Relacionando *Refactorings* e Métricas de Código Fonte - Um Primeiro Passo para Detecção Automática de Oportunidades de Refactoring

Glauco de Figueiredo Carneiro[1,2] e Manoel Gomes de Mendonça Neto[1]
Núcleo de Pesquisa em Redes de Computadores – Universidade Salvador[1]
E-mail: {glauco.carneiro, mgmn}@unifacs.br

Empresa Brasileira de Correios e Telégrafos[2]
Diretoria Regional da Bahia

Resumo

Refactoring – melhorar a estrutura interna do software sem modificação no seu comportamento observável – é um mecanismo importante para se evitar a degradação da qualidade do software. Fundamental para tal finalidade é a identificação de trechos do código fonte que apresentam oportunidades de *refactoring* – comumente chamados de *bad smells*. Este artigo propõe uma abordagem para auxiliar na detecção de *bad smells* através de medição de código fonte e seu foco é a apresentação de um primeiro passo no sentido implementar esta abordagem. Para tal finalidade foi realizado um estudo que relaciona métricas, *refactorings* e *bad smells*. O estudo é dividido em duas partes. A primeira parte – *top-down* – é baseada na aplicação analítica do método Meta Pergunta Métrica (MPM ou GQM, em inglês) na definição de métricas para detecção de *bad smells*. A segunda parte – *bottom-up* – é um estudo empírico do relacionamento entre métricas conhecidas de código fonte, *refactorings* e *bad smells*.

Palavras chave: *refactoring*, métricas de software, método GQM.

Abstract

Software refactoring - improving the internal structure of the software without changing its observable behavior - is an important action towards avoiding software quality decay. Key to this activity is the identification of portions of the source code that offers opportunities for refactoring - the so called "code bad smells". This work presents an approach to help on the detection of code bad smells through source code measurement on focuses on a first step towards the implementation of this approach. It presents a study relating metrics, refactorings, and bad smells. The study is broken into two parts. The first - top-down - part is based on the analytical application of the Goal-Question-Metric (GQM) method to define metrics to detect code bad smells. The second - bottom-up - part is an empirical study on the relationship between well-known source code metrics, refactorings and code bad smells.

Keywords: refactoring, software metrics, GQM method, empirical study.

1. Introdução

Aplicações usando linguagens orientadas a objeto podem ser reestruturadas a partir de *refactorings* [15][16]. Operações de *refactoring* reorganizam a hierarquia de classes e redistribuem as variáveis de instância e métodos. De acordo com Fowler [11], o objetivo do *refactoring* é tornar o software mais fácil de ser compreendido e modificado, reestruturando implementações existentes para torná-las mais flexíveis, dinâmicas e reutilizáveis. Este artigo

apresenta um estudo com o objetivo de introduzir métricas no uso de *refactorings*. Esta é uma etapa essencial para o estabelecimento de uma metodologia para auxílio na detecção de oportunidades de *refactoring* usando métricas [6]. São estabelecidos relacionamentos entre alguns dos principais tipos de *refactorings* propostos por Fowler [11] e métricas obtidas através do código fonte do software. O uso de métricas tem potencial para auxílio na execução do processo de *refactoring* e torna seus resultados analisáveis quantitativamente. Como resultado, tem-se um processo de *refactoring* mais previsível e menos dependente de heurísticas pessoais. As Seções seguintes estão organizadas da seguinte forma: na Seção 2 são apresentados conceitos relacionados a *refactoring* para mostrar o contexto no qual foi desenvolvido o trabalho; na Seção 3 é apresentada metodologia para tornar mensuráveis os indicadores de necessidade de *refactoring*; na Seção 4 são apresentados os resultados de dois estudos de caso para estabelecer relacionamento entre *refactorings*, *bad smells* - termo usado por Fowler para descrever oportunidade de *refactoring* - e métricas e, por último, são apresentadas conclusões do trabalho realizado.

2. Contexto do Trabalho

O conceito de *refactoring* foi originalmente definido por Opdyke [15] como uma transformação de programa que preserva o seu comportamento e redefinido por Roberts [16] como uma transformação do programa onde uma pré-condição particular fosse satisfeita. *Refactoring* tem os seguintes objetivos: melhorar o projeto do software, tornar o código mais facilmente compreensível, auxiliar na resolução de possíveis problemas, tornar mais dinâmica a evolução do software e, conseqüentemente, reduzir os custos a ela associados. Proposto e avaliado a partir de 1990, *refactoring* tem proporcionado um desenvolvimento crescente de práticas de desenvolvimento de *software* [11]. Como exemplo, pode-se citar que um dos princípios básicos do *Extreme Programming* [4] é a realização de *refactoring* de forma contínua como parte fundamental do processo de desenvolvimento de software.

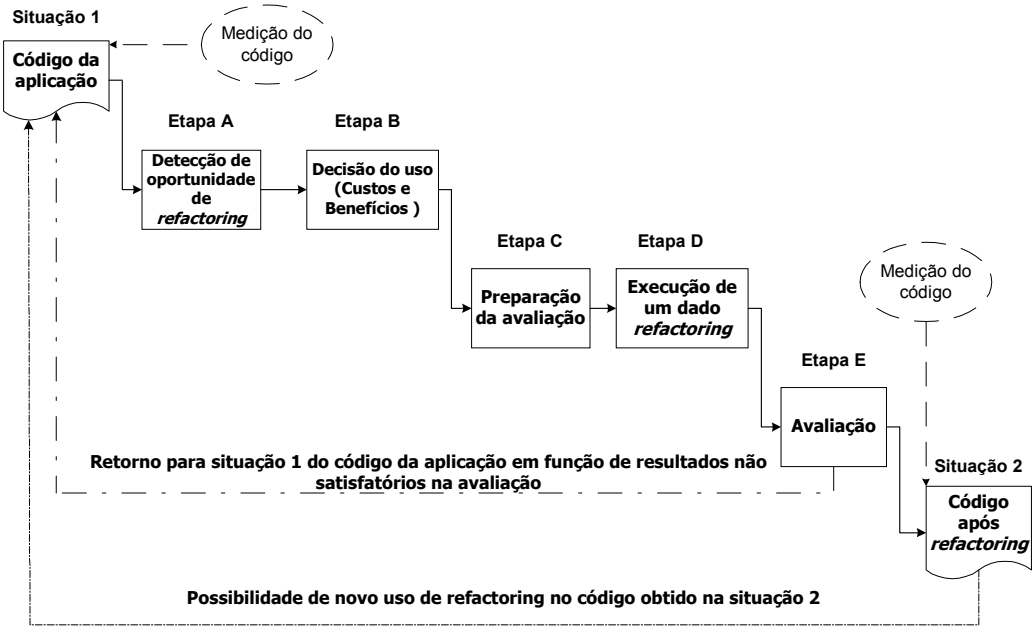

Figura 1: Processo de aplicação de *refactoring*

2.1. Processo de Refactoring

Conforme mostrado na Figura 1, processo de *refactoring* pode ser representado pelas seguintes etapas:

