Critérios de Manutenibilidade para Construção de Produtos de Software Orientados a Aspectos

Rodrigo P. dos Santos[1,2], Cláudia M. L. Werner[2], Heitor A. X. Costa[1], Paulo A. P. Júnior[1], André F. Amâncio[1], Antônio M. P. de Resende[1]

[1]Grupo PqES/DCC/UFLA, Universidade Federal de Lavras, Brasil
Caixa Postal 3037 – CEP 37200-000 Lavras/MG, Brasil

[2]COPPE/UFRJ, Universidade Federal do Rio de Janeiro, Brasil
Caixa Postal 68511 – CEP 21945-970 Rio de Janeiro/RJ, Brasil

{rps,werner}@cos.ufrj.br, heitor@ufla.br,
{paulojr,afancio}@comp.ufla.br, tonio@dcc.ufla.br

Abstract. *Traditionally, the maintenance activity begins after the software delivery to the user. However, due to complexity and costs in this activity, the maintainability must be incorporated into the artifacts produced during the development process, aiming to develop software with characteristics that make this activity less costly. The paper objective is to show a proposal of maintainability criteria to build implementation model of aspect-oriented software, elaborated from code conventions of Java and AspectJ programming languages and according to ISO/IEC 9126 standard.*

Resumo. *Tradicionalmente, a atividade de manutenção se inicia após a entrega do produto de software ao usuário. Entretanto, devido à complexidade e aos custos dessa atividade, a manutenibilidade deve ser incorporada aos artefatos produzidos durante o processo de desenvolvimento, visando construir software com características que tornem essa atividade menos dispendiosa. O objetivo deste trabalho é apresentar uma proposta de critérios de manutenibilidade para a construção do Modelo de Implementação de produtos de software orientados a aspectos, elaborada a partir de convenções das linguagens de programação Java e AspectJ e sob a luz da norma ISO/IEC 9126.*

1. Introdução

A manutenção assume um papel importante no ciclo de vida do software [Zvegintzov e Parikh, 2005]. No início dos anos 90, muitas organizações alocavam no mínimo 50% de seus recursos financeiros para a manutenção e, na década atual, esse valor tem chegado a 70%, tornando-se um dos desafios da Engenharia de Software [Pressman, 2006]. Além disso, tecnologias como a Orientação a Aspectos (OA) são criadas, buscando reduzir a complexidade da organização de produtos de software. A OA estende a orientação a objetos (OO) com o intuito de tratar o entrelaçamento e o espalhamento de certos requisitos (interesses transversais), visando manutenibilidade e reusabilidade [Elrad *et al.*, 2001]. Apesar das melhorias almejadas, alguns pontos podem dificultar a manutenção (e.g., inversão de dependência e inconsciência) [Santos *et al.*, 2007], o que requer estudos que a relacionem com o Desenvolvimento de Software Orientado a Aspectos (DSOA).

A fim de reduzir a complexidade e os custos de manutenção, a manutenibilidade deve ser incorporada aos artefatos produzidos durante o processo de desenvolvimento, visando construir produtos de software com características que tornem a manutenção menos dispendiosa. O objetivo deste trabalho é apresentar uma proposta de critérios de manutenibilidade para a construção do Modelo de Implementação de produtos de software OA, elaborada a partir de características identificadas em convenções das linguagens de programação Java e AspectJ e sob a luz da norma ISO/IEC 9126. O artigo está dividido da seguinte forma: a seção 2 discorre sobre a fundamentação teórica e lista alguns trabalhos relacionados. A seção 3 apresenta os critérios de manutenibilidade e a seção 4 ilustra sua aplicação. Por fim, são apresentadas as conclusões.

2. Fundamentação Teórica

Zvegintzov e Parikh (2005) apontam a constatação de condições prévias para a realização de modificações como um tópico de pesquisa sobre manutenção. A norma ISO/IEC 9126 (Tecnologia da Informação – Características e Métricas de Qualidade de Software) [ISO Std. 9126, 1991] [ISO Std. 9126, 2001] propõe um modelo de referência na avaliação de produtos de software que define características de qualidade, dentre elas a manutenibilidade. Segundo a norma, a manutenibilidade de produtos de software consiste em um conjunto de atributos que evidencia o esforço para realizar modificações e possui as subcaracterísticas *analisabilidade*, *modificabilidade*, *estabilidade* e *testabilidade*.

