PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Programa de Pós-Graduação em Informática

MÉTRICAS PARA AVALIAÇÃO DO GRAU DE QUANTIFICAÇÃO DE SISTEMAS ORIENTADOS POR ASPECTOS

Jaqueline Faria de Oliveira

Belo Horizonte

2010

Jaqueline Faria de Oliveira

MÉTRICAS PARA AVALIAÇÃO DO GRAU DE QUANTIFICAÇÃO DE SISTEMAS ORIENTADOS POR ASPECTOS

Dissertação apresentada ao Programa de Pós-Graduação em Informática como requisito parcial para obtenção do Grau de Mestre em Informática pela Pontifícia Universidade Católica de Minas Gerais.

Orientador: Prof. Dr. Marco Túlio de Oliveira Valente

Co-orientador: Prof. Dr. Humberto Torres Marques Neto

Belo Horizonte

2010

À minha família e amigos.

Sem estes nenhum objetivo pode ser alcançado.

AGRADECIMENTOS

Agradeço primeiramente a Deus por ser o meu guia em todos os momentos, principalmente naqueles em que pensei em desistir.

Agradeço a minha família por terem me ensinado o valor das pessoas, do conhecimento e das conquistas. Sinceros agradecimentos aos meus pais e minhas irmãs, Joice e Tania, por me apoiarem em todas as minhas iniciativas, pela paciência, pelo grande incentivo, e pela compreensão nos momentos em que estive ausente. Ao meu namorado Mateus, que me apoiou nos momentos mais difíceis e compreendeu sempre minhas faltas.

Agradeço aos amigos que me apoiaram e ajudaram, direta e indiretamente nesta caminhada.

Aos professores Raquel Mini, Clodoveu Davis, Ana Maria e Zenilton Kleber por compartilharem seus conhecimentos nas disciplinas que tive oportunidade de cursar. À Giovanna, secretária acadêmica, pela presteza e atenção especial que dá a cada aluno. Aos colegas do mestrado um agradecimento especial, pois o apoio destes foi fundamental para que eu pudesse seguir em frente nesse projeto.

Agradeço à FAPEMIG pelo incentivo para conclusão deste trabalho.

Agradeço a César Couto, pelo apoio para o desenvolvimento desta pesquisa, sua participação foi muito importante para a conclusão deste trabalho.

Agradeço ao meu co-orientador Humberto Torres, pelo apoio didático e especialmente motivacional para a conclusão desta dissertação, por ter se esforçado juntamente comigo para a conclusão desta etapa de minha vida.

Finalmente agradeço ao meu professor e orientador Marco Túlio, pelo conhecimento compartilhado e pelo grande exemplo de profissional da educação e pesquisador. Qualquer agradecimento seria pouco perto da sua dedicação, apoio, disponibilidade e compreensão. Meus sinceros agradecimentos por essa lição de vida.

Muito obrigada a todos!

RESUMO

A refatoração de sistemas orientados por aspectos é projetada para separar e modularizar interesses transversais e reduzir o espalhamento e entrelaçamento de código. No entanto, estes resultados não são sempre alcançados. Pesquisas sobre a avaliação da refatoração de sistemas orientados por aspectos são baseadas na aplicação de métricas de software. Essas métricas geralmente medem atributos do software já refatorado, o que implica na necessidade de realizar a refatoração antes de conseguir qualquer resultado significativo.

Neste trabalho, defendemos a utilização da quantificação como um fator de medição para avaliar a refatoração do interesse para aspectos. A refatoração é indicada quando interesses transversais possuem um alto grau de quantificação, ou seja, quando diversas chamadas do interesse podem ser modularizadas por um número pequeno de *advices*. Para isso, propomos duas novas métricas para avaliar o grau de quantificação dos interesses que serão refatorados para aspectos. A métrica *Quantification Degree* (QD) que mede o grau de quantificação do interesse transversal, e a métrica *Scattering Reduction* (SR), que por sua vez mede a redução do espalhamento de código. Ambas as métricas são calculadas com base no código fonte original, permitindo aos engenheiros de software uma avaliação do resultado da refatoração antes de qualquer alteração estrutural do sistema. As métricas propostas são descritas formalmente e são apresentados exemplos de sua aplicação em estudos de caso que utilizam sistemas reais.