a) **Etapa A - Detecção**: identificar trechos de código com oportunidades de *refactoring*.
b) **Etapa B - Análise da relação entre custos e benefícios**: qual a motivação e os benefícios decorrentes da aplicação do *refactoring*.
c) **Etapa C - Preparação da avaliação**: deve-se preparar o código para realização da avaliação antes da aplicação de *refactoring*.
d) **Etapa D - Execução**: procedimentos usados para aplicar o *refactoring*.
e) **Etapa E - Avaliação**: uso de mecanismos para verificar a preservação de comportamento do programa.

A metodologia proposta neste artigo almeja apenas a detecção de oportunidades de aplicação de *refactorings* - a etapa A da Figura 1. Na realização dos estudos de casos cujos resultados são apresentados na Seção 4, partiu-se do princípio que todos os *refactorings*, aplicados de forma isolada ou seqüencial, obedecem às caraterísticas de manutenção do comportamento observável, conforme estabelecido por Opdyke [15].

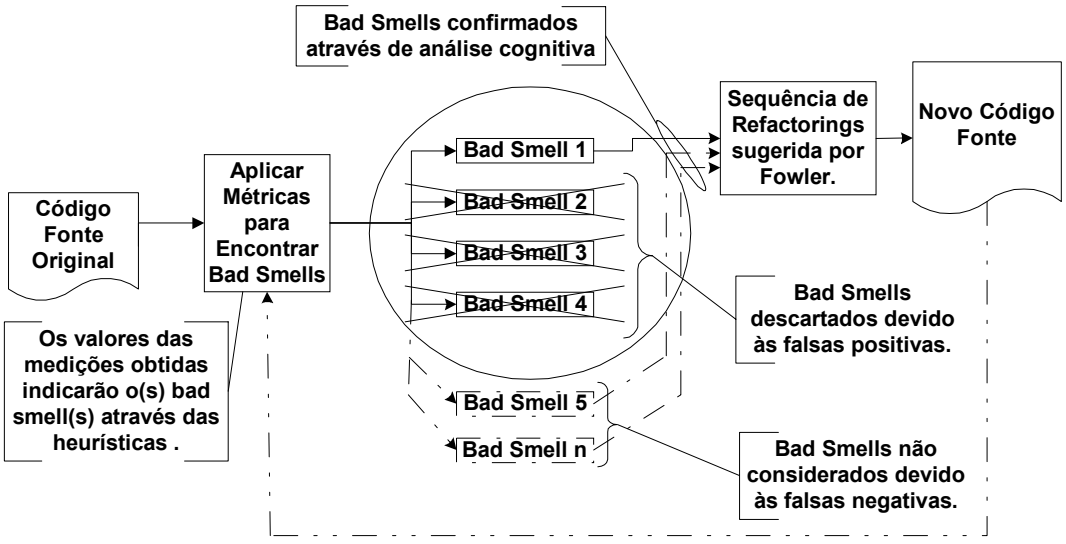

Figura 2: Aplicação de refactoring usando a metodologia proposta

2.2. Metodologia Proposta

A Figura 2 apresenta a metodologia proposta [6][7]. Ela parte do princípio que podem ser estabelecidos conjuntos de métricas para identificar a oportunidade de *refactoring* em uma aplicação orientada a objeto a qual se tenha acesso ao código fonte. A metodologia consiste em:

a) Aplicar a medição ao código fonte da aplicação.
b) Associar os resultados obtidos com os *bad smells*.
c) Efetuar análise cognitiva para verificar a necessidade de aplicação de um ou mais *refactorings* ou a ocorrência de falsas positivas ou negativas.
d) Aplicar um ou mais *refactorings* associados aos *bad smells* como indicado por Fowler [11].

A metodologia proposta indica, portanto, como associar as medições obtidas com os *bad smells*. Sua grande vantagem é reduzir o volume de código fonte a ser analisado pela equipe de *refactoring*.

É importante ressaltar que o uso de métricas cria dois tipos de riscos: a indicação de falsas positivas e as falsas negativas como apresentadas na Figura 2. No caso das falsas positivas, as métricas conduzirão a falsas suspeitas de *bad smells* que deverão ser descartadas através de análise cognitiva. Já as falsas negativas são *bad smells* não detectados pelas métricas, mas que de fato existem e que poderiam ser detectados através de análise cognitiva. Neste trabalho, quando considerada a possibilidade de redução do código fonte a ser analisado, parte-se da premissa que as vantagens do uso de métricas superam os custos associados às falsas positivas e falsas negativas [8].

2.3. Abordagens Similares

Como resultado do levantamento bibliográfico realizado, verificou-se as seguintes abordagens de detecção de oportunidades de *refactoring*: a tradicional detecção baseada em análise cognitiva sobre código fonte [11] e UML [1]; detecção baseada em meta-programação declarativa [18]; detecção baseada em invariantes [9][13] e detecção baseada em métricas [8]. Apenas a última tem similaridades com a abordagem apresentada neste artigo, pois propõe heurísticas para a identificação de *refactorings* já executados entre diferentes versões de uma dada aplicação [8], mas limita-se a estudar cinco *refactorings*: Migração de Trecho de Código para uma Nova Superclasse, Migração de Trecho de Código para uma Nova Subclasse, Mesclagem com Superclasse, Mesclagem com Subclasse e Migração de Trecho de Código para outra Classe. Sendo que destes cinco *refactorings*, os três últimos não constam no catálogo apresentado por Fowler [11]. A abordagem apresentada neste artigo é mais abrangente, possibilita ampliar o escopo da análise para outros *refactorings* e relacioná-los com *bad smells*.

É importante ressaltar que apesar do objetivo final deste trabalho ser a implementação da metodologia proposta na Figura 2, este artigo tem como foco um problema menor: o estudo do relacionamento entre métricas, *refactorings* e *bad smells* usando as abordagens descritas nas Seções 3.1 e 3.2. Desta forma, apresenta-se um primeiro passo no sentido de criar e validar a metodologia apresentada na Figura 2.