A manutenibilidade deve ser considerada em produtos de software "não tradicionais", como aqueles OA. Dentre as tecnologias para DSOA, destaca-se AspectJ, uma extensão da linguagem de programação Java para a implementação de aspectos. Nesse sentido, seria interessante agregar as subcaracterísticas de manutenibilidade aos artefatos que compõem os modelos gerados no DSOA. A Tabela 1 mostra os principais elementos de AspectJ [Kiczales, 2001] [Garcia *et al.*, 2004].

Tabela 1 – Principais elementos de AspectJ

Elemento	Descrição
join points (pontos de junção)	Trechos de código na execução de um produto de software nos quais os aspectos são aplicados. Conceitos abstratos em AspectJ.
pointcuts (conjuntos de pontos de junção)	Declarações responsáveis por selecionar *join points*, ou seja, detectam quais *join points* o aspecto deve interceptar.
advices (adendos ou comportamentos transversais)	Utilizado para implementar interesses transversais – trechos de código executados a cada ocorrência de *join point* descrito no *pointcut*.
introductions ou *inter-type declaration* (declaração intertipos)	Modificam uma classe estruturalmente, acrescentando-lhe novos membros e relacionamentos por meio da cláusula *declare parents*.
aspects (aspectos)	Encapsulam *pointcuts*, *advices* e *introductions* em uma unidade modular de implementação, definidos de maneira semelhante às classes.

Durante as pesquisas realizadas, notou-se a ausência de trabalhos que tratam diretamente a manutenibilidade ao longo do processo de desenvolvimento de software, sobretudo em DSOA. Com o intuito de explorar a utilização de padrões de codificação, Medina (1984) apresenta características da documentação de software que facilitam o entendimento e a usabilidade do código, considerando a documentação "intracódigo" (comentários) e o manual de usuário, e descreve padrões sintáticos que focam na estrutura lógica. No entanto, não existe uma preocupação com questões intrínsecas a um paradigma de programação em especial.

Mais recentemente, Sant'Anna *et al.* (2003) propõem um *framework* para avaliar a manutenibilidade e a reusabilidade de produtos de *software* OA, com métricas e um modelo de qualidade para medir os atributos *separação de interesses*, *acoplamento*, *coesão* e *tamanho*, utilizado na comparação de soluções OA com OO e na avaliação de decisões de projeto em DSOA. Todavia, o trabalho se preocupa mais com a medição e avaliação de propriedades do software do que propriamente com o tratamento da manutenibilidade no DSOA.

Por fim, Costa (2005) propõe critérios e diretrizes para a verificação do Modelo de Análise e para a construção do Modelo de Projeto e do Modelo de Implementação de produtos de software OO, baseados no uso dos padrões de projetos *Singleton* e *State*, nas convenções das linguagens de programação Eiffel, Java e Smalltalk e na norma ISO/IEC 9126. Entretanto, a abordagem se restringe a OO e não provê extensão ou aplicação em DSOA.

3. Critérios de Manutenibilidade para o Modelo de Implementação

O contexto de pesquisa deste trabalho envolve o desenvolvimento de uma abordagem relativa à incorporação da manutenibilidade aos modelos gerados no DSOA, mediante a elaboração de critérios de manutenibilidade que guiem a construção e/ou a avaliação/adaptação dos artefatos de software, além de diretrizes e facilitadores de manutenibilidade que auxiliem a aplicação dos critérios a artefatos de maior nível de abstração. Busca-se reduzir o esforço de transição dos artefatos entre os diferentes níveis de abstração. Nesse sentido, adotou-se uma estratégia *bottom-up*, partindo-se da definição de critérios para a construção do Modelo de Implementação. A justificativa reside no fato de que a principal atividade da manutenção é a alteração de código [Pressman, 2006].