Nós também propomos nesta pesquisa o projeto e a implementação da ferramenta *ConcernMetrics*. Esta ferramenta é usada para automatizar o cálculo das métricas propostas nesta dissertação, QD e SR. A ferramenta *ConcernMetrics* também calcula métricas de separação de interesses já difundidas para avaliação de sistemas orientados por aspectos. Estes cálculos são feitos diretamente no código orientado por objetos de um sistema existente, ou seja, antes de interesses transversais serem extraídos para aspectos. São apresentados resultados da utilização da ferramenta *ConcernsMetrics* em sistemas reais.

Nossos resultados mostram que as métricas QD e SR calculam efetivamente o grau de quantificação de interesses transversais antes da refatoração do sistema. Ressaltamos também que a ferramenta *ConcernMetrics*, desenvolvida ao longo deste trabalho, facilita o cálculo das métricas propostas.

Palavras-chave: Programação orientada por aspectos. AspectJ. Quantificação. Separação de Interesses. Métricas. Refatoração.

ABSTRACT

Refactoring of aspect-oriented systems is designed to separate and modularize the crosscutting concerns and reduce the spreading and interlacing of code. However, these results are not always achieved. The research on evaluation of the refactoring aspect-oriented systems are based on the application of software metrics. These metrics usually measure attributes of software already refactored, which implies the need to perform refactoring before achieving any significant result.

In this work, we advocated the use of quantification as a factor of measurement for assessing the interests of refactoring to aspects. Refactoring is indicated when crosscutting concerns have a high degree of quantification, ie, when several calls of concern can be modularized by a small number of advices. For this, we propose two new metrics to evaluate the degree of quantification of interests that will be refactored to aspects. The metric Quantification Degree (QD), which measures the degree of quantification of crosscutting interest, and the metric Scattering Reduction (SR), which in turn measures the reduction of the spreading code. Both metrics are calculated based on the original source code, allowing software engineers to evaluate the result of refactoring before any structural change in the system. The proposed metrics are formally described and presented examples of its application in case studies using real systems.

We also present this research work the design and the implementation of the ConcernMetrics. This tool is used to automate the calculation of the metrics proposed in this dissertation, QD and SR. The ConcernMetrics also calculates metrics of separation of concerns already widespread for evaluation of aspect-oriented systems. These calculations are done directly on the object-oriented code of an existing system, ie, before being extracted crosscutting concerns to aspects. We present the results of using the tool ConcernsMetrics with real systems.

Our results show that the metrics QD and SR calculate effectively the degree of quantification of crosscutting concerns before the refactoring of the system. We also emphasize that the tool ConcernMetrics developed throughout this paper facilitates the calculation of the metrics proposed.

Key-words: Aspect-oriented programming. AspectJ. Quantification. Separation of Concerns. Metrics. Refactoring.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AIF - *Attribute Inheritance Factor*

