Uma Abordagem Quantitativa para Desenvolvimento de Software Orientado a Aspectos

Eduardo Figueiredo[1, 2, ♣], Carlos Lucena[1], Alessandro Garcia[2]

[1]Departamento de Informática, PUC-Rio, Brasil

[2]Computing Department, Lancaster University, UK

{e.figueiredo, a.garcia}@lancaster.ac.uk, lucena@inf.puc-rio.br

Abstract. *Design assessment in software engineering has strictly based on module coupling, cohesion, and size. With the advent of aspect-oriented (AO) programming, this traditional approach has led to a number of assessment breakdowns since it does not consider new AO abstractions (e.g. concerns) and composition mechanisms. This paper: (i) presents AO metrics which consider concerns as explicit abstractions, (ii) proposes a systematic assessment method, and (iii) investigates the hypothesis that heuristic rules enhances the assessment process. Our proposal is supported by a measurement tool. We have evaluated our approach through its application to six software systems from different domains and with distinct degrees of complexity. The analysis was based on the metrics and heuristics support for finding the presence of design flaws relative to certain crosscutting concerns.*

Resumo. *Avaliação de sistemas em engenharia de software tem sido baseada estritamente em acoplamento, coesão e tamanho dos módulos. Com o surgimento da programação orientada a aspectos (OA), esta tradicional abordagem de avaliação tem levado a ocorrência de muitos falsos positivos e falsos negativos nos resultados por não considerar novas abstrações OA, como interesses, e novos mecanismos de composição. Neste contexto, este artigo: (i) apresenta métricas OA que consideram interesses como abstrações explícitas nas medições, (ii) propõe um método sistemático de avaliação, e (iii) investiga a hipótese de regras heurísticas auxiliarem o processo de avaliação de software. Nossa proposta é suportada por uma ferramenta de medição. Adicionalmente, avaliamos a abordagem heurística através de sua aplicação em seis sistemas, pertencentes a diferentes domínios e com distintos graus de complexidade. A análise tem por base o quanto as métricas e heurísticas suportam o desenvolvedor na tarefa de encontrar problemas de projeto relativos a certos interesses transversais.*

1. Introdução

O desenvolvimento de software orientado a aspectos (DSOA) [Kiczales et al., 1997] é um paradigma recente que provê novas abstrações e mecanismos para suportar a

♣ Mestrado e doutorado (atual) suportados pela CAPES.

modularidade de interesses transversais. Entretanto, atingir um software orientado a aspectos (OA) de elevada qualidade não é trivial devido aos novos mecanismos de composição existente neste paradigma, tais como adendos, pontos de corte e declarações inter-tipos [Kiczales et al., 1997]. Neste sentido, o uso inapropriado de abstrações e mecanismos OA pode potencialmente levar a violação de importantes princípios de projetos, tais como baixo acoplamento, alta coesão, baixa complexidade dos módulos, modularidade de interesses transversais em aspectos, entre outros. Estudos recentes [Garcia et al., 2005, 2006] mostram que a separação de certos interesses em aspectos podem trazer mais problemas do que vantagens. Mesmo a aspectização de tradicionais interesses transversais, tais como tratamento de exceções [Filho et al., 2006] e o padrão Observer [Garcia et al., 2006], pode causar impacto negativo na modularidade do sistema em certas circunstâncias.

Tais problemas de modularidade e outros problemas de projeto, como *bad smells* [Fowler, 1999], não são facilmente detectáveis e uma análise *ad hoc* de grandes sistemas é normalmente cara e trabalhosa. *Bad smell* é definido como uma característica estrutural do projeto que denota um desvio de critérios de qualidade do sistema [Marinescu, 2004]. Abordagens de avaliação existentes falham ao encontrar problemas em projetos OA por negligenciar as novas abstrações (e.g. interesses) e mecanismos de composição deste paradigma. Portanto, tais abordagens tendem a erroneamente reportar problemas (falsos positivos) e falham por não identificar reais problemas (falsos negativos) [Figueiredo et al., 2007]. Até o momento, não é conhecido nenhum método de avaliação que considere os benefícios da separação de interesses em relação aos potenciais danos causados a tradicionais atributos de modularidade. Além disso, ferramentas para DSOA se restringem a algumas etapas do processo de avaliação, tais como medição [Ceccato and Tonella, 2004] e mineração de aspectos [Robillard and Murphy, 2002]. No mais, tais ferramentas não se integram a um método de avaliação.

Para cobrir a lacuna deixada por trabalhos existentes, este artigo propõe um método sistemático para avaliação quantitativa de artefatos de projeto e implementação produzidos no DSOA. O método tem a distinta característica de considerar interesses como abstrações explícitas no processo de avaliação. Além disso, ele é baseado em um conjunto de métricas e regras heurísticas sensíveis aos interesses do sistema. As métricas contribuem com informações quantitativas enquanto as regras complementam o método fornecendo algum raciocínio qualitativo. No mais, uma ferramenta de suporte é apresentada para facilitar a análise de projetos complexos. Todos os elementos propostos são sistematicamente avaliados através de suas aplicações em seis estudos de casos que incluem uma grande variedade de interesses transversais. Estes estudos têm mostrado que o método heurístico é capaz de encontrar problemas não triviais relacionados à existência de interesses transversais.

O restante do artigo é organizado da seguinte forma. A Seção 2 introduz algumas métricas existentes para DSOA e propõe novas métricas. A Seção 3 apresenta um método sistemático de avaliação composto de métricas (Seção 2) e de compreensivas regras heurísticas que são discutidas detalhadamente na Seção 4. Uma ferramenta de medição que suporta o método é apresentada na Seção 5. A seção 6 faz uma avaliação do método heurístico proposto enquanto a Seção 7 discute brevemente as limitações de trabalhos relacionados, destacando as interfaces deste artigo com a literatura existente. Concluímos com as principais contribuições desta pesquisa na Seção 8.

2. Avaliação de Software Orientado a Aspectos

Esta seção descreve a natureza de métricas OA existentes (Seção 2.1) e propõe uma nova categoria de métricas OA que chamamos orientadas a interesses (Seção 2.2).

2.1. Métricas OA Existentes

Com a expansão do DSOA, várias métricas OA têm sido definidas para quantificar atributos internos de software como acoplamento, coesão e tamanho. A maioria é baseada em extensões de métricas tradicionais como, por exemplo, Linhas de Código (LOC) e Tamanho do Vocabulário (VS) [Garcia et al., 2005]. Esta última conta o número de componentes (classes, interfaces e aspectos) que um sistema possui. Existem também outras métricas OA que estendem definições de métricas bem conhecidas no contexto de orientação a objetos (OO), como as de Chidamber e Kemerer [1994]. Exemplos de tais métricas são: Número de Operações (NOO) [Figueiredo e Staa, 2005], Número de Atributos (NOA) e Perda de Coesão em Operações (LCOO) [Garcia et al., 2006]. Uma característica comum de todas estas métricas é que elas medem elementos que possuem limites explicitamente definidos nas linguagens de modelagem e programação, tais como classes, interfaces, aspectos, operações, etc. Por outro lado, problemas típicos de modularidade estão relacionados a inadequada separação de interesses que atravessam diferentes elementos do projeto [Sant'Anna et al., 2007].