Estendendo o trabalho de Costa (2005), com o diferencial de tratar a manutenibilidade no DSOA, a abordagem apresenta um conjunto de padrões ou exigências que visam guiar a construção de artefatos do Modelo de Implementação, para que produtos de software OA agreguem manutenibilidade. Esses critérios, denominados Critérios de Manutenibilidade para o Modelo de Implementação (*Maintainability Criteria for Implementation Model* – MC_IM), foram elaborados a partir de características das linguagens de programação Java e AspectJ e de pesquisas acerca de métricas, convenções, bom estilo e *guidelines* de programação extraídos de [Kiczales, 2001], [Elrad *et al.*, 2001], [Soares e Borba, 2002], [Camargo e Masiero, 2005], [Piveta *et al.*, 2005] e [Filman *et al.*, 2005], baseando-se na característica de manutenibilidade (ISO/IEC 9126).

Procurou-se manter um conjunto de critérios que cobrisse questões importantes da construção de um Modelo de Implementação OA, a fim de favorecer a sua aplicabilidade e calibragem. Os critérios são referenciados por MC_IM (Tabela 2 e Tabela 3) e organizados em categorias, visando facilitar seu entendimento e referência diante do conjunto de critérios estabelecidos, favorecendo sua ação em conjunto: (i) armazenamento (estado dos objetos em meio persistente); (ii) comentário (informações estratégicas e/ou táticas); (iii) estrutura (disposição e ordem das partes que constituem as entidades da OA); (iv) formato (construção do identificador dos elementos do Modelo de Implementação); (v) localização (posição dos elementos no Modelo de Implementação); (vi) lógica (seqüência coerente, regular e necessária de acontecimentos); e (vii) requisitos (tratamento dos requisitos de forma a identificar aspectos).

Tabela 2 – Critérios de Manutenibilidade para o Modelo de Implementação (I)

Categoria	Critérios de Manutenibilidade para o Modelo de Implementação
Armazenamento	MC_IM 1 – O objeto de uma classe persistente, quando instanciado, tem seu estado armazenado em meio persistente através de aspectos
	MC_IM 2 – O objeto de uma classe persistente, quando sofre alterações no seu estado, tem o seu estado atualizado em meio persistente através de aspectos
	MC_IM 3 – O objeto de uma classe persistente, quando excluído, tem o seu estado excluído do meio persistente através de aspectos
Comentário	MC_IM 4 – Os arquivos físicos contendo aspectos têm comentário introdutório que fornece informações sobre nome do arquivo, conteúdo, autor, aspectos coesos, classes atingidas pelo aspecto, versão, localização e outras informações relevantes para a caracterização do aspecto
	MC_IM 5 – Comentários descritivos antecedem *advices* e métodos auxiliares com funcionalidade complexa
Estrutura	MC_IM 6 – Os elementos componentes estruturais de um aspecto são dispostos de maneira padronizada para facilitar a sua localização
	MC_IM 7 – Os elementos componentes estruturais de um *advice* estão dispostos de maneira a facilitar a sua localização
	MC_IM 8 – *Advices* de um aspecto estão agrupados de acordo com a sua natureza (tipo)
	MC_IM 9 – Comandos de *advices* estão dispostos um em cada linha
	MC_IM 10 – A declaração dos atributos dos aspectos e das variáveis de *advices* é posicionada uma em cada linha
	MC_IM 11 – A declaração de *pointcuts* dos aspectos é posicionada uma em cada linha, de forma que *pointcuts* extensos sejam divididos em *pointcuts* simples, os quais são agrupados em um novo *pointcut* geral formado a partir de *pointcuts* simples
	MC_IM 12 – Uma linha em branco separa blocos de comandos logicamente coesos
	MC_IM 13 – *Advices* possuem poucos parâmetros formais
Formato	MC_IM 14 – O identificador de aspectos, *pointcuts* e *introductions* permite rápido reconhecimento destes elementos
Localização	MC_IM 15 – Os testes realizados no produto de *software* para a sua liberação estão disponíveis em um aspecto
	MC_IM 16 – A localização de retorno em um *advice* está na sua última linha de código
	MC_IM 17 – A responsabilidade de verificar as pré-condições para iniciar a execução de um caso de uso é destinada a um aspecto
	MC_IM 18 – A responsabilidade de verificar as pós-condições para encerrar a execução de um caso de uso é destinada a um aspecto
	MC_IM 19 – A validação de uma nova associação, através da verificação da cardinalidade de um relacionamento, é realizada por um aspecto
	MC_IM 20 – Um aspecto que contém *introduction* não contém nada além desta declaração
Lógica	MC_IM 21 – Arquivos físicos contêm uma, e somente uma, definição de aspecto
	MC_IM 22 – Os atributos de um aspecto são caracterizados como privados ao aspecto
	MC_IM 23 – O método construtor de um aspecto possui duas, e somente duas, funções. Uma delas é instanciar um aspecto (a depender do tipo definido pelo programador) e a outra é atribuir valores iniciais aos atributos do aspecto instanciado (estado inicial)
	MC_IM 24 – *Advices* identificados são simples e com linhas de código suficientes para ter apenas uma função/tarefa (coesão)
	MC_IM 25 – É declarada precedência entre aspectos quando mais de um aspecto apresentam *pointcuts* que interceptam os mesmos *join points*
	MC_IM 26 – *Pointcuts* ou *advices* duplicados são refatorados
	MC_IM 27 – Cada aspecto deve tratar um interesse específico
	MC_IM 28 – Aspectos com poucas responsabilidades ou com generalidade especulativa não devem ser implementados
	MC_IM 29 – *Pointcuts* são nomeados
	MC_IM 30 – *Pointcuts* declarados em classes devem ser fáceis de migrar para aspectos
	MC_IM 31 – Métodos abstratos não devem ser implementados via *introductions*