AST - *Abstract Syntax Tree*

BAC - *Basic and Advanced Dynamic Crosscuts*

CAE - *Coupling on Advice Execution*

CBC - *Coupling Between Components*

CBO - *Coupling Between Objet Classes*

CDA - *Crosscutting Degree of an Aspect*

CDC - *Concern Diffusion over Components*

CDO - *Concern Diffusion over Operations*

CF - *Coupling Factor*

CFA - *Coupling on Field Access*

CIA - *Classes, Interfaces, and Aspects*

CIM - *Coupling on Intercepted Modules*

CLC - *Concern Lines of Code*

CMC - *Coupling on Method Call*

CRR - *Code Replication Reduction*

CS - *Class Size*

DIT - *Depth of the Inheritance Tree*

DOSC - *Degree of Scattering Across Classes*

DOSM - *Degree of Scattering Across Methods*

HHC - *Heterogeneous and Homogeneous Crosscuts*

JPS - *join point shadows*

LCOM - *Lack of Cohesion in Methods*

LCOO - *Lack of Cohesion in Operations*

LOC - *Lines of Code*

MIF - *Method Inheritance Factor*

NOA - *Number of Operations Added by a Subclass*

NOC - *Number of Children*

NOO - *Number of Operations Overridden by a Subclass*

POA - Programação Orientada por Aspectos

POO - Programação Orientada por Objetos

QD - *Quantification Degree*

RFC - *Response For a Class*

RFM - *Response For a Module*

SDC - *Static and Dynamic Crosscuts*

SI - *Specialization Index*

SR - *Scattering Reduction*

WMC - *Weighted Methods per Class*

WOC - *Weighted Operations per Component*

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

O paradigma de programação orientada por aspectos (POA) tem se tornado a tecnologia mais difundida para modularização de interesses transversais. Esses interesses normalmente encontram-se espalhados e entrelaçados no código fonte, que por sua vez podem ocasionar problemas de reutilização, modularização, extensibilidade, entre outros. POA estende paradigmas de programação tradicionais, como a procedural e orientada por objetos, possibilitando abstrações no comportamento do sistema. Essas abstrações podem ocorrer de duas formas, por meio da transversalidade estática que permite alterações na hierarquia de classes e introdução de métodos e atributos em classes através de declarações inter-tipo, e da transversalidade dinâmica que permite a alteração no comportamento de um programa em pontos específicos (LADDAD, 2003). A linguagem AspectJ é um bom exemplo de linguagem orientada por aspectos, sendo na atualidade a mais madura. AspectJ estende Java com a inclusão de novas abstrações da linguagem, incluindo *join points*, *pointcuts*, *advices* e aspectos (KICZALES et al., 2001).

Como vantagens da POA podem-se citar a modularização de requisitos transversais, reutilização, extensibilidade, facilidade de manutenção e redução do espalhamento e entrelaçamento de código (KICZALES; HILSDALE, 2001; KICZALES et al., 1997; IRWIN et al., 1997; SOMMERVILLE, 2006). Com o intuito de validar esses benefícios em sistemas reais e ainda estudar outras vantagens e desvantagens na utilização de aspectos, diversos estudos foram feitos (EADDY et al., 2008; FIGUEIREDO et al., 2009; FILHO et al., 2006; APEL, 2010; GARCIA et al., 2007; STEIMANN, 2006). Atributos como tamanho, complexidade, acoplamento, coesão, herança, entre outros, amplamente estudados a partir da aplicação de métricas de software orientadas por objetos, são foco também de estudos atuais relacionados a sistemas orientados por aspectos (SANT'ANNA et al., 2003; CECCATO; TONELLA, 2004; APEL, 2010; EADDY et al., 2008). Nestes utilizam-se ainda as medidas de espalhamento e entrelaçamento de código, separação de interesses e quantificação. Dentre esses atributos medidos em sistemas orientados por aspectos, é verificado que a

noção de quantificação (FILMAN; FRIEDMAN, 2005) é pouco explorada, argumenta-se que um dos usos mais favoráveis dos aspectos acontece quando o seu código estende amplamente declarações quantificadas, ou seja, declarações que têm efeito em diversos pontos do código. Quando isso acontece, os aspectos contribuem para a separação de interesses, uma vez que o código pode ser confinado em um único bloco de código.

As avaliações em sistemas orientados por aspectos, conforme ocorre nos sistemas dos demais paradigmas de programação, são normalmente feitas utilizando-se métricas de software, as quais consistem no processo de medição de atributos do sistema. A utilização de métricas na engenharia de software, embora possam prover informações importantes para medição e controle, não é um processo amplamente difundido (FENTON; NEIL, 1999). Esse fato pode ser atribuído à dificuldade da aplicação de métricas em sistemas muito grandes, ou o fato dos processos de medição nas instituições não serem maduros, além disso o alto custo dos programas de métricas também pode ser um fator dificultador (KANER; MEMBER; BOND, 2004).