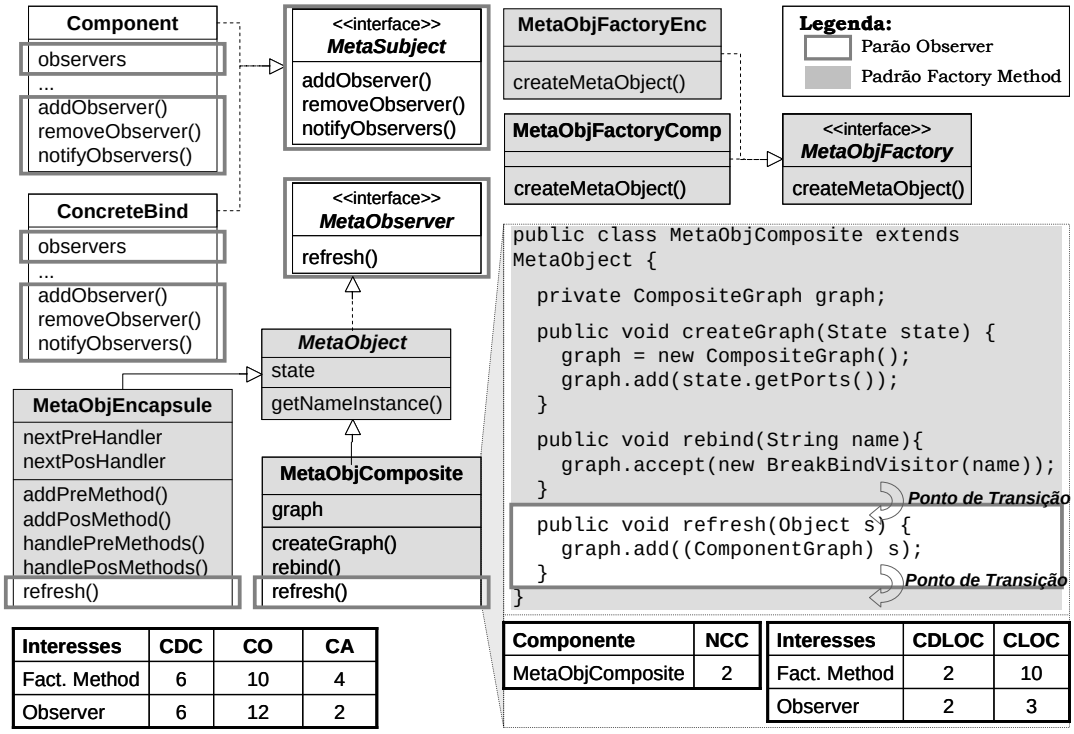
2.2. Métricas Orientadas a Interesses

Propomos nesta seção novas métricas [Figueiredo e Staa, 2005] [Figueiredo et al., 2007] que capturam características do projeto, mesmo que estas não estejam propriamente definidas nos limites das linguagens de projeto e implementação. A Tabela 1 lista seis destas métricas que chamamos de orientadas a interesses e provê uma curta descrição para cada uma delas. As métricas propostas complementam as existentes por explicitamente promover interesses a abstrações plausíveis para medições. Este novo conjunto de métricas apresenta alternativas para quantificar a difusão de um interesse pelos elementos sintáticos, como componentes (CDC) e linhas de código (CDLOC). Ele também permite avaliar como um particular interesse afeta tradicionais atributos de qualidade, como a complexidade dos componentes (NCC). Em adição, as métricas avaliam a dedicação de cada componente (ou do sistema) a um interesse específico em termos dos seus atributos (CA), operações (CO) e linhas de código (CLOC).

Tabela 1. Exemplos de métricas orientadas a interesses [Figueiredo et al., 2007]

Métrica	Descrição
Difusão do Interesse por Componentes (CDC)	Conta o número de classes e aspectos que possuem como propósito a realização total ou parcial de um interesse.
Difusão do Interesse por Linhas de Código (CDLOC)	Conta o número de "pontos de transição" entre um interesse específico e os demais interesses do sistema. Pontos de transição são os locais no código em que existe uma junção de interesses (veja Figura 1).
Número de Interesses por Componente (NCC)	Conta o número de interesses realizado por cada classe, interface ou aspecto do sistema.
Atributos do Interesse (CA)	Conta o número de atributos de cada componente que tem como propósito principal a realização de um interesse específico.
Operações do Interesse (CO)	Conta o número de métodos, construtores e adendos de cada componente que tem como propósito principal a realização de um interesse específico.
Linhas de Código do Interesse (CLOC)	Conta o número de linhas de código de cada componente que tem como propósito principal a realização de um interesse específico.

O diagrama de classes da Figura 1 é utilizado para exemplificar as métricas orientadas a interesses. Este diagrama representa uma pequena parte do projeto OO de um de nossos estudos de caso, *middleware* OpenOrb (Seção 6), destacando o código da classe `MetaObjComposite`. Além disso, são sombreados os elementos que implementam os interesses relativos aos padrões Factory Method e Observer [Gamma et al., 1995], avaliados neste sistema. Este sombreamento é necessário para que as métricas orientadas a interesses possam ser calculadas. Assim, a Figura 1 mostra que existe código relacionado ao padrão Factory Method em seis componentes (métrica CDC), dez operações (métrica CO) e quatro atributos (métrica CA). Em relação à classe `MetaObjComposite`, esta possui elementos que realizam os dois interesses avaliados (métrica NCC), ou seja, o método `refresh()` implementa o padrão Observer enquanto o restante da classe é atribuído ao padrão Factory Method. No mais, existem dois pontos de transição entre o dois interesses nesta classe (métrica CDLOC) e dez linhas de código são dedicadas ao padrão Factory Method (métrica CLOC).