3. Relacionando Métricas, *Refactorings* e *Bad Smells*

Este artigo busca estabelecer relacionamentos entre métricas, *refactorings* e *bad smells* usando as seguintes abordagens:

a) Uma abordagem *top-down* usando o método Meta Pergunta Métrica - MPM [1].

b) Uma abordagem *bottom-up* usando análise empírica.

A primeira abordagem almeja identificar analiticamente métricas adequadas à avaliação de *bad smells*. A segunda abordagem é empírica e utiliza mensuração de um grande conjunto de métricas para verificar o seu relacionamento com *refactorings* e *bad smells*. Estas abordagens estão coerentes com a metodologia proposta por Mendonça e Basili para criar e melhorar arcabouços de medição em engenharia de software [14].

3.1. Relacionando Métricas e Refactoring na Abordagem Top-Down

Esta abordagem tem o objetivo de indicar métricas adequadas à detecção dos *bad smells* propostos por Fowler [11]. Conforme apresenta a Figura 3, a metodologia proposta usa o catálogo de *bad smells* apresentado por Fowler como ponto de partida para definição de metas e, em seguida, busca identificar as perguntas e as métricas mais adequadas a estas

metas. O modelo aplicado mapeia *bad smells* para métricas conhecidas ou métricas novas ainda não difundidas na literatura.

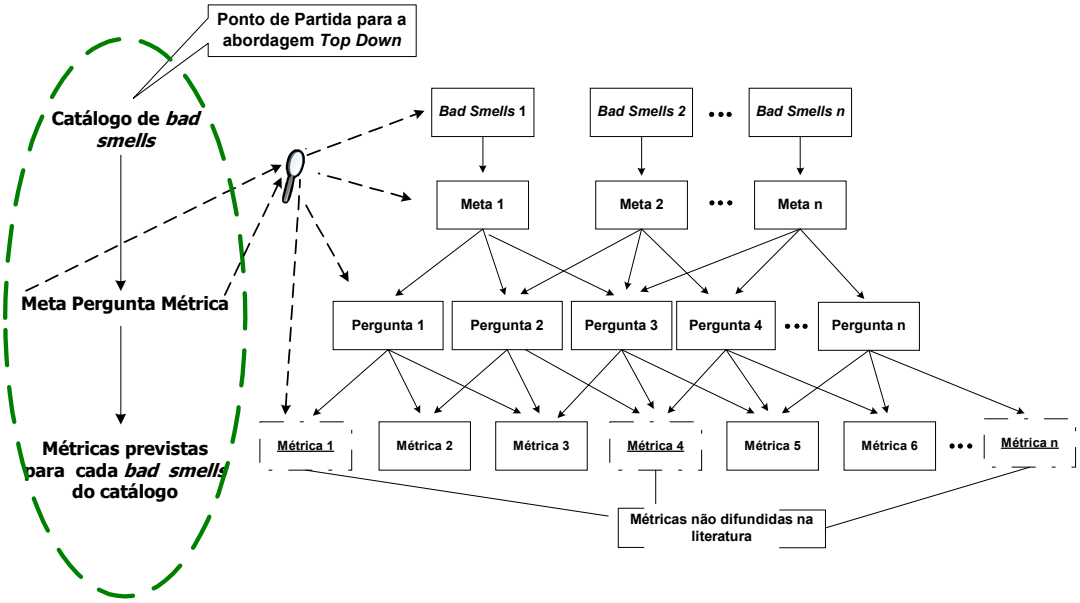

Figura 3: Abordagem *top-down*

Na Seção 4.1.1 é apresentado um exemplo da aplicação da abordagem *top-down* conforme descrito na Figura 3, enquanto que na Seção 4.1.2 é feita uma análise do resultado da aplicação da abordagem em questão para 16 dos 22 *bad smells* sugeridos por Fowler [11].

3.2. Relacionando Métricas e Refactoring na Abordagem Bottom-Up

Para complementar a abordagem *top-down* proposta, tem-se a abordagem *bottom-up* baseada em análise empírica. Na abordagem *bottom-up* considera-se um grande universo de métricas, sendo verificadas quais dessas métricas foram alteradas em função dos *refactorings* aplicados, assim como os *bad smells* correspondentes. Esta abordagem, apresentada na Figura 4, tem os seguintes objetivos:

a) Verificar se o que foi previsto na abordagem *top-down* ocorreu, isto é, se os valores das métricas derivadas realmente sofreram variações representativas.
b) Verificar ocorrências não previstas: métricas que variaram de forma inesperada.
c) Identificar quais as métricas foram previstas e não ocorreram.
d) Fornecer dados concretos para a expansão do número de heurísticas propostas por Demeyer em [8].

A análise da variação das métricas usa a matriz de *refactorings* [7] para que seja fornecido um painel informativo de medições ao longo da seqüência de *refactorings* e entre os *bad smells*. Esta matriz possibilita registrar o histórico dos *bad smells* detectados, assim como os respectivos *refactorings* usados para eliminá-los.