Em geral, trabalhos na área de métricas de software para sistemas orientados por aspectos medem características específicas do código fonte já refatorado para aspectos, dessa forma é necessária a implementação dos aspectos para posterior avaliação. Por exemplo, as métricas de separação de interesses *Concern Diffusion over Operations* (CDO) e *Concern Diffusion over Components* (CDC) (GARCIA et al., 2005; SANT'ANNA et al., 2003; GREENWOOD et al., 2007), dão uma noção da importância e impacto dos aspectos no sistema, porém seu cálculo é feito com base no código fonte orientado por aspectos. Alguns trabalhos propõem o refinamento das métricas CK (CHIDAMBER; KEMERER, 1994), para avaliação de projetos orientados por aspectos, como os trabalhos de Sant'Anna et al. (2003) e Ceccato e Tonella (2004). Outros trabalhos fazem novas propostas de métricas para avaliação de aspectos como o trabalho de Apel (2010), esses também se baseiam na análise do código orientado por aspectos.

Essas métricas apresentam vantagens e limitações diversas, porém nenhum conjunto de métricas consegue atender a todos os atributos necessários para a avaliação e previsão do resultado da refatoração de sistemas para aspectos. O conceito de atributos necessários para uma avaliação e previsão de software é muito complexo e obedece a diversos fatores, conforme cada sistema e tecnologia, esse fato também se aplica à avaliação de softwares orientados por aspectos. As métricas propõem-se, em sua maioria, a medir atributos estáticos baseadas nos produtos de software, código fonte, projeto, diagramas e etc. Outra característica observada nas métricas existentes é o fato de serem normal-

mente métricas avaliativas, ou seja, avaliam atributos já existentes baseados em artefatos estáticos, no caso de sistemas orientados por aspectos o código fonte já refatorado é o principal artefato. As métricas atuais para sistemas orientados por aspectos, em sua maioria, não provêem informações de previsão, ou seja, não possibilitam a estimativa prévia de determinados comportamentos ou resultados da refatoração do sistema baseados em um conjunto de informações existentes.

A carência de métricas para previsão de sistemas orientados por aspectos tem tornado difícil a avaliação da refatoração de um sistema previamente à implementação dos interesses utilizando-se aspectos, e essa dificuldade aumenta conforme o tamanho do sistema. De acordo com pesquisas, não são identificadas métricas que possibilitem a avaliação prévia da refatoração para aspectos baseada no código fonte original do sistema.

1.2 Objetivos

Definir métricas para avaliação do grau de quantificação de requisitos transversais com o intuito de prover uma medida para avaliação da refatoração de um sistema para aspectos. Possibilitar aos mantenedores de software decidirem, de uma forma economicamente efetiva, se vale a pena usar aspectos em seus sistemas através destas métricas. Os objetivos específicos desta pesquisa são detalhados a seguir:

- Definir a métrica *Quantification Degree* (QD) para avaliação do grau de quantificação de requisitos transversais, e a sub-métrica *Scattering Reduction* (SR) com o intuito de medir a redução do espalhamento de um interesse no sistema. Essas métricas têm o objetivo de inferir o grau de quantificação dos interesses transversais do sistema relativos a uma possível modularização destes através de aspectos, para serem utilizados como fatores de avaliação da refatoração para aspectos do sistema.

- Projetar e implementar a ferramenta *ConcernMetrics* para apoio no processo de cálculo das métricas de quantificação definidas. A ferramenta *ConcernMetrics* irá possibilitar a identificação e avaliação de requisitos transversais, criação de um *Modelo de Concerns* e efetuar o cálculo das métricas de quantificação QD e SR. Deverá ainda calcular métricas conhecidas de separação de interesses CDC e CDO para o código orientado por objetos e estimar o valor dessas métricas caso o sistema venha a ser migrado para aspectos. Esses cálculos serão feitos automaticamente sem nenhuma alteração no sistema e baseado em informações coletadas diretamente

do código fonte orginal. Essa ferramenta deve ser utilizada no ambiente de desenvolvimento Eclipse e avalia projetos desenvolvidos na linguagem Java.