Tabela 3 - Critérios de Manutenibilidade para o Modelo de Implementação (II)

Categoria	Critérios de Manutenibilidade para o Modelo de Implementação
Requisitos	MC_IM 32 – Requisitos não funcionais como distribuição, controle de concorrência, tratamento de exceções, depuração, sincronização de objetos concorrentes, coordenação de múltiplos objetos, serialização, atomicidade, replicação, segurança, visualização, *logging*, *tracing* e tolerância a falhas são implementados como aspectos
	MC_IM 33 – Requisitos funcionais referentes a regras de negócio espalhadas e/ou entrelaçadas com a funcionalidade básica do produto de *software* são implementados como aspectos

Os critérios apresentam a seguinte estrutura: (i) **categoria**: organiza os MC_IMs conforme sua natureza; (ii) **critério**: identifica o MC_IM com número e descrição; (iii) **justificativa**: justifica o uso do MC_IM; (iv) **subcaracterística(s)**: lista a(s) subcaracterística(s) de manutenibilidade definida(s) na norma ISO/IEC 9126 as quais o MC_IM visa melhorar pela aplicação do critério. A Figura 1 mostra um exemplo de MC_IM. A relação detalhada dos MC_IMs está apresentada [Santos, 2007]. Além disso, a utilização dos MC_IMs necessita de um treinamento: (i) revisão das estruturas de OA em AspectJ (caso necessário); (ii) apresentação da manutenibilidade (norma ISO/IEC 9126); e (iii) discussão dos MC_IMs. Pode-se ter duas situações para a utilização dos MC_IMs: (i) construção do Modelo de Implementação (critérios guiam as decisões de codificação); e (ii) avaliação/adaptação de um Modelo de Implementação existente (verificação do código, a fim de adequá-lo aos padrões ou exigências dos critérios).

MC_IM 29 – *Pointcuts* são nomeados.

Justificativa: Favorece o entendimento e a reutilização de um *pointcut* para a definição de outros *pointcuts* e *advices*. A definição de um *pointcut* não é usada diretamente em um *advice*, visando não ocultar as intenções deste e evitar redundância de código, diminuição da legibilidade e prejuízo aos benefícios da OA. Caso contrário, pode se tornar necessário recorrer à descrição contida em *pointcuts* anônimos (não-nomeados) para obtenção de uma idéia dos pontos afetados pelo *advice*.
Subcaracterística(s): *Analisabilidade* e *Modificabilidade*.