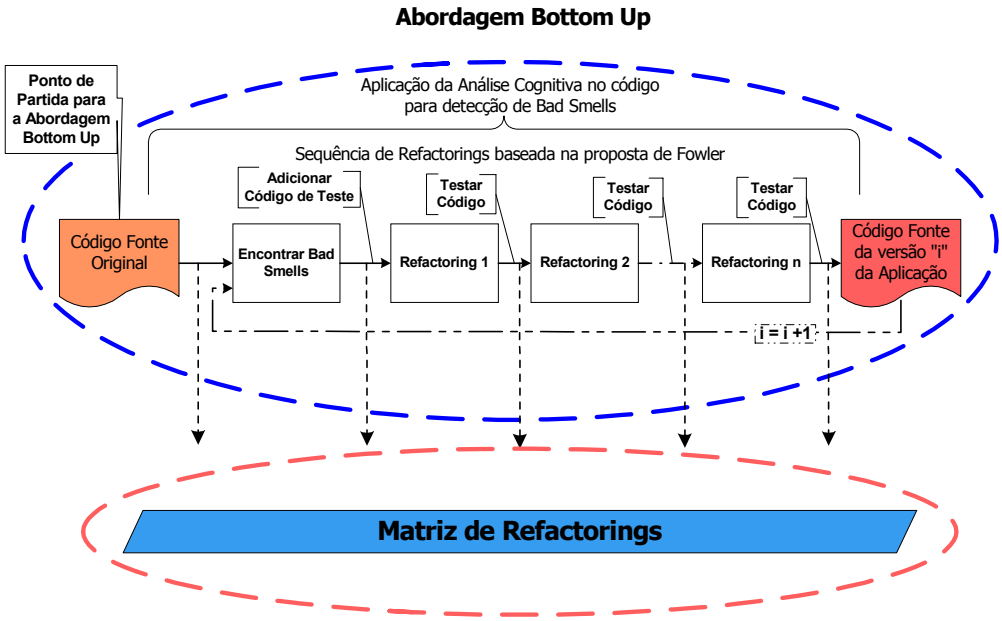

Figura 4: Abordagem *bottom-up*

3.3. Relacionamento entre as Abordagens Top Down e Bottom-Up

A abordagem *top-down* parte de cada *bad smell* para um conjunto de métricas obtido através do método Meta Pergunta Métrica. Este conjunto de métricas tem o objetivo de indicar a presença de um determinado *bad smell*. A principal vantagem desta abordagem é a possibilidade de identificação das métricas indicadas não difundidas ou não disponíveis para auxílio no uso de *refactoring*, fato que a torna independente do conjunto de métricas que será trabalhado na abordagem *bottom-up*.

Na abordagem *bottom-up* faz-se a verificação dos *bad smells* e *refactorings*. O uso da matriz de *refactorings* contendo as medições coletadas possibilita que se verifique se as métricas indicadas na abordagem *top-down* estão coerentes com os *bad smells* e os *refactorings* apresentados por Fowler.

As abordagens são complementares. Conforme ilustrado na Figura 5, métricas ainda não difundidas podem ser definidas, implementadas, validadas e usadas posteriormente na abordagem *bottom-up*. Por outro lado, a abordagem *bottom-up* verifica empiricamente a utilidade das métricas indicadas pela abordagem *top-down*.

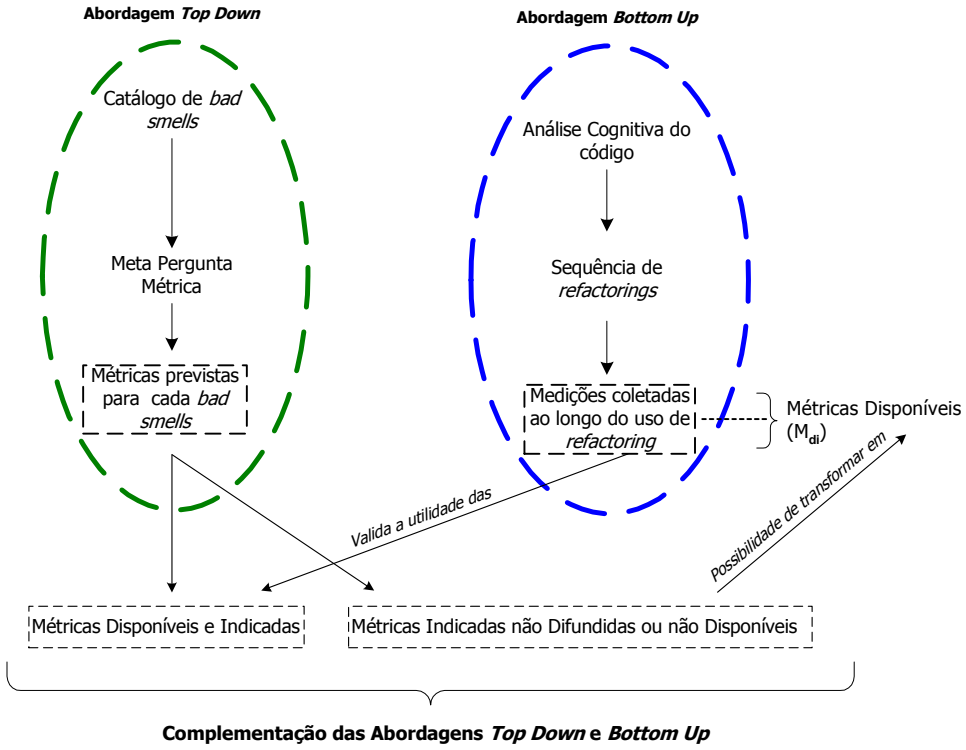

Figura 5: Relacionamento entre as abordagens Top Down e Bottom Up

4. Estudo de Caso

As abordagens propostas nas Seções 3.1 e 3.2 foram avaliadas através de estudos de caso da sua aplicação. O primeiro estudo de caso baseia-se na abordagem *top-down* para indicar/derivar métricas adequadas a 16 dos 22 *bad smells* propostos por Fowler [11]. O segundo estudo de caso usa a abordagem *bottom-up* para investigar o relacionamento entre métricas, *bad smells* e *refactorings* durante a melhoria de uma aplicação. A aplicação usada foi sugerida pelo próprio Fowler [12] após uma consulta via correio eletrônico.

4.1. Estudo de Caso usando a Abordagem Top-Down

Conforme descrito na Seção 3.1, o catálogo de *bad smells* é o ponto de partida para a definição de metas de medição. Cada meta é então mapeada para um grupo de perguntas e um conjunto de métricas relevantes às mesmas. Este conjunto pode ser composto por diferentes tipos de métricas conforme descrito a seguir:

- Tipo 1: Métricas simples e disponíveis.
- Tipo 2: Métricas deriváveis a partir de métricas simples disponíveis.
- Tipo 3: Métricas não disponíveis ou não difundidas, mas que podem ser obtidas através de análise estática do código fonte.
- Tipo 4: Métricas deriváveis a partir de métricas não disponíveis ou não difundidas do Tipo 3 e Tipo 6.
- Tipo 5: Métricas não disponíveis ou não difundidas, cujo resultado depende fortemente de análise cognitiva.
- Tipo 6: Métricas não disponíveis ou não difundidas, que podem ser obtidas através de análise dinâmica da execução do código fonte.

4.1.1. Exemplo de Identificação de Métricas para um *Bad Smell*

Esta subseção apresenta como exemplo o resultado da aplicação da abordagem *top-down* para o *bad smell* Código Duplicado. Código Duplicado é considerado o mais comum dos *bad smells*. Deve-se sempre encontrar uma forma de eliminar a repetição do código em uma aplicação.