- Com o apoio das métricas QD e SR e da ferramenta *ConcernMetrics*, será feita uma análise quantitativa em três sistemas de médio porte. Esta análise será feita através do resultado do cálculo das métricas QD e SR utilizando-se a ferramenta *ConcernMetrics*, baseada no código orientado por objetos. Esses mesmos cálculos serão feitos manualmente diretamente no código fonte já refatorado para aspectos. Os resultados serão comparados gerando uma relação entre o grau de quantificação dos sistemas e o resultado da refatoração para aspectos.

Os resultados obtidos nesta pesquisa podem ser úteis aos mantenedores de software para a identificação e modelagem de interesses transversais, para uma medida quantitativa dos requisitos transversais do sistema, assim como a medida da redução do espalhamento do interesse transversal através de aspectos. As informações obtidas através das métricas propostas poderão ser utilizadas como fatores para tomada de decisões quando se avalia a refatoração de um sistema para aspectos.

1.3 Estrutura da Dissertação

Nesta seção, apresenta-se a organização do conteúdo desta dissertação. No Capítulo 2, realiza-se uma revisão da literatura diretamente relacionada ao desenvolvimento desta dissertação. Essa revisão inclui Desenvolvimento Orientado por Aspectos, Conceitos gerais de Métricas de Software, Métricas de Software Orientado por Objetos e por Aspectos, Avaliações de Sistemas Orientados por Aspectos e Ferramentas para Automatização do Cálculo de Métricas de Software.

No Capítulo 3 são definidas as métricas QD e SR, que são a proposta desta dissertação. Inicialmente é apresentado um estudo quantitativo de sistemas refatorados para aspectos, a seguir as definições formais das métricas QD e SR e finalmente exemplos da aplicação dessas métricas.

No Capítulo 4 é detalhado o projeto e implementação da ferramenta *ConcernMetrics*, como interface, leitura do código fonte e algoritmos de decisão. Essa ferramenta possibilita a montagem do *Modelo de Concerns*, e baseado neste modelo, efetua os cálculos das métricas QD e SR. Além disso, calcula as métricas CDC e CDO para o código orientado por objetos e estima o valor dessas métricas caso o sistema seja refatorado para aspectos.

No Capítulo 5 é feita uma avaliação manual da aplicação das métricas QD e SR em sistemas reais analisando seus resultados e os comparando com resultados desses mesmos sistemas já em uma versão orientada por aspectos. Ainda nesse capítulo são apresentados estudos da utilização da ferramenta *ConcernMetrics* em sistemas reais, e comparados os resultados obtidos pela ferramenta com resultados calculados manualmente nos códigos orientados por objetos e orientados por aspectos para esses sistemas.

No Capítulo 6 será apresentada a conclusão desta dissertação, onde são apresentadas as principais contribuições, comparações com trabalhos relacionados e ainda algumas propostas para trabalhos futuros com o intuito de dar continuidade à pesquisa desenvolvida.

2 REVISÃO DA LITERATURA

Neste capítulo, serão apresentados os principais conceitos, tecnologias e trabalhos relacionados ao tema desta dissertação. Na Seção 2.1, é apresentado o paradigma de programação orientada por aspectos, na Seção 2.2 são descritos os principais conceitos da linguagem de programação AspectJ, na Seção 2.3 são apresentados os conceitos gerais de Métricas de Software, na Seção 2.4 e 2.5 são descritas respectivamente Métricas de Software Orientado por Objetos e Métricas de Software Orientado por Aspectos, na Seção 2.6 são expostos alguns estudos de avaliação de software orientado por aspectos e na Seção 2.7 serão apresentadas ferramentas de automação de cálculo de métricas de software. Por fim, na Seção 2.8, são apresentadas as considerações finais da Revisão da Literatura.