Interesses	CDC	CO	CA
Fact. Method	6	10	4
Observer	6	12	2

Componente	NCC
MetaObjComposite	2

Interesses	CDLOC	CLOC
Fact. Method	2	10
Observer	2	3

Figura 1. Diagrama de classes ilustrando as métricas orientadas a interesses

3. Um Método Sistemático de Avaliação

Avaliar apropriadamente os relevantes atributos de um sistema é pré-requisito para conquistar um software de alta qualidade. Além disso, explorar tais atributos proporciona uma avaliação ampla que considera as vantagens e desvantagens de diferentes soluções alternativas. Portanto, propomos nesta seção um método iterativo, baseado no princípio de que não somente a separação de interesses deve ser o fator determinante na análise OA, mas também outros atributos essenciais da engenharia de software, como acoplamento entre componentes, coesão e tamanho. O método proposto [Figueiredo et al., 2005] é estruturado em quatro passos (Figura 2) que avaliam: (1) como os interesses são modularizados nos artefatos de software, tais como classes,

aspectos, operações e atributos; (2) a complexidade interna dos componentes; (3) a modularidade da perspectiva do sistema, ou seja, no que diz respeito aos relacionamentos entre componentes; e (4) o sistema em termos de problemas de projeto mais específicos como *bad smells* [Fowler, 1999].

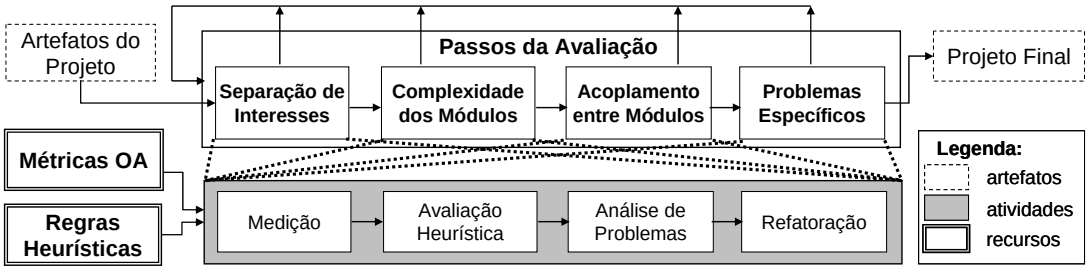


Figura 2. Método de avaliação orientado a interesses [Figueiredo et al., 2005]

A avaliação da separação de interesses no Passo 1, assim como os demais passos do método, é composta de quatro atividades (Figura 2): Medição, Avaliação Heurística, Análise de Problemas e Refatoração. Além disso, para executar estas atividades, o método de avaliação depende de dois tipos de recursos, Métricas AO (Seção 2) e Regras Heurísticas (Seção 4). Na atividade de Medição, as métricas OA são aplicadas para coletar dados sobre os atributos de qualidade do software. Em seguida, é feita a Avaliação Heurística dos números pela aplicação de regras, e estas geram alertas de possíveis problemas. Na atividade seguinte, Análise de Problemas, o desenvolvedor julga os alertas gerados pelas regras com o intuito de identificar quais são os reais problemas. Finalmente, a partir destes problemas é possível decidir pela Refatoração ou não do sistema em questão.

4. Regras Heurísticas Orientadas a Interesses

Além das métricas (Seção 2.2) e do método de avaliação (Seção 3), propomos a utilização de regras heurísticas como mecanismo de suporte a análise de modularidade orientada a interesses [Figueiredo et al., 2005, 2007]. Tais regras são direcionadas a fornecer informações relevantes sobre os interesses do sistema de tal forma que permita ao desenvolvedor utilizar meios mais abstratos ao invés de trabalhar diretamente com os números resultantes das medições. Assim, as expressões heurísticas englobam conhecimento de como os interesses se realizam em projetos OO ou OA. As heurísticas propostas nesta seção não somente detectam problemas de projetos como também capturam os elementos sintáticos que apresentem valores anormais para um determinado conjunto de métricas. Desta forma, os alertas gerados pelas regras oferecem informações adicionais que ajudam o desenvolvedor a se concentrar em certas partes e interesses do projeto potencialmente mais problemáticos.

As regras suportam a atividade de Avaliação Heurística do método (Figura 2) classificando-se em três maiores grupos de acordo com os passos do método. O primeiro grupo define heurísticas para avaliar o espalhamento de interesses (Passo 1). Uma vez que o Passo 2 (Complexidade Interna dos Módulos) e o Passo 3 (Acoplamento entre Módulos) estão relacionados a atributos de modularidade, definimos o segundo grupo de regras para avaliar de forma unificada estes atributos. O terceiro grupo de heurísticas detecta problemas mais específicos no projeto de tal forma a suportar o Passo 4 do método. As três categorias de regras heurísticas são discutidas e

exemplificadas nas seções 4.1, 4.2 e 4.3, respectivamente. Por restrições de espaço, apenas algumas regras são apresentadas em cada categoria e heurísticas adicionais podem ser encontradas em [Figueiredo et al., 2005, 2007] ou na página do projeto[1].

4.1. Heurísticas para Espalhamento de Interesses

Idealmente, cada componente do sistema deveria implementar um único interesse que corresponde ao seu coerente e bem definido conjunto de responsabilidades. Apesar de este ser o cenário ideal, componentes normalmente agregam múltiplos interesses, ou seja, um *interesse primário* e outros interesses "intrusos" que são chamados *transversais*. Interesses transversais incluem responsabilidades que não estão relacionadas ao propósito principal do componente e, portanto, o projeto do sistema se beneficiaria pela extração de tais interesses transversais para novos componentes. Neste contexto, as regras do primeiro grupo que compõe o método, chamadas Heurísticas para Espalhamento de Interesses, classificam a forma pelo qual um interesse se manifesta nos elementos do software. Elas têm o objetivo de identificar interesses transversais e, para isso, combinam informações de diferentes métricas de tamanho, tais como VS, NOA, NOO e LOC (Seção 2.1), com informações obtidas por métricas orientadas a interesses, tais como CDC, CDLOC, CA, CO e CLOC (Seção 2.2).

```
R01: se (CDLOC > 0) então INTERESSE é ENTRELAÇADO

R02: se (CDC/VS > 0.5) para um INTERESSE ENTRELAÇADO
então INTERESSE ENTRELAÇADO é ESPALHADO

R03: se (CA/NOA < 0.5) e (CO/NOO < 0.5) para pelo menos um
    componente com INTERESSE ESPALHADO
então INTERESSE ESPALHADO é TRANSVERSAL NO PROJETO

R04: se (CLOC/LOC < 0.5) para pelo menos um componente com
    INTERESSE TRANSVERSAL NO PROJETO
então INTERESSE é TRANSVERSAL NA IMPLEMENTAÇÃO
```

As regras R01 a R04 definidas acima representam quarto exemplos de heurísticas para espalhamento de interesses. A primeira regra, R01, utiliza a métrica CDLOC para classificar o interesse em *entrelaçado* ou *primário*. O valor zero para esta métrica significa que em todos os componentes nos quais o interesse se encontra só existe tal interesse e, portanto, ele é o interesse primário destes componentes. Por outro lado, um valor maior que zero para a métrica CDLOC indica entrelaçamento de interesses em pelo menos um dos componentes do sistema (e.g. classe `MetaObjComposite` na Figura 1). De forma semelhante, a regra R02 é usada para verificar quando um interesse, além de entrelaçado, também se espalha sobre muitos componentes. Esta regra heurística utiliza as métrica CDC e VS para determinar a percentagem de componentes do sistema afetados pelo interesse. Se este valor for acima de 50%, o interesse avaliado é classificado como *espalhado*.