Os *refactorings* normalmente aplicados para eliminar este tipo de *bad smell* são: *Migração de Trecho de Código para um Novo Método* (usado para composição de métodos), *Migração de Trecho de Código para uma Nova Classe* (usado para mover funcionalidades entre objetos), *Migração de Método para Superclasse* e *Migração de Trecho de Código para um Novo Método Modelo* (usados para trabalhar com generalizações). As ocorrências mais comuns deste *bad smell* estão descritas na Tabela 1. A aplicação da abordagem MPM inicia no mapeamento do *bad smell* para uma Meta de Mensuração seguindo o padrão proposto por Basili e Rombach [2]. Neste padrão, uma meta é descrita por quatro facetas: objeto de análise, finalidade, foco, e ponto de vista. A seguinte meta foi derivada para o *bad smell* Código Duplicado:

Analisar o código fonte com o objetivo de melhorá-lo (refactoring) com respeito a Código Duplicado do ponto de vista do programador e da equipe de manutenção.

As perguntas derivadas a partir desta meta e as métricas necessárias para respondê-las são então colocadas em uma tabela. A Tabela 2 mostra as perguntas e métricas para o *bad smell* Código Duplicado. A tabela também identifica o tipo das métricas de acordo com a classificação mostrada no início desta Seção.

Tabela 1: Possíveis ocorrências do *bad smell* Código Duplicado e sugestões de *refactorings*

Possibilidades mais prováveis de ocorrência do *Bad Smell*	**Sugestão de correção do *Bad Smell* através de *refactoring***
A mesma expressão existe em dois métodos da mesma classe.	*Migração de Trecho de Código para um Novo Método*
A mesma expressão existe em duas suclasses da mesma classe.	*Migração de Trecho de Código para um Novo Método* e *Migração de Variável para Superclasse*, respectivamente.
Duas ou mais ocorrências de código similar	*Migração de Trecho de Código para um Novo Método* e *Migração de Trecho de Código para um Novo Método Modelo*, respectivamente.
Os métodos possuem o mesmo objetivo com algoritimos diferentes.	Escolher o melhor dos dois algoritimos e depois usar o *refactoring Substituição de Algoritmo*.
O código duplicado ocorre em duas classes não relacionadas.	Aplicar *Migração de Trecho de Código para uma Nova Classe* em uma das classes e depois usar o novo componente na outra classe.

Tabela 2. Perguntas e Metas derivadas para o *bad smell* Código Duplicado

Pergunta	**Métricas**
1) A mesma expressão existe em dois métodos da mesma classe?	**M1.1:** Podem ser obtidas através de análise estática qualitativa. Não difundidas na literatura (Tipo3).
2) Há ocorrências do mesmo código em duas suclasses da mesma classe?	**M1.2:** Podem ser obtidas através de análise estática qualitativa. Não difundidas na literatura (Tipo3).
3) O código é similar, mas não é o mesmo?	**M1.3:** Resultado fortemente dependente de análise cognitiva. Não difundidas na literatura (Tipo5).
4) Os métodos têm o mesmo objetivo com algoritmos diferentes?	**M1.4:** Através de análise estática qualitativa podem ser obtidas métricas auxiliares no processo de obtenção do resultado desejado. Não difundidas na literatura (Tipo3).

4.1.2. Análise dos Resultados

O estudo de caso da abordagem *top-down* para 16 dos *bad smells* contidos no catálogo apresentado por Fowler [11] teve como resultado 36 conjuntos de métricas. Essas métricas estão distribuídas conforme apresentado na Tabela 3.

Tabela 3: Distribuição dos conjuntos de métricas obtidas na abordagem *Top-Down*

Tipo de Métricas	Conjuntos de métricas obtidos da abordagem *Top Down*	Quantidade de conjuntos	Quantidade de conjuntos (%)
1	M2.3, M2.4, ,M3.2, M3.3, M3.4, M3.5, M7.1	7	19,44
2	M2.1, M3.1	2	5,56
3	M1.1, M1.2, M1.4, M4.1, M4.2, ,M4.3, M9.2, M10.1, M11.2, M12.1, M13.1, M13.3	12	33,33
4	M5.2, M5.3, M6.1, M7.2	4	11,11
5	M1.3, M2.2, M5.1, ,M8.1, M9.1, M9.3	6	16,67
6	M11.1, M12.1, M13.2, M14.1, M15.1	5	13,89
Total		**36**	**100**

Dos 36 conjuntos de métricas, somente sete conjuntos são do Tipo 1 (contém métricas disponíveis e indicadas para o auxílio do processo de análise cognitiva) e dois conjuntos são do Tipo 2 (contém métricas disponíveis com possibilidade de derivação de novas métricas a partir do conjunto obtido). Os outros 27 conjuntos são métricas não disponíveis ou não difundidas distribuídas entre os demais tipos já apresentados na Seção 4.1. A porcentagem de métricas disponíveis (Tipos 1 e 2) para suportar a metodologia é relativamente baixa: sendo 19,44% do Tipo 1 e 5,56% do Tipo 2, representando 25% da quantidade de conjuntos obtidos da abordagem *top-down*. Assim, a abordagem *top-down* indica que há a necessidade de derivação de novas métricas específicas para detecção de oportunidades de *refactoring*. Este é um dos principais obstáculos para a implementação da metodologia apresentada na Figura 2. Todavia, boa parte das métricas não disponíveis (58,33% distribuídos da seguinte forma: Tipo 3 com 33,33%, Tipo 4 com 11,11% e Tipo 6 com 13,89%) são factíveis e implementáveis, enquanto que somente 16,67% das métricas não disponíveis são do Tipo 5 (métricas não disponíveis ou não difundidas, cujo resultado depende fortemente de análise cognitiva). Este é um resultado importante pelo fato de mostrar que a metodologia descrita na Figura 2 é factível.

4.2. Estudo de Caso usando a Abordagem Bottom-Up

O objetivo deste estudo de caso é a coleta de informações a respeito do comportamento das métricas disponíveis antes, durante e depois do uso de uma seqüência específica de *refactorings*. Mais especificamente, a análise das medidas coletadas no estudo *bottom-up* vão fornecer subsídios para:

a) Verificar se o que foi previsto na abordagem *top-down* ocorreu.
b) Verificar ocorrências não previstas pela abordagem *top-down*.
c) Identificar ocorrências que foram previstas, mas não ocorreram.

4.2.1. Descrição do Estudo de Caso da Abordagem *Bottom-Up*

Este estudo de caso consiste na execução de uma seqüência de *refactorings* em uma aplicação para obtenção de um conjunto de medidas antes, durante e depois da seqüência de *refactorings*. Este cenário foi adotado levando-se em consideração que bons resultados no uso de *refactoring* somente serão obtidos se os mesmos forem utilizados de forma encadeada. O seu uso de forma isolada promove somente pequenas modificações, enquanto que de forma encadeada e combinada tem grande impacto em uma aplicação [12].

