Uma Estratégia de Reestruturação de Modelos baseada em Métricas para Apoiar a Reengenharia de Software Orientado a Objetos para Componentes

Ana Maria Moura[1], Aline Vasconcelos[2], Cláudia Werner[1]

[1]COPPE/UFRJ – Programa de Engenharia de Sistemas e Computação
Caixa Postal 68511 – CEP: 21945-970 – Rio de Janeiro – RJ – Brasil
[2]CEFET Campos (Centro Federal de Educação Tecnológia de Campos)
Dr. Siqueira, 273 – Pq. Dom Bosco – CEP: 28030-130 - Campos dos Goytacazes- RJ

{anammmoura, aline, werner}@cos.ufrj.br

Abstract. *Object oriented (OO) systems can become more maintainable and reusable by restructuring their classes into cohesive groups with a defined functionality, generating components. Software reengineering represents an approach that transforms an existent system to a new paradigm or platform, guaranteeing its survival. In this paper, it is proposed a reengineering approach from objects to components supported by a restructuring strategy based on metrics, being applied on OO static models. These metrics are based on CBD principles (Component Based Development). The proposed strategy is evaluated by a pilot case study.*

Resumo. *Sistemas orientados a objetos (OO) podem se tornar mais manuteníveis e reutilizáveis através da reestruturação de suas classes em grupos coesos e com uma funcionalidade definida, formando componentes. A reengenharia de software representa uma abordagem que transforma um sistema existente para um novo paradigma ou plataforma, lhe garantindo maior sobrevida. Neste artigo, é proposta uma abordagem de reengenharia de objetos para componentes apoiada por uma estratégia de reestruturação baseada em métricas, a qual é aplicada sobre modelos OO estáticos. As métricas se baseiam em princípios de DBC (Desenvolvimento Baseado em Componentes). A estratégia proposta é avaliada por um estudo de caso piloto.*

1. Introdução

De acordo com Cheesman e Daniels (2001), o DBC se diferencia das outras abordagens de desenvolvimento (ex: orientação a objetos) através da separação entre a especificação do componente e a sua implementação, e na divisão dos serviços dos componentes em interfaces. As interfaces podem ser compreendidas como uma representação explícita das funcionalidades de um componente, sendo o meio de comunicação entre os mesmos. Deste modo, evita-se que mudanças na implementação de um componente afetem os demais, uma vez que os serviços continuam sendo garantidos pelas interfaces. A transformação de um software orientado a objetos para componentes pode torná-lo mais manutenível e reutilizável.

Neste artigo, adota-se o seguinte conceito de componentes: "Componentes de software são artefatos auto-contidos, claramente identificáveis, que descrevem ou realizam uma função específica e têm interfaces claras, documentação apropriada e um grau de reutilização definido" [Sametinger 1997].

Neste contexto, a reengenharia, que por definição é o exame e alteração de um sistema para reconstituí-lo em uma nova forma com a re-implementação desta nova forma [Chikofsky e Cross 1990], pode ser aplicada na mudança de paradigma OO para DBC. A reengenharia envolve a engenharia reversa, podendo envolver alguma forma de reestruturação, seguida de engenharia progressiva [Chikofsky e Cross 1990]. Em uma abordagem de reengenharia, o primeiro passo é compreender o sistema alvo. Na literatura, encontram-se diversas abordagens de engenharia reversa que podem ser utilizadas na obtenção de um modelo de projeto atual para um sistema OO, empregando a análise estática e/ou dinâmica (ex: [Bojic e Velasevic 2000; Vasconcelos 2007]). Após a compreensão do sistema atual, o projeto do sistema pode ser reestruturado, de forma que a estrutura do sistema seja melhorada sem alteração de suas funcionalidades. A terceira e última etapa da reengenharia é a engenharia progressiva que é o processo tradicional de mover-se das abstrações de alto nível, projetos independentes de implementação, para a implementação física de um sistema.