Apesar de espalhamento e entrelaçamento de interesses serem características indesejáveis ao sistema, o caso mais grave de problema ocorre quando o interesse é intruso em um ou mais componentes, isto é, interesse transversal. Assim, a regra R03 usa as métricas orientadas a interesses CA e CO bem como duas métricas de tamanho, NOA e NOO, para calcular a dedicação dos componentes ao interesse avaliado. Ou seja,

[1] http://www.lancs.ac.uk/postgrad/figueire/heuristics/

esta regra identifica as percentagens de operações e atributos de cada componente do sistema que implementam o interesse e se estes valores forem menores que 50% em pelo menos um componente, o interesse é classificado como *transversal no projeto*. O raciocínio é que componentes dedicam apenas uma pequena quantidade de operações e atributos aos interesses transversais.

O último exemplo de regra desta categoria, R04, avalia mais detalhadamente (em termos de linhas de código) o interesse previamente classificado como transversal no projeto. Esta regra utiliza as métricas CLOC e LOC para calcular a percentagem de linhas de código que os componentes dedicam à realização do interesse em questão. Se este valor é inferior a 50% em pelo menos um destes componentes, o interesse é classificado como *transversal na implementação*. O raciocínio da regra R04 é semelhante ao de R03, entretanto, decidimos separá-las em duas heurísticas porque enquanto a análise de atributos e operações pode ser efetuada no nível de projeto, a análise em termos de linha de código requer artefatos de implementação.

4.2. Heurísticas para Modularidade de Componentes

O segundo grupo de regras heurísticas analisa os efeitos causados por interesses transversais em atributos de modularidade dos componentes, tais como acoplamento e coesão. Estas regras utilizam tanto métricas orientadas a interesses quanto métricas de acoplamento, coesão e tamanho em sua formação. Por exemplo, a regra a seguir representa uma heurística que verifica a coesão dos componentes nos quais existem interesses transversais. Note que este segundo conjunto de regras somente é aplicável após as heurísticas para espalhamento de interesses (Seção 4.1) terem identificado os componentes que possuem interesses transversais.

```
R05: se (LCOO > (LCOOsistema/VS)) e (NCC > 1)
então COMPONENTE com INTERESSE TRANSVERSAL apresenta PERDA DE COESÃO
```

O exemplo de regra R05 classifica os componentes que apresentem perda de coesão a partir de dois atributos medidos: (i) o valor tradicional de coesão baseado em LCOO e (ii) uma nova dimensão de coesão orientada a interesses baseada na métrica NCC. O primeiro atributo compara o valor de LCOO do componente com a média de coesão dos demais componentes do sistema (LCOOsistema/VS). O segundo atributo, por outro lado, aplica uma métrica de coesão orientada a interesses para verificar se o componente agrega mais de um conjunto de responsabilidades. Assim, o valor de NCC maior que 1 significa que o componente realiza mais de um interesse e, portanto, desempenha funções que não estão necessariamente relacionadas.

4.3. Heurísticas para *Bad Smells*

O último grupo de heurísticas que compõe o método aplica um raciocínio orientado a interesses para detecção de problemas de projeto mais específicos. Dois exemplos bem conhecidos de *bad smells* [Fowler, 1999], God Class e Shotgun Surgery, são utilizados para ilustrar heurísticas desta categoria. O primeiro *bad smell*, God Class, refere-se a um componente que desempenha a maior parte do trabalho no sistema, delegando apenas pequenos conjuntos de tarefas para classes triviais e usando dados destas classes. O outro *bad smell*, Shotgun Surgery, ocorre quando uma alteração em alguma característica (ou interesse) do sistema implica em muitas outras "pequenas" alterações em diferentes locais [Fowler, 1999]. Um exemplo de Shotgun Surgery geralmente

encontrado em modelagens puramente OO é o mecanismo de persistência. Caso não haja uma preocupação em isolar o código relacionado à persistência, este interesse se espalha pelos diversos componentes da modelagem OO e se entrelaça a outros interesses. Em conseqüência, sérios problemas são enfrentados com a manutenção do mecanismo de persistência como, por exemplo, alteração do banco de dados.

```
R06: se (NCC > 2) e (LCOO > (LCOOsistema/VS) e (NOA > 1) e
(NOO > (NOOsistema/VS)) )
então COMPONENTE com INTERESSE TRANSVERSAL é GOD CLASS

R07: se (CDC > 10) e (CDLOC > 20)
então INTERESSE TRANSVERSAL é SHOTGUN SURGERY
```

A regra heurística R06 procura identificar componentes (classes ou aspectos) que podem ser classificados como Good Class. Esta regra verifica a complexidade dos componentes em termos de sua perda de coesão (LCOO) e do número de atributos (NOA) e operações (NOO). Além disso, R06 considera também a quantidade de interesses realizados pelo componente (NCC), uma vez que realizar muitos interesses pode ser um sinal de responsabilidade excessiva do componente. Diferentemente de R06 que olha para os componentes, a regra heurística R07 verifica, dentre os interesses do sistema, aqueles que são os possíveis candidatos a Shotgun Surgery. Esta regra classifica como Shotgun os interesses que afetam muitos componentes (CDC>10) e se encontram altamente entrelaçados a outros interesses (CDLOC>20). Em outras palavras, um interesse apresenta este *bad smell* quando qualquer alteração nele pode resultar em um efeito cascata que impacta muitos artefatos do software.

Uma parte sensível das regras heurísticas apresentadas neste artigo é a definição de valores limites. O valor limite é um parâmetro importante para a eficácia da regra, mas encontrar o valor ideal não é um problema novo e intrinsecamente caracteriza qualquer abordagem baseada em medição [Marinescu, 2004]. Duas estratégias são usadas na definição de valores limites em nossos estudos: boas sugestões encontradas na literatura e configuração empírica. No primeiro caso, a escolha do valor é guiada por experiências semelhantes e acertos apontados em trabalhos relacionados. Por outro lado, quando estes acertos não são conhecidos ou não estão disponíveis, os valores podem ser configurados empiricamente de acordo com as características específicas do projeto (e.g. tamanho) e da equipe de desenvolvimento. A configuração empírica requer suporte automatizado, pois, utilizando uma ferramenta o desenvolvedor pode definir um valor limite e verificar imediatamente os efeitos de sua ação. Na próxima seção apresentamos uma ferramenta que suporta esta questão.