Foi adotado no estudo de caso um exemplo obtido mediante consulta por correio eletrônico ao próprio Fowler. O exemplo demonstra uma seqüência de 77 *refactorings* agrupados em cinco *bad smells* cujo objetivo é a melhoria do uso do conceito de herança. A aplicação possui classes que inicialmente não estão relacionadas, mas que possuem comportamento similar. Ao longo do processo de *refactoring* tais classes são reestruturadas em uma superclasse e quatro subclasses. A aplicação é pública e utiliza a linguagem Java, o que tem várias vantagens:

a) Representatividade da linguagem no universo das aplicações OO.
b) Amplo uso da linguagem em estudos sobre *refactoring*.
c) A aplicação já foi adotada em vários estudos de assuntos relacionados a *refactoring* por outros grupos de pesquisa [12].
d) Sua acessibilidade possibilita que outros grupos de pesquisa possam reproduzir e expandir os resultados obtidos.
e) A aplicação foi desenvolvida de forma independente, desta forma as abordagens descritas aqui não influenciaram no processo de desenvolvimento de *software*.
f) A aplicação, a identificação dos *bad smells* e o uso dos *refactorings* estão bem documentados [12], possibilitando a análise dos resultados obtidos.

O principal risco à validade do estudo de caso é que os resultados obtidos estão limitados aos *bad smells* detectados e à seqüência de *refactorings* adotada. O estudo de caso não aborda o conjunto completo de *bad smells* e *refactorings* sugeridos por Fowler em seu livro [11].

4.2.2. Execução do Estudo de Caso da Abordagem *Bottom-Up*

O estudo de caso foi realizado com a versão 6 do *Together Contol Center* [17]. Esta ferramenta disponibiliza cerca de 40 métricas de código fonte. Na coleta dos dados para análise foram consideradas:

a) Execução de todas as etapas de *refactoring* sugeridas em [12]. As etapas foram documentadas e organizadas separadamente para permitir a análise posterior do *refactoring* aplicado.
b) Obtenção das medidas disponibilizadas pela ferramenta em cada etapa de *refactoring*.
c) Classificação das etapas segundo o *refactoring* aplicado e o *bad smell* ao qual está associado.

Para a análise dos dados obtidos no estudo de caso foram adotadas as seguintes etapas conforme apresentado na Figura 6:

Procedimento A: Medição dos valores para as métricas em cada etapa de *refactoring* para cada classe. Estes valores são organizados em uma tabela.

Procedimento B: Obtenção da variação entre as etapas consecutivas de *refactoring* para cada métrica por classe. Descarte das métricas que não apresentaram variação.

Procedimento C: A partir da tabela obtida no procedimento B, separam-se os valores para cada métrica em tabelas distintas e acrescentam-se as informações de *refactoring* e *bad smell* associados a cada etapa.

Procedimento D: Para a obtenção das maiores variações para cada métrica foi adotada a técnica *Box plots* [10], onde são obtidos os valores de mediana e os quatro quartis das métricas. Para que sejam analisados somente os valores fortemente afetados para uma dada métrica, são considerados somente os valores que não estiverem contidos na

amplitude inter-quartil (medida que define a diferença entre os primeiro e o terceiro quartis).

Procedimento E: Cria-se para cada métrica disponível uma tabela para que sejam apresentados os *refactorings* que exercem maior impacto na variação da mesma. Na tabela em questão, os atributos "*refactoring* afetado pela métrica" e "quantidade de valores fortemente afetados" são obtidos do *procedimento D*, "a quantidade de valores com variação" é obtida do *procedimento C* e o "total de *refactorings* aplicados no estudo de caso" é obtido após a aplicação de um filtro na tabela do *procedimento A,* onde são considerados somente os *refactorings* relacionados com a classe cujas medidas estão sendo analisadas.

A partir do *procedimento E* é obtida a informação de quais métricas apresentaram maior variação para um dado *refactoring*. Com os resultados obtidos do *procedimento E*, os *refactorings* similares foram agrupados para a posterior verificação das métricas afetadas e com maior variação. Espera-se também que os resultados obtidos do *procedimento E* sirvam para identificar quais métricas apresentaram maior variação antes e depois de cada *bad smell*.

4.2.3. Resultados do Estudo de Caso da Abordagem Bottom-Up

A abordagem aqui apresentada realizou estudo empírico para fornecer dados que possibilitassem ampliar o escopo do relacionamento entre *refactorings* e métricas, além da possibilidade de relacioná-los com *bad smells*. Para a identificação dos *refactorings* que apresentaram variação por métrica, usou-se o processo de análise representado na Figura 6.

Várias tabelas resultaram da etapa E, uma delas esta exemplificada na Tabela 6. Elas apresentam os *refactorings* que ocasionaram maior variação para uma dada métrica. Os campos "quantidade de valores fortemente afetados", "quantidade de valores com variação" e "total" são usados para informar o grau de importância da métrica para o *refactoring*. Assim, a métrica será tanto mais sensível ao uso de um determinado *refactoring* quanto maior a relação *quantidade de valores com variação / total*, onde o campo "total" informa quantas ocorrências do referido *refactoring* estão envolvidos com a classe em questão. A esta relação denominamos Coeficiente de Associação entre Métrica e *Refactoring* (CAMR). Uma outra relação, denominada Coeficiente de Associação Forte entre Métrica e *Refactoring* (CAFMR), expressa a *quantidade de valores fortemente afetados / total*. Estes dois coeficientes são usados para estabelecer o nível de associação entre métricas e *refactorings* variando no intervalo entre zero e um.

A associação entre *refactorings* e métricas é um artifício a ser usado na redução do volume do código a ser analisado. Este relacionamento servirá como base para a criação de heurísticas de detecção de oportunidades de *refactoring* através da criação de filtros baseados em faixas de valores para conjuntos de métricas com maiores associações com determinados *refactorings*. Várias tabelas foram criadas para a identificação das métricas com maior variação por *refactoring*, dentre elas apresentamos a Tabela 7 como exemplo. O conjunto completo das tabelas é apresentado em [7].