A abordagem proposta de reengenharia de software OO para componentes é composta por duas etapas: a reestruturação dos modelos de pacotes e classes, que visa obter agrupamentos de classes (i.e. pacotes) candidatos a componentes, de modo que atendam aos princípios de DBC; e a geração dos componentes propriamente, que visa obter um modelo de componentes com suas interfaces a partir dos agrupamentos de classes (i.e. pacotes) reestruturados. A ênfase deste artigo é na etapa de reestruturação estando a segunda etapa ainda em desenvolvimento. O modelo deve ser previamente obtido através de alguma abordagem de engenharia reversa para que as decisões de projeto e implementação sejam re-aproveitadas. Garante-se assim que o modelo esteja atualizado e reflita os acoplamentos reais existentes entre as classes do sistema.

Na literatura, existem alguns trabalhos de reengenharia para componentes e/ou extração de componentes. Prado (2005) propõe uma estratégia para a reengenharia de sistemas legados procedurais em componentes distribuídos. A abordagem proposta neste artigo difere-se pelo fato do sistema alvo ser OO e por gerar componentes a partir de agrupamentos de classes (i.e. pacotes). Ganesan e Knodel (2005) apresentam uma estratégia para identificar componentes específicos de domínio reutilizáveis a partir de sistemas OO, suportando a migração para uma linha de produtos. A abordagem proposta neste artigo difere-se por reestruturar todo o sistema ao invés de selecionar algumas classes para compor os componentes, re-organizando todos os agrupamentos para que atendam a alguns princípios de DBC. Washizaki e Fukazawa (2004) propõem uma estratégia para a extração automática de componentes a partir de sistemas OO com foco na linguagem Java. Porém, comparando-se com a abordagem proposta neste artigo, esta última não é dependente de linguagem de programação, uma vez que trabalha em nível de modelo, o qual pode ser re-implementado em diferentes tecnologias de componentes (ex: .NET, EJB etc.). Além disso, a abordagem proposta é semi-automatizada, permitindo que o engenheiro de software tenha flexibilidade sobre os novos componentes gerados, sendo apoiado por métricas na sua tomada de decisão.

Partindo desta Introdução, o restante do artigo está organizado da seguinte forma: a Seção 2 apresenta uma visão geral da estratégia de reestruturação do modelo proposta, apresentando as métricas e as reestruturações sugeridas com base nos resultados das métricas; a Seção 3 apresenta um estudo de caso conduzido para avaliar a etapa de reestruturação da abordagem e as lições aprendidas obtidas; e a Seção 4 apresenta as considerações finais e trabalhos futuros.

2. Uma Estratégia de Reestruturação baseada em Métricas

A reestruturação do modelo de classes visa que os agrupamentos de classes, candidatos a componentes, tenham "bons projetos" para componentes. Na literatura há trabalhos que enfatizam princípios de projetos de componentes [Vitharana 2004; Gonçalves *et al.* 2006; Ganesan e Knodel 2005]. Em síntese, os princípios são: não deve existir relacionamento de herança entre componentes distintos, pois um componente não deve conhecer a implementação do outro; os componentes devem prover uma funcionalidade específica; classes generalizadas são boas candidatas a comporem componentes, pois permitem maior adaptação e extensão em novos sistemas; a comunicação inter-componentes, feita via suas interfaces, é muito cara em termos de tempo e recursos de plataforma e, por isso, deve-se diminuir o acoplamento entre componentes, a fim de minimizar o tráfego na rede.

A estratégia proposta de reestruturação, como mostra a Figura 1, é realizada a partir dos modelos de pacotes e classes que representem o projeto detalhado atual do sistema, recuperado através de alguma abordagem de engenharia reversa. Este é transformado para um novo paradigma. O engenheiro deve reestruturar estes modelos, solicitando a coleta das métricas. Os resultados são exibidos junto aos valores de referência e interpretação das métricas. A cada métrica está associado um conjunto de reestruturações que são sugeridas comparando-se os valores obtidos aos seus valores de referência. Após o engenheiro selecionar uma reestruturação, o modelo é atualizado para refletir a reestruturação aplicada. A seguir, um novo ciclo é iniciado e as métricas são coletadas novamente, assim como novas sugestões de reestruturações são geradas.