2.1 Programação Orientada por Aspectos

O paradigma de programação orientada por objetos (POO) permite modelar um software conforme a visão do homem sobre o mundo, por meio de objetos que podemos categorizar, descrever, organizar, manipular e relacionar (PRESSMAN, 2005). Segundo Pressman (2005), o paradigma de POO é muito mais que um paradigma de programação, é uma visão completa de engenharia de software, que traz como principais vantagens a possibilidade de reutilização, agilidade de desenvolvimento, a manutenabilidade e qualidade do software. Por meio do paradigma de POO, é possível implementar os requisitos funcionais e não funcionais de um sistema. Os requisitos funcionais são os serviços que o sistema deve oferecer, como deve reagir a entradas e saídas específicas. Os requisitos não funcionais são restrições sobre os serviços ou funções oferecidas pelo sistema, como segurança, performance, log, entre outros (SOMMERVILLE, 2006).

A POO apresenta grande eficácia para a implementação da maioria dos requisitos funcionais utilizando classes, métodos e atributos. Porém, nem sempre consegue implementar com sucesso requisitos não-funcionais e até alguns requisitos funcionais específicos, porque esses são requisitos transversais, ou seja, estão distribuídos por diversas classes

do sistema, ocasionando o espalhamento e entrelaçamento do código (KICZALES et al., 1997; SOMMERVILLE, 2006). Um requisito se encontra espalhado pelo código do sistema quando sua implementação está distribuída em diversos pontos, podendo ocorrer quando há diversas chamadas praticamente idênticas ou há a implementação em diversos pontos do mesmo interesse. Um requisito se encontra entrelaçado quando o código do requisito transversal está implementado juntamente com o código fonte do requisito funcional (LADDAD, 2003; KICZALES et al., 1997; SOMMERVILLE, 2006).

Esses interesses, por estarem espalhados ou entrelaçados pelas classes, podem ocasionar diversas implicações durante o desenvolvimento, manutenção e evolução do sistema (KICZALES et al., 1997). Como exemplos pode-se citar a dificuldade de rastreamento de interesses, baixa produtividade, baixo grau de reúso, baixa qualidade interna com baixa coesão modular, alto grau de acoplamento de módulos e também baixa extensibilidade (TIRELO et al., 2004; EADDY et al., 2008), além de que, conforme o estudo de Eaddy et al. (2008), pode proporcionar uma abertura a geração de defeitos no projeto. A Figura 1 representa os requisitos transversais espalhados e, na Figura 2, os requisitos transversais entrelaçados pelos módulos do sistema.