5. Ferramenta de Suporte

Uma das maiores preocupações em se propor uma nova abordagem de avaliação é seu suporte ferramental. Para lidar com esta limitação, o método orientado a interesses proposto neste artigo é suportado pela ferramenta de avaliação de programas Java e AspectJ chamada AJATO[2] [Figueiredo, Garcia and Lucena, 2006]. Os dois principais objetivos desta ferramenta são: (i) calcular as métricas tradicionais e orientadas a interesses (Seção 2) e (ii) aplicar as regras heurísticas orientadas a interesses (Seção 4).

[2] Acrônimo em inglês para AspectJ Assessment Tool
http://www.teccomm.les.inf.puc-rio.br/emagno/ajato/

A ferramenta AJATO também suporta a geração automática de alertas de problemas e a associação de valores limites às regras heurísticas.

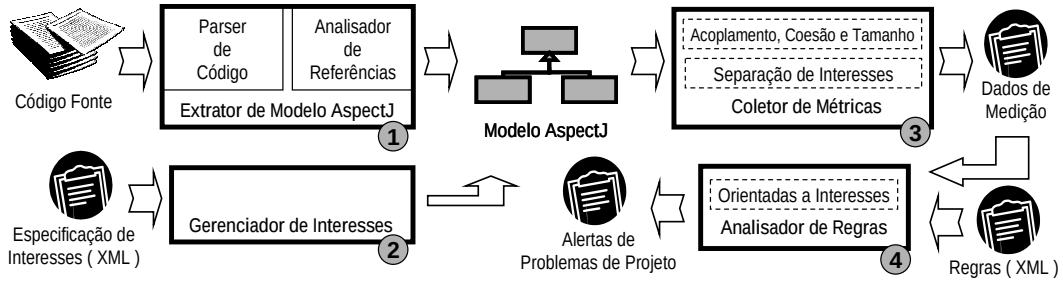


Figura 3. Arquitetura da ferramenta de avaliação [Figueiredo, Garcia e Lucena, 2006]

A versão atual de AJATO permite a avaliação de sistemas implementados em Java e AspectJ. Sua arquitetura define quatro módulos apresentados na Figura 3: Extrator de Modelo AspectJ, Gerenciador de Interesses, Coletor de Métricas e Analisador de Regras. O módulo Extrator de Modelo AspectJ efetua o *parser* do código fonte de entrada e constrói uma estrutura representativa do sistema, chamada Modelo AspectJ (Figura 3). Esta estrutura tem o objetivo de facilitar a tarefa de medição. Além disso, dois submódulos compõem o Extrator de Modelo: Parser de Código e Analisador de Referências. O primeiro submódulo extrai informações do código enquanto o segundo captura os relacionamentos existentes entre os elementos sintáticos. Exemplos destes relacionamentos são importações, herança, associações e chamadas de métodos.

O módulo Gerenciador de Interesses faz o mapeamento de estruturas sintáticas do Modelo AspectJ para os interesses a serem avaliados. Para fazer este mapeamento, necessário para que as métricas orientadas a interesses sejam aplicadas, o Gerenciador de Interesses utiliza uma especificação em formato XML. O terceiro módulo da ferramenta, Coletor de Métricas, é responsável pelo cálculo das medições no Modelo AspectJ. As métricas medidas por este módulo se classificam em quatro categorias principais: acoplamento, coesão, tamanho e separação de interesses. Finalmente, o módulo Analisador de Regras tem como propósito principal a aplicação de regras heurísticas. Ele utiliza dois artefatos de entrada: (i) o resultado das medições disponibilizado pelo módulo Coletor de Métricas e (ii) um arquivo que define as regras e a ordem em que estas devem ser aplicadas. O resultado deste módulo é um relatório com os alertas de possíveis problemas de projetos.

6. Avaliação

O método de avaliação proposto neste artigo foi usado no contexto de seis diferentes estudos empíricos (Seção 6.1). Em adição a estes estudos, outros grupos de pesquisa [Bartolomei et al., 2006] [Filho et al., 2006] [Godil and Jacobsen, 2005] têm aplicado nossa abordagem orientada a interesses e têm fornecido importantes contribuições [Figueiredo, 2006]. A avaliação da abordagem apresentada nesta seção ocorre em duas dimensões: eficácia do método para detecção de problemas relacionados a interesses transversais (Seção 6.2) e precisão (i.e. taxa de acerto) das heurísticas (Seção 6.3).

6.1. Os Estudos de Caso

Seis estudos têm sido utilizados na avaliação de nossa abordagem e estes foram selecionados por apresentarem uma série de critérios relevantes a esta pesquisa.

Primeiro, temos a nossa disposição as implementações OO e OA destes sistemas. Segundo, pela heterogeneidade de interesses que neles podem ser encontrados (segunda coluna, Tabela 2), incluindo tanto interesses de escopo arquitetural, tais como persistência e tratamento de exceções, quanto interesses mais localizados a certas partes do software, como alguns padrões de projeto [Gamma et al., 1995]. Terceiro, os sistemas são bons representantes de diferentes domínios de aplicações cobrindo desde simples instâncias de padrões até aplicações Web reais, sistema de *middleware* reflexivo e sistema multi-agentes. Finalmente, nossos estudos de caso servem como efetivos *benchmarks* por envolverem cenários nos quais é difícil decidir pela aspectização ou não de certos interesses transversais [Figueiredo et al., 2007].

O primeiro estudo de caso [Figueiredo e Staa, 2005] [Garcia et al., 2005, 2006] no qual aplicamos o método é uma biblioteca de padrões de projeto desenvolvida por Hannemann e Kiczales (HK) [2002]. Da mesma forma que HK, nestes trabalhos nós tratamos cada papel definido nos padrões como um interesse. O segundo estudo é um sistema de *middleware* em conformidade com o modelo OpenOrb [Cacho et al., 2006] e o terceiro é uma ferramenta de medição [Figueiredo, Garcia e Lucena, 2006]. Nestes dois estudos consideramos cada padrão de projeto como um interesse a ser avaliado. Os dois estudos seguintes são: o núcleo do sistema de controle de versões, CVS, da plataforma Eclipse [Filho et al., 2006] e um sistema Web para registro de queixas relacionadas ao sistema de saúde da cidade de Recife, chamado Health Watcher [Figueiredo et al., 2007] [Greenwood et al., 2007]. Tanto no CVS quanto no Health Watcher a avaliação foi direcionada a interesses de grande impacto nos artefatos do sistema desde a arquitetura até a fase de implementação. Finalmente, a última aplicação alvo de nossa pesquisa é um sistema multi-agentes para gerência de portais Web chamado Portalware [Oliveira et al., 2006] e neste sistema o foco da avaliação foi em interesses específicos de agentes.