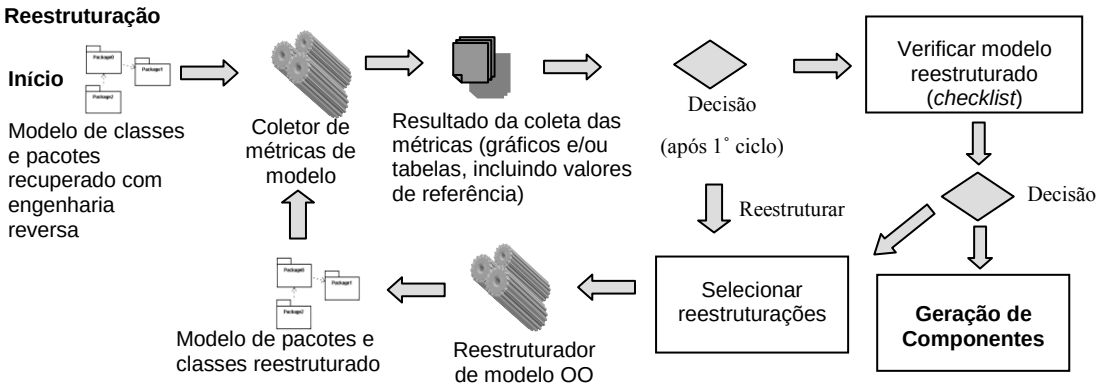

Figura 1. Estratégia proposta de reestruturação de modelos OO estáticos.

Como mostra a Figura 1, o engenheiro de software após o primeiro ciclo pode fazer uma verificação do modelo reestruturado, com o objetivo de avaliar a preservação das suas funcionalidades e melhoria da qualidade. Após a verificação, o engenheiro de software pode optar por continuar a reestruturar, baseando-se nos princípios de DBC, nos valores das métricas e em seu conhecimento do domínio, para que o modelo tenha melhores agrupamentos. Ele ainda pode optar por gerar componentes a partir de cada agrupamento reestruturado, caso já tenha conseguido bons agrupamentos com melhores valores de métricas e semanticamente organizados. Vale ressaltar que a abordagem requer um engenheiro com conhecimento do domínio para guiar o processo.

2.1. As Métricas

Como apresentado na Seção 2, as reestruturações são apoiadas por um conjunto de

métricas. Cada métrica foi selecionada ou adaptada com base nos seguintes critérios: ser mensurável a partir de um modelo OO; ser coletada de forma automática; auxiliar o cumprimento dos princípios de DBC. A seguir o conjunto inicial de métricas é apresentado com o seguinte *template* (modelo) proposto: nome, nível de granularidade (pacote ou classe), interpretação com a justificativa da métrica, descrição, fórmula e valor de referência. As características do *template* visam facilitar a compreensão da métrica. Como estas métricas são obtidas a partir de modelos OO, toda vez que a métrica se refere a um componente, o mesmo representa um pacote. Cada valor de referência de métrica foi obtido na literatura, quando disponível, ou este valor pode ser comparado à média dos valores obtidos quando a abordagem for utilizada pela organização. Alguns valores de referência podem ser reconfigurados à medida que a organização crie uma base de referência (*baseline*).

⇨ **Número de Filhos por Superclasse para cada Componente** (NOCC – *Number Of Children per Component Superclass*). Adaptada de NOC [Chidamber e Kemerer 1994].

- **Nível de Granularidade:** Classe. **Interpretação:** Segundo Vitharana (2004) superclasses não devem possuir subclasses em componentes distintos ao qual ela pertence, pois este relacionamento faz com que um componente conheça detalhes da implementação do outro. **Descrição:** Mensura o número de filhos da superclasse em cada componente. **Fórmula:** $NOCC_k = TNOC_{ki}$, onde $TNOC_{ki}$ – total de subclasses da superclasse k por componente i, onde k varia de 1 ao número total de superclasses do modelo e i varia de 1 ao número total de componentes do sistema. **Valor de referência**: 0 para componentes distintos ao da superclasse (não-reconfigurável).