Figura 1 – Exemplo de MC_IM

4. Exemplo de Aplicação: Sistema do Domínio Bancário

A fim de verificar a aplicabilidade dos MC_IMs, decidiu-se por utilizá-los para guiar a construção do Modelo de Implementação de um Sistema do Domínio Bancário (SDB). O SDB gerencia operações requeridas por uma agência bancária: efetuar *login* e *logoff*, cadastrar e remover clientes, alterar senha no sistema, ver saldo, realizar transferência e efetuar depósito e saque. Os usuários são classificados em administrador e cliente. O SDB possui os seguintes aspectos: `APersistencia` (persistência), `ATransacoes` e seu sub-aspecto `ATransacoesBancarias` (controle de transação), `ALogging` (histórico de acessos), `AIUsuario` (*declare parents*), `ATracing` (controle de fluxo de execução), `APreEPosCondicoes` (suporte à verificação de pré e pós-condições), `AAutenticacao` (autenticação de usuário) e `AExcecoes` (tratamento de exceções). A seguir, são analisados trechos de aspectos do SDB, exemplificando a aplicação de alguns MC_IMs. Para evitar sobrecarga, mesmo que alguns critérios estejam presentes em vários trechos, eles serão destacados em apenas um dos exemplos.

A coerência entre objetos em memória e no meio persistente deve ser respeitada. Como a implementação da persistência encontra-se normalmente espalhada pelo código, encapsulá-la em aspectos é uma boa prática. Os MC_IM 1, MC_IM 2 e MC_IM 3 são verificados no aspecto `APersistencia` (Figura 2) pelas operações *abrir conta*, *alte-*

rar senha (destacada no exemplo) e *excluir clientes*. Além disso, para melhorar o entendimento, atende-se aos MC_IM 4 e MC_IM 6, através do comentário introdutório e da disposição interna dos elementos do aspecto (sugerida pelo MC_IM 6), respectivamente. *Advices* também possuem poucos parâmetros formais (MC_IM 13) e existe um construtor definido conforme o MC_IM 23. Os atributos de aspecto estão posicionados um por linha (MC_IM 10) e são declarados como *private* (MC_IM 22).

