Garcia, A., Sant’Anna, C., Kulesza, U., and Lucena, C. (2005). Assessing aspect-oriented artifacts: Towards a tool-supported quantitative method. In Proceedings of the Workshop on Quantitative Approaches in OO Software Engineering (QAOOSE, held with ECOOP 2005), Glasgow, Scotland.
- [Fowler et al., 1999] Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (1999). Refactoring: Improving the Design of Existing Code. Addison Wesley.
- [Garrido and Johnson, 2002] Garrido, A. and Johnson, R. (2002). Challenges of refactoring c programs. In IWPSE ’02: Proceedings of the International Workshop on Principles of Software Evolution, pages 6–14, New York, NY, USA. ACM Press.
- [Iwamoto and Zhao, 2003] Iwamoto, M. and Zhao, J. (2003). Refactoring aspect-oriented programs. In Akkawi, F., Aldawud, O., Booch, G., Clarke, S., Gray, J., Harrison, B., Kandé, M., Stein, D., Tarr, P., and Zakaria, A., editors, Proceedings of The 4th AOSD Modeling With UML Workshop.
- [Kerievsky, 2004] Kerievsky, J. (2004). Refactoring to Patterns. Pearson Higher Education.
- [Kiczales et al., 2001] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. G. (2001). An overview of aspectj. In Proceedings of the 15th European Conference on Object-Oriented Programming, pages 327–353, London, UK. Springer-Verlag.
- [Mens and Tourwé, 2004] Mens, T. and Tourwé, T. (2004). A survey of software refactoring. IEEE Trans. Softw. Eng., 30(2):126–139.
- [Monteiro, 2005] Monteiro, M. P. (2005). Refactorings to Evolve Object-Oriented Systems with Aspect-Oriented Concepts. PhD thesis, Universidade do Minho.
- [Monteiro and Fernandes, 2005] Monteiro, M. P. and Fernandes, J. M. (2005). Towards a catalog of aspect-oriented refactorings. In Proceedings of the 4th international conference on Aspect-oriented software development (AOSD’05), pages 111–122, New York, NY, USA. ACM Press.
- [Opdyke, 1992] Opdyke, W. F. (1992). Refactoring: A Program Restructuring Aid in Designing ObjectOriented Application Frameworks. PhD thesis, University of Illinois, Urbana-Champaign.
- [Sant’Anna et al., 2003] Sant’Anna, C., Garcia, A., Chavez, C., Lucena, C., and von Staa, A. (2003). On the reuse and maintenance of aspect-oriented software: An assessment framework. In Proceedings of the Brazilian Symposium on Software Engineering, pages 19–34, Manaus.
- [Sommerville, 2007] Sommerville, I. (2007). Software engineering. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 8th edition.

[Tahvildari and Kontogiannis, 2004] Tahvildari, L. and Kontogiannis, K. (2004). Improving design quality using meta-pattern transformations: a metric-based approach. Journal of Software Maintenance and Evolution: Research and Practice, 16(4-5):331–361.