⇨ **Número de Implementações de Interfaces por Componente** (NOIC – *Number Of Implementation per Component*). Adaptada de NOC [Chidamber e Kemerer 1994].

- **Nível de Granularidade:** Classe. **Interpretação:** Classes que implementam uma mesma interface estão inseridas em um contexto comum e fornecem os mesmos serviços, de modo que uni-las deixará a funcionalidade em um único componente. **Descrição:** Mensura o número de classes que implementam uma determinada interface em cada componente. **Fórmula:** $NOIC_k = TNOIC_{ki}$, onde $TNOIC_{ki}$ – total de classes implementadoras da interface k por componente i, onde k varia de 1 ao número total de interfaces e i varia de 1 ao número total de componentes do sistema. **Valor de referência**: 0 para componentes distintos ao da interface (reconfigurável).

⇨ **Acoplamento entre Classes por Componente** (CBOC – *Coupling Between Object Classes per Component*). Adaptada de CBO [Chidamber e Kemerer 1994].

- **Nível de Granularidade:** Classe. **Interpretação:** o acoplamento da classe com relação ao componente ao qual ela pertence deve ser igual ou maior do que o acoplamento com relação aos demais componentes. **Descrição:** mensura o valor de CBO (*Coupling Between Object Classes*) de cada classe k para cada componente i. **Fórmula:** $CBOC_k = CBO_{ki}$, onde CBO_{ki} – CBO de cada classe k para cada componente i, onde i varia de 1 ao número total de componentes do modelo e k varia de 1 ao número total de classes. **Valor de referência**: não possui, devendo ser montado um gráfico mostrando o crescimento do acoplamento da classe por componente.

⇨ **Abstração** (A - *Abstraction*). Retirada de [Gonçalves *et al.* 2006].

- **Nível de Granularidade:** Pacote. **Interpretação:** um componente deve ser abstrato ou bastante abstrato, sendo mais facilmente adaptado em diferentes contextos de

reutilização dentro de um domínio [Gonçalves *et al.* 2006]. **Descrição:** razão entre o número de classes abstratas e interfaces e o número total de classes e interfaces de cada componente. **Fórmula:** $A_i = AC_i/TC_i$, onde AC_i – Total de classes abstratas e interfaces do componente i e, TC_i – total de classes do componente i, onde i varia de 1 ao número total de componentes do modelo. **Valores de referência**: 0 – 0,25: bastante concreto; 0,26 – 0,5: concreto; 0,51 – 0,75: abstrato; 0,76 – 1: bastante abstrato (reconfigurável).

⇨ **Componentes Conectados** (ConnComp – *Connected Components*). Retirada de [Sdmetrics 2007].

- **Nível de Granularidade:** Pacote. **Interpretação:** grupos de classes isolados no componente podem significar que o componente possui mais de uma funcionalidade e perda de coesão. **Descrição:** mensura o número de grupos de classes conectadas, isoladas dos demais grupos, em um componente. **Fórmula:** $ConnComp_i = TGCC_i$, onde $TGCC_i$ – total de grupos de classes conectadas de um componente i, onde i varia de 1 ao número total de componentes. **Valor de referência**: 1 (não-reconfigurável).

⇨ **Acoplamento por Componente Requerido** (CRC – *Coupling per Required Component*). Adaptada de [Blois 2006].

- **Nível de Granularidade:** Pacote. **Interpretação:** verifica quantos componentes são necessários ao utilizar cada componente, pois o número de componentes requeridos deve ser o menor possível. A comunicação inter-componentes é muito custosa [Vitharana 2004]. **Descrição:** a métrica soma todos os componentes que possuem classes, as quais o componente avaliado depende. **Fórmula:** $CRC_i = TRC_i$, onde TRC_i – total de componentes requeridos pelo componente i, onde i varia de 1 ao número total de componentes. **Valor de referência**: comparar os componentes requeridos por um componente com a média de componentes requeridos do modelo (reconfigurável).