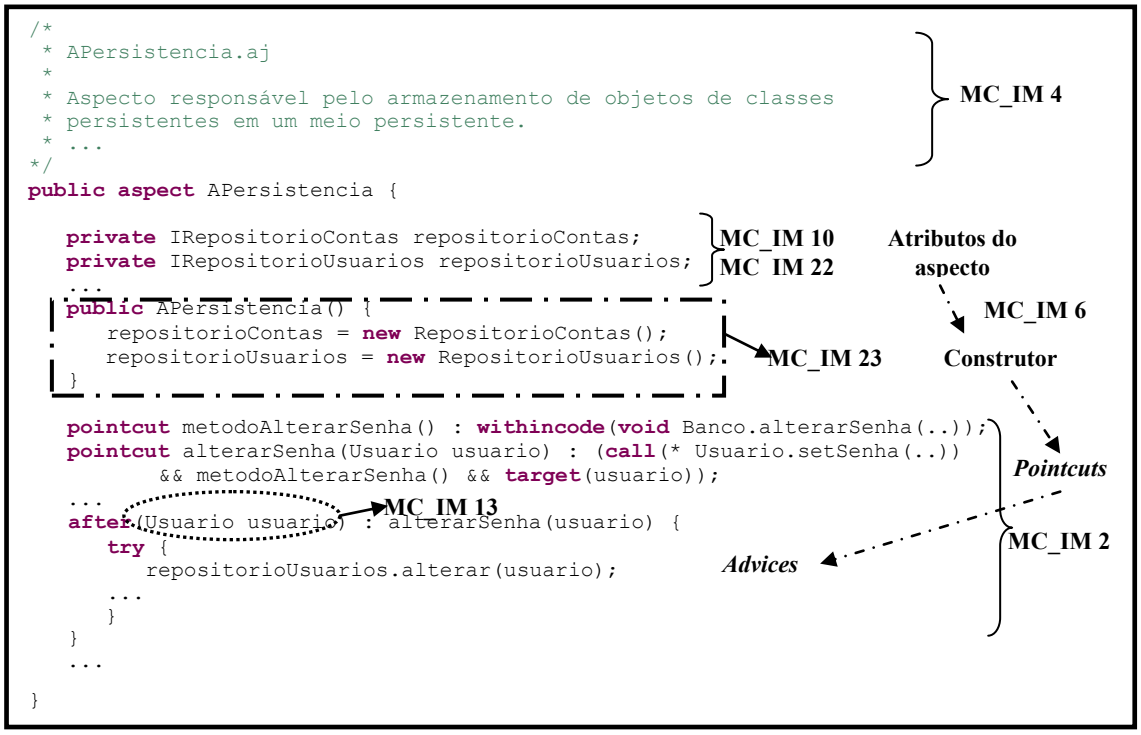

Figura 2 – Aspecto *APersistencia* pertencente ao arquivo *APersistencia.aj*

O encapsulamento de testes em aspectos ocorre no aspecto `ATracing` (Figura 3), por meio do método `testesTracing()` (MC_IM 15) e, ao ser o único aspecto do arquivo `ATracing.aj`, atende ao MC_IM 21. Como *pointcuts* do aspecto atuam sobre *join points* nos quais *pointcuts* de outros aspectos podem atuar, deve-se utilizar a declaração de precedência (MC_IM 25). Utilizou-se MC_IM 26 devido à seguinte situação: ao invés de utilizar dois métodos de rastreamento, `traceEntry(String str)` e `traceExit(String str)`, que conteriam a mesma seqüência de instruções invocadas pelos *advices before* e *after*, refatorou-se os códigos dos métodos, eliminando-os e criando o método `trace(String str)`. O aspecto não é complexo (MC_IM 27) e acata ao MC_IM 32 (requisito não-funcional). Durante a construção do SDB, esse aspecto seria *abstract*. Entretanto, não havia necessidade de estender o aspecto `ATracing` apenas para definir os pontos afetados; implementou-se isso em um aspecto apenas (MC_IM 28). Ao agrupar e ordenar *advices* (sugestão do MC_IM 8), respeita-se o MC_IM 8, e atende-se ao MC_IM 11, por meio da divisão de *pointcuts* complexos em *pointcuts* simples, posicionados um por linha. Além disso, o nome atribuído ao aspecto (iniciado por A) e aos *pointcuts* (iniciado por letra minúscula) segue um padrão (MC_IM 14) e *pointcuts* são nomeados, facilitando sua reutilização (MC_IM 29).

Em [Santos, 2007], mostra-se ainda que o aspecto `APreEPosCondicoes` representa a aplicação do MC_IM 17 e do MC_IM 18, e `AIUsuario`, devido à declaração

inter-tipos, utiliza o MC_IM 20. Ao todo, aplicou-se 29 critérios, cobrindo 88% dos MC_IMs. Apenas os MC_IM 19, MC_IM 30, MC_IM 31 e MC_IM 33 não puderam ser aplicados, devido ao escopo do SDB. Entretanto, isso está previsto para os próximos exemplos de aplicação.

Figura 3 – Aspecto *ATracing* pertencente ao arquivo *ATracing.aj*

5. Conclusão

A atividade de manutenção é inerente a produtos de software e, dada uma quantidade elevada de códigos reparados e atualizados, considera-se essa atividade a mais dispendiosa do ciclo de vida [Pressman, 2006]. Embora as novas metodologias busquem reduzir a complexidade da organização de produtos de software, elas não necessariamente apresentam uma influência positiva na manutenção [Zvegintzov e Parikh, 2005]. Dessa forma, foram propostos um conjunto de critérios de manutenibilidade para guiar a construção do Modelo de Implementação de produtos de software OA (MC_IMs), parte de uma abordagem relativa à definição de critérios, diretrizes e facilitadores de manutenibilidade para cada um dos modelos gerados no DSOA.