6.2. Detecção de Problemas

A avaliação heurística orientada a interesses tem demonstrado grande sucesso por revelar inúmeros problemas de projeto nos sistemas adotados. A Tabela 2 apresenta resumidamente o resultado da aplicação do método proposto nos seis estudos de caso. A duas primeiras colunas desta tabela listam os sistemas utilizados e a natureza dos interesses avaliados e cada um deles. As três demais colunas mostram o número de problemas de projeto encontrado em cada estudo. Tais problemas são classificados em três categorias de acordo com as heurísticas utilizadas em suas detecções (Seção 4): interesses transversais, componentes com modularidade pobre e *bad smells*.

Tabela 2. Resumo de problemas de projeto encontrados nos estudos de caso

Estudos de Caso	Natureza dos Interesses	Interesses Transversais	Componentes com Modularidade Pobre	Bad Smells [Fowler, 1999]
Biblioteca de Padrões	Papéis dos Padrões	14	10	2
Middleware OpenOrb	Padrões de Projeto (e suas interações)	13	7	6
Ferramenta de Medições		8	6	3
Plugin CVS do Eclipse	Interesses de impacto arquitetural	1	44	3
Health Watcher		4	48	10
Portalware	Interesses de Agentes	4	9	6

No total, 198 problemas relacionados a interesses transversais foram apontados pelo método em nossos estudos, o que tem demonstrado grandes oportunidades para

melhorarias destes sistemas [Figueiredo et al., 2007]. Deste montante, a maior parte (124 casos) caracteriza componentes cuja modularidade poderia ser melhorada (quarta coluna, Tabela 2), ou seja, componentes que além de possuírem interesses transversais também apresentam problemas em atributos tradicionais da engenharia de software, tais como elevada complexidade interna, alto acoplamento e baixa coesão. Nos seis estudos foram encontrados 44 interesses transversais (terceira coluna), sendo a grande maioria relacionada a padrões de projetos (3 primeiros estudos). Além disso, os problemas encontrados incluem 30 ocorrências de *bad smells* bem conhecidos, tais como God Class e Shotgun Surgery.

Por restrições de espaço não apresentamos uma discussão detalhada sobre os resultados dos estudos de caso, mas estas se encontram publicadas nos artigos referenciados pela Seção 6.1. No entanto, uma conclusão geral é ser difícil atingir um projeto OA de alta qualidade mesmo para especialistas em DSOA. Muitos problemas ocorrem porque existem diferentes interesses a serem modularizados e, ao mesmo tempo, muitos outros princípios fundamentais da engenharia de software precisam ser observados e satisfeitos. Desta forma, ferramentas e métodos de avaliação como propostos neste artigo são de fundamental importância para melhoria de qualidade de software nesse novo paradigma de desenvolvimento.

6.3. Precisão das Heurísticas

Diferentemente da seção anterior que procurou demonstrar como o método orientado a interesses é capaz de encontrar problemas, nesta seção avaliamos o grau de precisão apresentado pelas heurísticas em nossos estudos. Para tal avaliação, comparamos o resultado das regras heurísticas com a opinião de especialistas nos domínios dos sistemas e com trabalhos anteriores publicados na literatura [Hannemann and Kiczales, 2002] [Soares, Laureano and Borba, 2002]. São considerados especialistas aqueles pesquisadores que participaram do desenvolvimento (e/ou avaliaram previamente a qualidade) dos sistemas alvos de nossos estudos. Além disso, estes pesquisadores publicaram as observações de seus estudos de tal forma que pudéssemos utilizar para efeito de comparação. Como resultado, esta dimensão da avaliação conta quantas vezes a literatura/especialistas concordam com as heurísticas e quantas vezes eles discordam.

A Tabela 3 apresenta de forma resumida o resultado da aplicação das heurísticas para espalhamento de interesses (Seção 4.1) utilizando o critério acima descrito. A primeira coluna lista os estudos de caso. Note que o primeiro estudo é direcionado a padrões de projeto, mas avalia os papéis dos padrões. Em contrapartida, os dois estudos seguintes, *middleware* e ferramenta de medição, envolvem interações entre pares de padrões de projeto e avaliam cada padrão como um interesse. A segunda e a terceira coluna da Tabela 3 mostram, respectivamente, os interesses avaliados e a melhor forma de separar tais interesses (OO ou OA) segundo a literatura e os especialistas consultados. Por limitação de espaço, apenas alguns dos interesses avaliados nos três primeiros estudos são mostrados. Finalmente, a última coluna indica quando as heurísticas classificam os interesses como transversais ou não (base) e se esta classificação concorda (acerto) ou discorda (falha) da informação presente na coluna anterior. É importante ressaltar que esta tabela mostra uma visão limitada destes estudos. Por exemplo, no estudo do Health Watcher o método foi aplicado em 10 sucessivas versões OO e OA do sistema [Greenwood et al., 2007] de tal forma a avaliar

da estabilidade deste software e, principalmente, avaliar a característica de aplicação iterativa do método (Seção 3).

Tabela 3. Resultados parciais das heurísticas para espalhamento de interesses

Estudos de Caso			**Interesses Avaliados**	**Especialistas e Literatura**	**Heurística (Resultado)**
Biblioteca de Padrões de Projetos	Padrões	Chain of Respons.	Papel Handler	OA	Base (Falha)
		Factory Method	Papel Creator	OO	Transversal (Falha)
		Observer	Papel Observer	OA	Transversal (Acerto)
			Papel Subject	OA	Transversal (Acerto)
		Mediator	Papel Colleague	OA	Transversal (Acerto)
			Papel Mediator	OA	Base (Falha)
Middleware OpenOrb	Composição de Padrões	Observer com Factory Method	Observer	OA	Transversal (Acerto)
			Factory Method	OO	Base (Acerto)
		Façade com Singleton	Façade	OO	Base (Acerto)
			Singleton	OA	Transversal (Acerto)
Ferramenta de Medições		Prototype com State	Prototype	OA	Transversal (Acerto)
			State	OA	Base (Falha)
		Interpreter com Proxy	Interpreter	OA	Transversal (Acerto)
			Proxy	OA	Transversal (Acerto)
Plugin CVS do Eclipse			Tratam. de Exceções	OA	Transversal (Acerto)
Health Watcher			Negócios	OO	Base (Acerto)
			Concorrência	OA	Transversal (Acerto)
			Distribuição	OA	Transversal (Acerto)
			Persistência	OA	Transversal (Acerto)
			Tratam. de Exceções	OA	Transversal (Acerto)
Portalware			Adaptação	OA	Transversal (Acerto)
			Autonomia	OA	Transversal (Acerto)
			Colaboração	OA	Transversal (Acerto)
			Interação	OA	Transversal (Acerto)