O conjunto de métricas proposto é apenas inicial, podendo ser estendido à medida que a abordagem seja utilizada e sejam descobertas novas métricas. É proposta uma ordem de priorização das métricas para guiar o engenheiro nas reestruturações. A ordem leva em consideração métricas de classe, antes de métricas de pacotes, porque nesta ordem os pacotes podem deixar de existir ou tornarem-se mais coesos minimizando as sugestões de reestruturações a partir das métricas de pacote. Dentro desta ordem, a sub-priorização de métricas de classe é: CBOC, seguida de NOCC ou NOIC, porque nesta ordem as classes serão movidas para os pacotes aos quais estejam mais acopladas e, por isso, as hierarquias e realizações poderão ser movidas para os pacotes aos quais a maioria de seus elementos esteja mais acoplada. A sub-priorização das métricas de pacote é: A, CC e CRC, porque nesta ordem serão geradas superclasses ou interfaces que podem unir grupos de classes isolados e, em seguida, a extração de novos componentes ocorrerá em menor número e, conseqüentemente, a geração de novos componentes poderá ser feita a partir de pacotes mais coesos e menos acoplados. Os relacionamentos entre pacotes devem ser revistos após as reestruturações, pois as movimentações de classes, criação e eliminação de pacotes que venham a ficar vazios, podem afetar as dependências originais entre os pacotes.

2.2. As Reestruturações

O conjunto de reestruturações é apresentado com o seguinte *template*: nome; métrica relacionada; mecanismo; e opcionalidade, ex: opcional (O) ou fortemente recomendada (FR), onde fortemente recomendada significa a possibilidade de ferir algum princípio de

DBC. A opcionalidade não influencia a ordem de aplicação das reestruturações porque as mesmas são guiadas pelas respectivas métricas (Seção 2.1) que lhes deram origem.

⇨ **Mover Hierarquia** – **Métricas relacionadas:** Número de Filhos de Superclasse por Componente e Número de Implementações de Interfaces por Componente **Mecanismo:** mova as superclasses e/ou interfaces para os pacotes que possuam o maior número de subclasses ou classes implementadoras imediatas, e também toda a hierarquia relacionada acima e subclasses ou demais classes implementadoras para o pacote de destino. **Opcionalidade:** O para interfaces e FR para superclasses.

⇨ **Mover Classe** – **Métrica relacionada:** Acoplamento entre Classes por Componentes. **Mecanismo:** Mova as classes para os pacotes aos quais elas possuam o maior valor de acoplamento para os demais elementos. **Opcionalidade:** O.

⇨ **Extração de Superclasse** – **Métrica relacionada:** Abstração. **Mecanismo:** Crie superclasses para as classes do pacote que possuam atributos com o mesmo nome e/ou operações com assinaturas similares, quando a métrica estiver indicando que o componente é bastante concreto ou concreto. No caso de operações com o mesmo nome, mas com variação de parâmetros (i.e. sobrecarga), tipos de retorno ou quando houver diferenças nos tipos de atributos, uma notificação será gerada e o engenheiro deverá escolher a(s) opção(ões) para a superclasse. **Opcionalidade:** O.

⇨ **Extração de Interface** – **Métrica relacionada:** Abstração. **Mecanismo:** Crie interfaces para as classes do pacote que possuam somente operações com a mesma assinatura ou assinaturas similares, quando a métrica estiver indicando que o componente é bastante concreto ou concreto. No caso de variação de parâmetros ou tipos de retorno, uma notificação será gerada e uma alguma(s) opção(ões) irá(ão) para a interface. **Opcionalidade:** O.

⇨ **Extração de Componente** – **Métrica relacionada:** Componentes Conectados. **Mecanismo:** Crie novos pacotes ou componentes a partir dos grupos de classes conectados e isolados dos demais do pacote. **Opcionalidade:** FR.

⇨ **Colapso de Componentes** – **Métrica relacionada:** Acoplamento por Componente Requerido. **Mecanismo:** Unir dois ou mais pacotes através de seus projetos ou em um novo pacote, considerando-se as seguintes situações: um pacote requer outros pacotes, os quais não são requeridos por outros pacotes; ou os pacotes estão envolvidos em uma dependência circular. **Opcionalidade:** O.