Para verificar a aplicabilidade dos MC_IMs, construiu-se um sistema de domínio bancário. Os critérios propiciaram a organização das informações do Modelo de Implementação ao estabelecerem padrões de formato, estrutura, lógica, requisitos e localização. Isso contribui para facilitar a atividade de manutenção ao definir padrões que atuam sobre a legibilidade de código, visando melhorias. Experimentos e estudos de caso, referentes à avaliação/adaptação de Modelos de Implementação OA existentes, devem ser realizados, abrangendo outros domínios de aplicação, bem como a avaliação da efetivi-

dade dos MC_IMs estabelecidos e a sua aderência a outras linguagens OA. A partir disso, novos critérios poderão ser necessários e, nesse caso, adicionados ao conjunto dos MC_IMs. Critérios, diretrizes e facilitadores de manutenibilidade para Modelos de Projeto e de Análise também devem ser construídos, mantendo-se a rastreabilidade entre os critérios definidos para os três modelos. Assim, um ambiente para apoiar a abordagem poderá ser construído visando sua integração ao processo de desenvolvimento.

Agradecimentos

Os autores agradecem ao CNPq pelo apoio financeiro na realização deste trabalho.

Referências

Camargo, V. V.; Masiero, P. C. (2005) "Frameworks Orientados a Aspectos", In: XIX SBES, Uberlândia/MG, p. 200-216.

Costa, H. A. X. (2005) "Critérios e Diretrizes de Manutenibilidade para a Construção do Modelo de Projeto Orientado a Objetos". Tese (Doutorado). Escola Politécnica – Universidade de São Paulo, São Paulo/SP, 199p.

Elrad, T.; Kiczales, G.; Aksit, M.; Lieberher, K.; Ossher, H. (2001) "Discussing Aspects of AOP". Communications of the ACM, v. 44, n. 10, p. 33-38.

Filman, R. E.; Elrad, T.; Clarke, S.; Aksit, M. (2005) "Aspect-Oriented Software Development". Addilson Wesley – Pearson Education, 755p.

Garcia, A.; Chavez, C.; Soares, S.; Piveta, E.; Penteado, R.; Camargo, V. V.; Fernandes, F. (2004) "Relatório do 1º WASP", 10p.

ISO Std. 9126. (1991) "Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their use". International Organization for Standardization.

ISO Std. 9126. (2001) "Software Enginnering – Product Quality Part 1: Quality Model". International Organization for Standardization.

Kiczales, G. (2001) "Aspect-Oriented Programming with AspectJ" (Tutorial), In: OOPSLA'2001, Tampa, Flórida, EUA.

Medina, E. A. (1984) "Some Aspects of Software Documentation", In: Proc. of the 3rd SIGDOC, Cidade do México, México, p. 57-59.

Piveta, E. K.; Hecht, M.; Pimenta, M. S.; Price, R. T. (2005) "*Bad Smells* em Sistemas Orientados a Aspectos", In: XIX SBES, Uberlândia/MG, p. 184-199.

Pressman, R. S. (2006) "Engenharia de Software", 6ª ed. McGraw-Hill.

Sant'Anna, C. N.; Garcia, A.; Chavez, C.; Lucena, C. J. P.; Staa, A. (2003) "On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework", In: XVII SBES, Manaus/AM, p. 19-34.

Santos, R. P. (2007) "Critérios de Manutenibilidade para Construção e Avaliação de Produtos de Software Orientados a Aspectos". Monografia. DCC/UFLA, Lavras/MG, 92p.

Santos, R. P.; Silva, M. C. O.; Werner, C. M. L.; Murta, L. G. P. (2007) "Questões e Desafios de Orientação a Aspectos no Contexto da Reutilização de Software", In: Proc. of the I LA-WASP, João Pessoa/PB, p. 185-185.

Soares, S.; Borba, P. (2002) "AspectJ – Programação Orientada a Aspectos em Java" (Tutorial), In: VI SBLP, Rio de Janeiro/RJ, p. 39-55.

Zvegintzov, N.; Parikh, G. (2005) "60 years of Software Maintenance: Lessons Learned", In: Proc. of the 21st ICSM, Budapeste, Hungria, p. 726- 727.