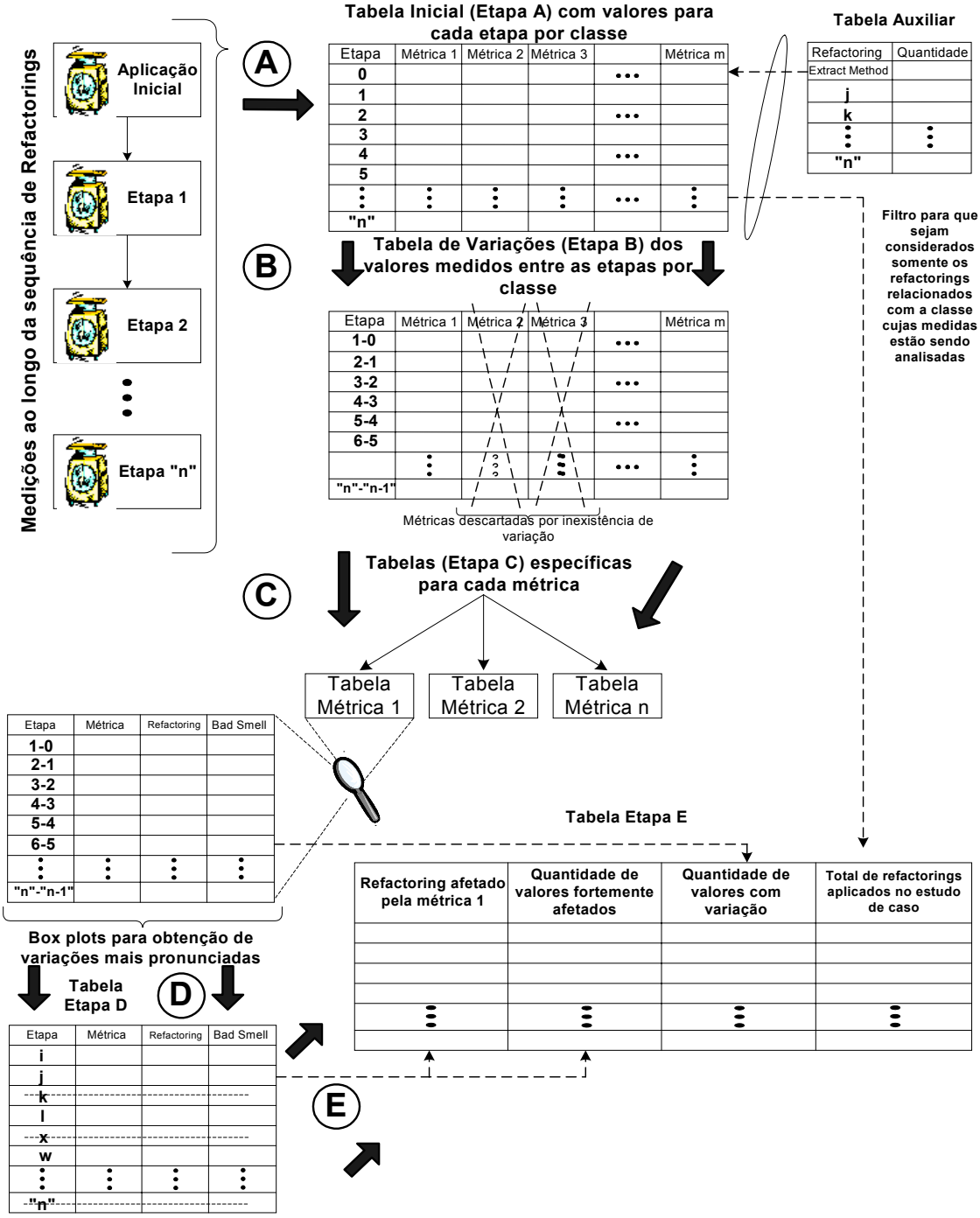

Figura 6: Processo de análise dos dados

Tabela 6: Identificação das principais variações para a métrica Cyclomatic Complexity (CC)

Refactoring afetado pela métrica CC	Quantidade de valores fortemente afetados	Quantidade de valores com variação	Total	CAMR	CAFMR
Migração de Método para Superclasse	2	2	6	1/3	1/3
Criação de novo método	1	2	2	1	½
Migração de Trecho de Código para uma Nova Subclasse	1	1	2	½	½
Dissolução de Variável Temporária/Substituição de Parâmetro por Método/Remoção de Parâmetros	1	1	1	1*	1*
Migração de Trecho de Código para um Novo Método e Parametrização de Método	1	1	1	1*	1*

*Refactorings com somente uma ocorrência

Em relação à identificação das métricas com maior variação por *bad smell*, serão necessários mais estudos de caso para expressar o grau de relacionamento entre *bad smells* e métricas, pois, devido ao número pequeno de ocorrências de *bad smells* reportados neste estudo de caso, não foi possível a obtenção de dados representativos para tal finalidade.

Tabela 7: Métricas afetadas por Migração de Trecho de Código para um Novo Método

Métrica	Quantidade de valores fortemente afetados	Quantidade de valores com variação	Total	CAMR	CAFMR
Hplen	1	4	4	1	¼
Locom3	3	3	4	¾	¾
Noprtr	1	1	4	¼	¼
PprivM	3	4	4	1	¾
PprotM	3	4	4	1	¾
PpubM	2	4	4	1	½

4.2.4. Análise dos Resultados

Em relação aos *refactorings* Migração de Método para Superclasse, Migração de Variável para Superclasse, Migração de Trecho de Código para um Novo Método, Criação de Subclasse, Criação de Método, verificou-se que o número de métricas associadas aos mesmos ocorre conforme descrito na Tabela 8. Pela distribuição apresentada, verifica-se que os resultados com maior número de ocorrências de *refactoring* trarão maior confiabilidade no que tange aos Coeficientes de Associação entre Métricas e *Refactorings* (CAMR e CAFMR). Reconhecemos a necessidade da realização de um número maior de estudos de caso para que seja alcançado um número apropriado de ocorrências dos *refactorings* e, conseqüentemente, maior representatividade nos resultados obtidos. Os outros oito *refactorings* que ocasionaram variação das métricas ocorreram somente uma vez, fato que os tornam, neste estudo de caso, pouco representativos para fins de análise.

Tabela 8: Relacionamento entre refactorings e métricas

Refactoring	Número de métricas associadas	Número de ocorrências do refactoring
Migração de Método para Superclasse	24	6
Migração de Variável para Superclasse	4	4
Migração de Trecho de Código para um Novo Método	6	4
Criação de Subclasse	18	2
Criação de Método	16	2

5. Considerações Finais

Este artigo apresenta uma metodologia que introduz métricas na aplicação de *refactorings*. Argumenta-se que, partindo da análise de código fonte, há a possibilidade de redução da dependência da análise cognitiva através do estabelecimento de conjuntos de métricas relacionados com tipos específicos de *refactorings*. Como um passo inicial no sentido do estabelecimento desta metodologia, este artigo apresentou duas abordagens para relacionar métricas, *refactorings* e *bad smells* [6][7]: a abordagem *top-down* que usa o paradigma Meta Pergunta Métrica seguindo um processo analítico e a abordagem *bottom-up* seguindo um processo empírico.