3. Estudo de Caso Piloto

Um estudo de caso piloto foi realizado para verificar a viabilidade da estratégia de reestruturação proposta, antes que o seu ferramental fosse desenvolvido. Deste modo, elimina-se o risco de perder o esforço de desenvolvimento em uma abordagem inviável. O estudo foi realizado com o suporte do ambiente Odyssey [Odyssey 2008], no qual foi obtido o projeto do sistema alvo, através da sua ferramenta de engenharia reversa estática, o Ares [Veronese e Neto 2001], que recupera modelos de pacotes e classes do código de sistemas Java. Um engenheiro do domínio aplicou a estratégia proposta. O estudo foi desenvolvido em três etapas: planejamento, execução e avaliação.

Planejamento – O sistema alvo foi o TraceMining [Vasconcelos 2007] que é uma ferramenta utilizada na recuperação de arquitetura de referência através de técnicas de mineração de dados e está presente no ambiente Odyssey. O projeto desse sistema é

composto por cinco pacotes, como mostra a Figura 2a, e quarenta e três classes. As métricas sugeriram quarenta e uma reestruturações. Neste estudo, não foram realizados todos os possíveis ciclos de reestruturações devido à falta de maior apoio ferramental.

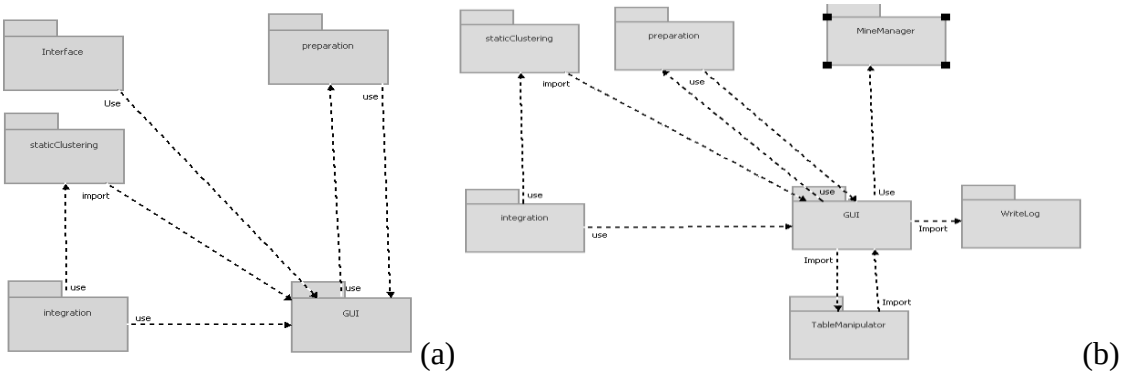

Figura 2. Modelo de pacotes antes (a) e depois (b) da reestruturação.

Execução – O engenheiro do domínio selecionou as reestruturações baseando-se nos gráficos gerados para as métricas, calculadas manualmente, seus valores de referência e em seu conhecimento do domínio. Por exemplo, a partir do pacote GUI foi extraída a métrica componentes conectados (CC), obtendo-se o valor 6, por isso, foram sugeridas 6 sugestões de extração de componentes. Uma das reestruturações aplicadas foi a extração do pacote TableManipulator, que tornou o pacote GUI que englobava diferentes funcionalidades mais coeso. O pacote Preparation obteve o valor 3 para a métrica CC e, após ser reestruturado, deu origem ao pacote WriteLog. O resultado se encontra na Figura 2b.

Avaliação – Após o estudo de caso, obteve-se indícios da viabilidade da estratégia de reestruturação, que gerou agrupamentos mais semânticos, na opinião do engenheiro do domínio, e mais convenientes aos princípios de DBC (Seção 3). Os valores das métricas melhoraram, reduzindo-se o número de restruturações para 22. O estudo de caso indicou ainda melhorias para o conjunto de métricas e reestruturações e a necessidade de indicação do nível de granularidade das métricas (classe ou pacote). O engenheiro do domínio teve dificuldade na aplicação de várias reestruturações ao mesmo tempo, pois a cada reestruturação os valores das métricas podem se modificar. Assim, a cada reestruturação o modelo deve ser reavaliado e novas reestruturações sugeridas (Figura 1). Indicou-se ainda a necessidade de priorizar a aplicação das reestruturações, visto a diversidade de reestruturações possíveis no mesmo ciclo. Vale ressaltar que a abordagem apresentada já contempla essas melhorias.