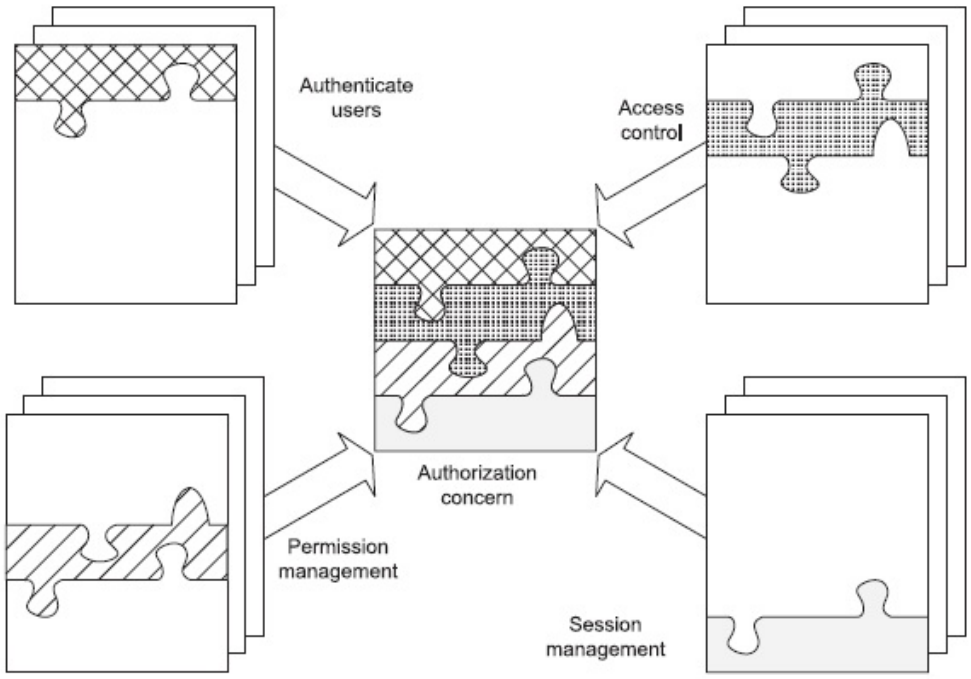


Figura 1: Exemplo de requisito transversal espalhado por diversos módulos (LADDAD, 2003).

A programação orientada por aspectos (POA), proposta por Kiczales e Hilsdale

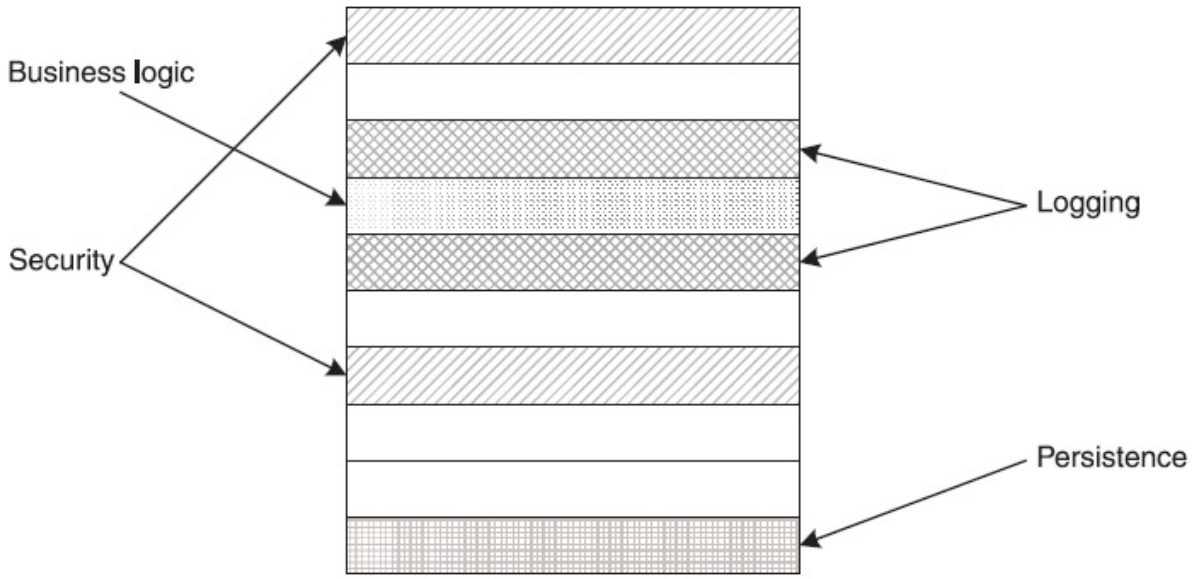


Figura 2: Exemplo de requisito transversal entrelaçado ao código fonte do interesse funcional (LADDAD, 2003).