Ainda em relação à Tabela 3, nos limitamos a discutir as falhas das regras heurísticas, ou seja, as ocorrências de falsos positivos e falsos negativos. Falsos positivos ocorrem quando as regras classificam o interesse como transversal e este não o é; portanto, elas erroneamente reportam um problema. Por outro lado, falsos negativos ocorrem quando as regras omitem um problema de interesse transversal. A Tabela 3 mostra um falso positivo levantado pelas regras e diz respeito ao padrão Factory Method quando avaliado o papel Creator. Neste caso não se pode considerar que as regras falham totalmente porque uma das classes que realiza o papel avaliado (`GUIComponentCreator`) possui código não relacionado ao padrão. O que ocorre é um entrelaçamento nesta classe do interesse do padrão com código que implementa a gerência de interface com o usuário (janelas e quadros), o que não deveria acontecer em classes desempenhando o papel de construtores de objetos. Portanto, mesmo tendo erroneamente classificado o interesse como transversal, as regras detectaram um problema de separação de interesses nesta instancia do padrão.

Em adição, a Tabela 3 apresenta três falsos negativos das heurísticas quando avaliados os padrões Chain of Responsibility (papel Handler), Mediator (papel Mediator) e State. O padrão Chain of Responsibility não foi detectado como interesse transversal porque os componentes da instancia disponível [Hannemann e Kiczales, 2002] são muito simples (devido à sua natureza em ser uma biblioteca de padrões). Assim, as heurísticas não foram capazes de identificar a característica transversal deste

padrão. Situação semelhante ocorre com o padrão Mediator. Em contrapartida, o padrão State também resultou em falso negativo devido à estratégia adotada na avaliação. Apesar de este padrão possuir dois papéis (State e Context), estes foram considerados como um único interesse no terceiro estudo. Assim, as transições de estados (pertencente ao papel State) que geralmente atravessam outros componentes do sistema ocorrem dentro de classes desempenhando o papel Context nesta implementação específica do padrão. Portanto, o papel State atravessa apenas o papel Context que não é transversal, ocultando o comportamento transversal do padrão.

Em relação aos demais estudos, as heurísticas têm apresentado uma precisão geral de aproximadamente 85 % [Figueiredo et al., 2007]. Além disso, elas possuem melhor desempenho quando interesses de maior granularidade são avaliados, como nos três últimos estudos na Tabela 3. Finalmente, acreditamos que o amadurecimento e robustez do método se devem a sua aplicação em sistemas desenvolvidos por outros grupos de pesquisa (como os estudos 1, 2, 4, 5 e outros não mencionados nesta seção).

7. Trabalhos Relacionados

Tekinerdogan [2004] propôs um método de análise OA, mas seu objetivo é a identificação de interesses transversais em aquiteturas de software. Além disso, este método se concentra na avaliação de arquitetura com respeito a cobertura de cenários de casos de usos e não se preocupa com atributos de modularidade e separação de interesses. Nossa abordagem, por outro lado, apresenta um método quantitativo para avaliação de projeto e implementação baseado em importantes princípios de engenharia de software, tais como acoplamento, coesão, tamanho e separação de interesses.

Regras heurísticas para detecção de problemas de projetos são propostas por Marinescu [2004]. Entretanto, estas regras se aplicam exclusivamente a sistemas OO e não relacionam os problemas encontrados com os interesses do sistema. Em contrapartida, este artigo propõe regras heurísticas OA e as classificam em três categorias. No mais, nossas heurísticas envolvem métricas orientadas a interesses que relacionam os problemas encontrados com interesses que potencialmente os causam. Em comparação ao trabalho de Marinescu, consideramos nossas heurísticas como complementares, pois, constatamos ao aplicarmos tais heurísticas nos mesmos sistemas que as heurísticas propostas neste artigo e as de Marinescu detectam diferentes conjuntos de problemas [Figueiredo et al., 2007].

Refatorações orientadas a aspectos têm sido definidas na literatura, mas elas não se conectam diretamente aos valores de métricas para acoplamento, coesão, tamanho e separação de interesses. Em [Monteiro and Fernandes, 2005], alguns poucos *bad smells* são usados para identificar oportunidades de refatorações OA. Desta forma, consideramos catálogos de refatorações como sendo complementar a nossa abordagem, pois eles podem ser usados em uma das atividades do método proposto.

Muitas ferramentas têm sido especificamente desenvolvidas para suportar o mapeamento de interesses para elementos sintáticos dos sistemas. Estas ferramentas, chamadas de mineradoras de aspectos [Robillard and Murphy, 2002], podem facilitar a identificação de segmentos de programas que realizam os interesses a serem avaliados em nosso método. Estas abordagens também são vistas como complementares e elas podem ser integradas à ferramenta de avaliação que suporta nosso método (Seção 5). Além disso, uma ferramenta de medição OA é proposta em [Ceccato and Tonella, 2004]

para automatizar o processo de medição. Entretanto, esta ferramenta não está associada a um método sistemático de avaliação. No mais, nossa ferramenta suporta métricas orientadas a interesses além das métricas de acoplamento, coesão e tamanho.

8. Conclusões e Resumo das Contribuições

DSOA é um assunto que tem atraído o interesse tanto da comunidade acadêmica quanto da indústria. Apesar de sua popularidade, pouca atenção é dada para métodos de avaliação de artefatos de software produzidos seguindo este novo paradigma, ou mesmo se tais artefatos apresentam certas qualidades desejáveis. Neste contexto, as principais contribuições deste artigo são: (i) um método com passos bem definidos para avaliação orientada a interesses, (ii) novas métricas que diferem das existentes por explicitamente considerarem interesses como abstrações no processo de medição, (iii) um conjunto de heurísticas para suporte a detecção de problemas em projetos, (iv) uma ferramenta de medição e avaliação para análise automática de sistemas grandes e/ou complexos, e (v) seis estudos de caso que avaliam a viabilidade do método.