4. Conclusões e Trabalhos Futuros

Neste artigo, foi descrita uma estratégia de reestruturação de modelo de pacotes e classes, que é parte de uma abordagem de reengenharia de software OO para componentes. Como contribuições estão a possibilidade de tornar os sistemas OO mais reutilizáveis e manuteníveis, além do conjunto de métricas e reestruturações proposto.

Como trabalhos em andamento, encontram-se: a definição de um *checklist* para apoiar a verificação do modelo reestruturado e o desenvolvimento de um apoio ferramental para a abordagem proposta, após o estudo de caso piloto indicar a viabilidade da estratégia proposta. Como trabalho futuro está o desenvolvimento da etapa de geração de componentes e suas interfaces e a realização de estudos de caso

para avaliar o ferramental, envolvendo sistemas reais e em maior escala.

5. Referências

Blois, A. P. T. B. (2006). Uma Abordagem de Projeto Arquitetural Baseado em Componentes no Contexto de Engenharia de Domínio. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

Bojic, D., Velasevic, D. (2000). A Use-Case Driven Method of Architecture Recovery for Program Understanding and Reuse Reengineering. In: 4th European Software Maintenance and Reengineering Conference. Zuriq, Swiss, pp. 23-31, February.

Cheesman, J., Daniels, J. (2001). UML components: a Simple Process for Specifying Component-based Software. Addison-Wesley Longman Publishing.

Chidamber, S. R., Kemerer, C. F. (1994). A Metrics Suite for Object Oriented Design. IEEE Trans. Software Eng., vol. 20, pp. 476–493, June 1994.

Chikofsky, E. J.; Cross II, J. H. (1990). Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software, v. 7, n. 1*. pp. 13-17.

Ganesan, D., Knodel, J. (2005). Identifying Domain-Specific Reusable Components from Existing OO Systems to Support Product line Migration. *Workshop on Reengineering towards Product Lines*. Pittsburgh, USA, pp. 27–36, Nov.

Gonçalves, J. J., Oliveira, T. M. A., Oliveira K. M. (2006). Métricas de Reusabilidade para Componentes de Software. VI Workshop de Desenvolvimento Baseado em Componentes. Recife, Pernambuco, Brasil, pp. 14-21, Dez.

Odyssey. (2008). In: http://reuse.cos.ufrj.br/odyssey, acessado em 17/03/2008.

Prado, A.F. (2005). Reengenharia de Software Baseado em Componentes. In: Gimenes, I.M.S., Huzita, E.H.M. (eds), Desenvolvimento Baseado em Componentes: Conceitos e Técnicas, Rio de Janeiro, Ciência Moderna.

Sametinger, J. (1997). Software Engineering with Reusable Components. Springer-Verlag New York, Inc.

Sdmetrics. (2007). In: http://www.sdmetrics.com, acessado em 30/06/2007.

Vasconcelos, A. (2007). Uma Abordagem de Apoio à Criação de Arquiteturas de Referência de Domínio Baseada na Análise de Sistemas Legados. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

Veronese, G. O., Netto, F. J. (2001). ARES: Uma Ferramenta de Auxílio à Recuperação de Modelos UML de Projeto a partir de Código Java. Projeto Final de Curso de Bacharelado em Informática- Instituto de Matemática/UFRJ, Outubro.

Vitharana, P.; Jain, H.; Zahedi, F. (2004). Strategy-Based Design of Reusable Business Components. IEEE Trans. on Systems, Man and Cybernetics, vol 34, pp. 460 – 474.

Washizaki, H., Fukazawa, Y. (2005). A Technique for Automatic Component Extraction from Object-Oriented Programs by Refactoring. Science of Computer Programming, vol.56, no.1-2, pp.99-116.