O resultado das duas abordagens indica que métricas podem auxiliar no uso de *refactorings*. Um estudo de caso da aplicação da abordagem *top-down* constatou que as métricas disponíveis representam 25% do conjunto de métricas identificadas analiticamente (somatório dos percentuais das métricas dos Tipos 1 e 2), enquanto que as métricas não disponíveis representam os restantes 75% do conjunto. Este último valor foi obtido da soma dos seguintes percentuais: 58,33% são métricas factíveis e implementáveis (somatório dos percentuais das métricas dos Tipos 3, 4 6); 16,67% são métricas não disponíveis ou não difundidas cujo resultado depende fortemente de análise cognitiva (métricas do Tipo 5). Este é um resultado importante. Ele indica que a metodologia proposta na Figura 2 é factível, pois a maioria das métricas necessárias a ela são implementáveis.

A abordagem *bottom-up* constatou a existência de relacionamento entre métricas e *refactorings*. Todavia, ainda serão nencessários mais estudos de caso para expressar o grau deste relacionamento. O relacionamento entre *bad smells* e métricas não pôde ser avaliado devido ao número pequeno de ocorrências de *bad smells* reportados no estudo de caso.

Da mesma forma argumentada por Fowler [11], a metodologia proposta indica que a última e definitiva etapa para a decisão sobre a aplicação de um determinado *refactoring* é a análise cognitiva. Entretanto, para se chegar a esta última etapa, métricas, conforme indicado pelas abordagens *top-down* e *bottom-up*, podem representar um aspecto facilitador importante nas etapas do uso de *refactoring*.

Uma vez implementada, a metodologia proposta na Seção 3 proporcionará a redução do volume de código fonte a ser analisado pela equipe de *refactoring*. Além disso, a metodologia possibilitará a criação de uma base concreta para analisar a conveniência ou não de tratar um *bad smell*. Isto levantará a possibilidade de relacionamento entre a análise cognitiva tradicionalmente feita antes do *refactoring* e a avaliação empírica possibilitada pelas métricas utilizadas para quantificar *refactorings*.

Outro ponto importante é que o uso da abordagem MPM no contexto da metodologia proposta proporciona uma oportunidade de integração de duas áreas diferentes da engenharia de software, nominalmente, *refactoring* e métricas.

É importante ressaltar que este trabalho propôs uma metodologia e apenas deu um passo inicial na direção de sua implementação. Para consolidar tal implementação serão necessários, dentre outras, as seguintes atividades:

a) Realização de mais estudos de caso para que seja criada uma base de conhecimento mais ampla. Esta base de conhecimento será necessária para consolidar o relacionamento entre *refactorings*, *bad smells* e métricas.
b) A implementação das métricas identificadas na abordagem *top-down* e o uso das mesmas na metodologia.
c) Realização de estudos de caso para a definição de faixas de valores aceitáveis para as métricas indicadas nas abordagens *top-down* e *bottom-up.*

6. Referências Bibliográficas

[1] Astels, D. *Refactoring with UML*. Disponível em <www.xp2002.org_atti_DaveAstels--RefactoringWithUML.pdf>. Acesso em 02 de agosto de 2002.

[2] Basili, V. R., Rombach, H. D. *The TAME Project: Towards Improvement-Oriented Software Environments.* IEEE Transactions on Software Eng., vol. 14. no 6, pp. 758-773, June 1988.

[3] Basili, V., Weiss, D. *A methodology for collecting valid software engineering data.* IEEE Transactions on Software Eng., November 1984.

[4] Beck, K. *Extreme Programming Explained: Embrace Change.* Addison Wesley, 1999.

[5] Briand, L. *Measurement and Quality Modelling of OO Systems.*The Sixth International Symposium on Software Metrics. November 4-6, 1999. Boca Raton, Florida, USA.

[6] Carneiro, G. F., Mendonça Neto, M.G. Usando Medição de Código Fonte para Refactoring. In 2nd Ibero-American Symposium on Software Engineering and Knowledge Engineering. Oct, 2002. Salvador, Bahia, Brazil.

[7] Carneiro, G. F. Usando Medição de Código Fonte para Refactoring. Dissertação de Mestrado. Universidade Salvador. Abril, 2003. Salvador, Bahia, Brazil.

[8] Demeyer, S., Ducasse, S., Nierstrasz, O. *Finding Refactorings via Change Metrics.* Proceedings OOPSLA´2000, ACM Press.

[9] Ernst, M. D., Cockrell J., Griswold, W. G., David Notkin. *Dynamically discovering likely program invariants to support program evolution.* IEEE Transactions on Software Engineering, 27(2):1-25, February 2001.

[10] Fenton, N. and Pleeger S. *Software Metrics: A Rigorous and Pratical Approach.* Second Edition, PWS Publishing Company, 1997.

[11] Fowler, M. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 2000.

[12] Fowler, M. Capítulo 15 não publicado no livro *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 2000, mas disponibilizado pelo autor em http://www.refactoring.com/rejectedExample.pdf. Documento acessado em 20/03/2002.

[13] Kataoka, Y., Ernst, M., Griswold, W., Notkin, D.. *Automated Support for Program Refactoring using Invariants*. In ICSM'01, Proceedings of the International Conference on Software Maintenance, (Florence, Italy), November 6-10, 2001, pp. 736-743.

[14] Mendonça, M., Basili, V.. *Validation of an Approach for Improving Existing Measurement Frameworks.* IEEE Transactions on Software Engineering, v.26, n.6, p.484-499, 2000.

[15] Opdyke, W. *Refactoring Object-Oriented Frameworks.* Ph.D. Thesis. University of Illinois at Urbana-Champaign, 1992.

[16] Roberts, D. *Practical Analysis for Refactoring.* PhD Thesis. University of Illinois at Urbana-Champaign, 1999.

[17] TogetherControlCenter. Acesso em 03 de julho de 2002. Disponível em <http://www.togethersoft.com/products/controlcenter/index.jsp>.

[18] Tourwé, T., Brichau, J., Mens, T. *Using Declarative Metaprogramming to Detect Possible Refactorings.* ASE 2002. Acesso em 6 de setembro de 2002. Disponível em <http://www.cs.ubc.ca_~kdvolder_Workshops_ASE2002_DMP_papers_08tourwe-brichau-mens.pdf>.