A exceção da ferramenta, todos os elementos propostos neste artigo que constituem a abordagem sensível a interesses são independentes de linguagem e paradigma de desenvolvimento. Na verdade, em nossos estudos de caso o método foi aplicado em diferentes implementações OO e OA, tais como Java, AspectJ e CaesarJ [Figueiredo et al., 2005, 2007] [Greenwood et al., 2007]. Em relação à originalidade deste trabalho, nossos resultados foram publicados em um periódico de relevância internacional bem como em diversas conferências e workshops, isto é, 1 periódico, 7 conferências internacionais e 3 workshops. Além de apresentar o método e a ferramenta, nosso papel em algumas destas publicações foi oferecer suporte metodológico e ferramental bem como avaliar nossa abordagem. Em adição, a avaliação orientada a interesses que propomos tem sido largamente utilizada por grupos de pesquisa tanto em instituições nacionais (e.g. PUC-Rio, UNICAMP, UFPE, UFRN) como em instituições internacionais (e.g. Universidade de Lancaster, Universidade de Málaga, Universidade de Toronto, Universidade de Waterloo, Universidade de Milão). Mais recentemente, temos trabalhado nos desafios de aplicação do método orientado a interesses para avaliação de arquiteturas de software [Sant'Anna et al., 2007].

Referências

Bartolomei, T., Garcia, A. and Figueiredo, E. (2006) "Towards a Unified Coupling Framework for Measuring Aspect-Oriented Programs". In: Ws on Software Quality Assurance, USA.

Cacho, N., Sant'Anna, C., Figueiredo, E., Garcia, A., Batista, T. and Lucena, C. (2006) “Composing Design Patterns: A Scalability Study of Aspect-Oriented Programming”. In: Int’l Conf. on Aspect Oriented Software Development (AOSD), pp. 109-121, Germany.

Ceccato, M. and Tonella, P. (2004) “Measuring the Effects of Software Aspectization”. In: 1st Workshop on Aspect Reverse Engineering, The Netherlands.

Chidamber, S. and Kemerer, C. (1994) “A Metrics Suite for Object Oriented Design”. In: IEEE Transactions on Software Engineering, v. 20, n. 6, pp. 476-493.

Figueiredo, E., Garcia, A. and Lucena, C. (2006) “AJATO: an AspectJ Assessment Tool”. In: European Conference on Object-Oriented Programming (ECOOP), Nantes, France.

Figueiredo, E. (2006) “Uma Abordagem Quantitativa para Desenvolvimento de Software Orientado a Aspectos”. Dissertação de Mestrado, 140 p., Departamento de Informática, PUC-Rio, Brasil. http://www.lancs.ac.uk/postgrad/figueire/publications/dissertacao.pdf

Figueiredo, E., Garcia, A., Sant'Anna, C., Kulesza, U. and Lucena, C. (2005) "Assessing Aspect-Oriented Artifacts: Towards a Tool-Supported Quantitative Method". In: 9th ECOOP Ws on Quantitative Approaches in OO Software Eng. (QAOOSE), pp. 58-69, UK.

Figueiredo, E. e Staa, A. (2005) "Avaliação de um Modelo de Qualidade para Implementações Orientadas a Objetos e Orientadas a Aspectos". Relatório Técnico MCC 14/05, 29 páginas, Departamento de Informática, PUC-Rio, Rio de Janeiro.

Figueiredo, E., Sant'Anna, C., Garcia, A. and Lucena, C. (2007) "On the Saga of Concern-Sensitive Design Heuristics". Submitted to International Conference on Automated Software Engineering (ASE'07), Atlanta, Georgia, USA.

Filho, F., Cacho, N., Figueiredo, E., Maranhão, R., Garcia, A. and Rubira, C. (2006) "Exceptions and Aspects: The Devil is in the Details". In: 14th Symposium on Foundations of Software Engineering (FSE), pp. 152-162, Portland, USA.

Fowler, M. (1999) "Refactoring: Improving the Design of Existing Code". Addison-Wesley.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995) "Design Patterns: Elements of Reusable Object-Oriented Software". Addison-Wesley Longman Publishing.

Garcia, A., Sant'Anna, C., Figueiredo, E., Kulesza, U., Lucena, C. and Staa, A. (2006) "Modularizing Design Patterns with Aspects: A Quantitative Study". LNCS Transactions on Aspect Oriented Software Development (TAOSD), 1, pp. 36-74.

Garcia, A., Sant'Anna, C., Figueiredo, E., Kulesza, U., Lucena, C. and Staa, A. (2005) "Modularizing Design Patterns with Aspects: A Quantitative Study". In: Int'l Conf. on Aspect Oriented Software Development (AOSD), pp. 3-14, Chicago, USA.

Greenwood, P., Bartolomei, T., Figueiredo, E., Dosea, M., Garcia, A., Cacho, N., Sant'Anna, C., Soares, S., Borba, P., Kulesza, U. and Rashid, A. (2007) "On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study". In: ECOOP, Berlin, Germany.

Godil, I. and Jacobsen, H. (2005) "Horizontal Decomposition of Prevayler". In: Conf. of the Center for Advanced Studies on Collaborative Res. (CASCON), p. 83-100, Toronto, Canada.

Hannemann, J. and Kiczales, G. (2002) "Design Pattern Implementation in Java and AspectJ". In: OO Prog., Systems, Languages & Applications (OOPSLA), pp. 161-173, Seattle, USA.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. and Irwin, J. (1997) "Aspect-Oriented Programming". In: European Conf. on OO Programming (ECOOP), pp. 220-242.

Marinescu, R. (2004) "Detection Strategies: Metrics-Based Rules for Detecting Design Flaws". In: Int'l Conference on Software Maintenance (ICSM), pp. 350-359, Chicago, USA.

Monteiro, M. and Fernandes, J. (2005) "Towards a Catalog of Aspect-Oriented Refactorings". In: Int'l Conf. on Aspect Oriented Software Development (AOSD), pp. 111-122, Chicago.

Oliveira, A., Cysneiros, L., Leite J., Figueiredo, E. and Lucena, C. (2006) "Integrating Scenarios, i*, and AspectT in the Context of Multi-Agent Systems". In: Conference of the Center for Advanced Studies on Collaborative Research (CASCON), Toronto, Canada.

Robillard, M. and Murphy, G. (2002) "Concern Graphs: Finding and Describing Concerns Using Structural Program Dependencies". In: ICSE'02, pp. 406-416, Orlando, USA.

Sant'Anna, C., Figueiredo, E., Garcia, A. and Lucena, C. (2007) "On the Modularity Assessment of Software Architectures: Do my Architectural Concerns Count?" In: AOSD Workshop on Aspects in Architectural Description (AARCH), Vancouver, Canada.

Soares, S., Laureano, E. and Borba, P. (2002) "Implementing Distribution and Persistence Aspects with AspectJ". In: OOPSLA, pp 174-190, Seattle, USA.

Tekinerdogan, B. (2004) "ASAAM: Aspectual Software Architecture Analysis Method". In: IEEE/IFIP Conference on Software Architecture, pp. 5-14, Norway.