(2001), Kiczales et al. (1997), surgiu com o intuito de atender a necessidade de modularização dos requisitos que não sejam adequadamente implementados por linguagens orientadas por objetos (MENDHEKAR; KICZALES; LAMPING, 1997). O paradigma de POA propõe que o desenvolvimento de software orientado por aspectos torne possível implementar interesses, tanto transversais quanto não-transversais, de forma modularizada, ou seja, é possível retirar a chamada do interesse do código base e confiná-lo em uma única unidade chamada aspecto (KICZALES et al., 1997; SOMMERVILLE, 2006). A POA traz como principais vantagens a modularização de requisitos transversais, reutilização e extensibilidade de código, facilidade de manutenção e redução do espalhamento e entrelaçamento de código (KICZALES et al., 1997; IRWIN et al., 1997; SOMMERVILLE, 2006).

A implementação de interesses transversais através de aspectos pode ocorrer de duas formas: (1) *Static crosscutting* ou transversalidade estática, que permite alterações na hierarquia de classes e introdução de métodos e atributos em classes por meio de declarações inter-tipo (*inter-type declarations*); e (2) *Dynamic crosscutting* ou transversalidade dinâmica, que possibilita a alteração no comportamento de um programa em pontos específicos de sua execução (LADDAD, 2003).

A POA utiliza-se de uma unidade de modularização – aspectos – para confinar os interesses transversais, os pontos específicos onde os interesses transversais devem ser introduzidos são chamados de *join points*. Os *join point shadows*, ou sombra de ponto de junção, correspondem estaticamente ao trecho de código responsável pela ativação de um ponto de junção (HILSDALE; HUGUNIN, 2004). Os *pointcuts*, por sua vez contêm as regras para agrupar as chamadas dos *join points*, o código do interesse transversal que deve ser inserido em cada *join point* está implementado nos *advices*, estes podem ser inseridos antes (`before`), durante (`around`) ou depois (`after`) dos *join points* (LADDAD, 2003; SOMMERVILLE, 2006; TIRELO et al., 2004). O compilador de aspectos é denominado *weaver*, este tem por função realizar o processo de costura de código (*weaving*), introduzindo o código dos interesses nos *join points* especificados (LADDAD, 2003; SOMMERVILLE, 2006; TIRELO et al., 2004).

2.2 AspectJ

AspectJ é uma linguagem de programação orientada por aspectos proposta nos trabalho de Kiczales e Hilsdale (2001) e Kiczales et al. (1997), é atualmente a mais utilizada na implementação de programação orientada por aspectos em Java (APEL, 2010; GRADECKI; LESIECKI, 2003). AspectJ permite a implementação de requisitos transversais por meio da transversalidade estática e dinâmica (LADDAD, 2003).

A transversalidade estática em AspectJ é tratada por meio da cláusula `declare parents`, que possibilita a introdução de membros em classes e interfaces ou a alteração de uma hierarquia de classes, entre outras alterações nos componentes estáticos do programa. AspectJ porém tem um foco maior no tratamento da transversalidade dinâmica, sendo que a partir desta é possível a exposição de pontos de um programa que podem ser interceptados e a definição de quais ações serão associadas a esse ponto (LADDAD, 2003). Assim como é o foco desta dissertação, as demais informações sobre AspectJ serão referentes ao tratamento da transversalidade dinâmica.

Em AspectJ é possível selecionar os pontos de junção relativos aos eventos de execução utilizando-se o operador `execution` e chamadas de métodos através do operador `call`, mais a parametrização da assinatura do método em ambos os casos. No caso de tratamento de chamadas de construtores utiliza-se, além do operador `call`, também o operador `new` no lugar do nome do método e sem tipo de retorno (GRADECKI; LESIECKI, 2003; LADDAD, 2003). Por exemplo, o *pointcut* para interceptar as chamadas do método `f(int)` da classe `Point` terá a construção `call(void Point.f(int))`, a interceptação

do construtor da classe `Point` terá a construção `call(Point.new(int))` (TIRELO et al., 2004).

Os pontos de junção, responsáveis por leitura e escrita de atributos da classe, possibilitam a captura destes através dos operadores `get` para leitura e `set` para escrita, ambos parametrizados com a assinatura do campo (TIRELO et al., 2004; GRADECKI; LESIECKI, 2003; LADDAD, 2003). Por exemplo, a construção `get(Point.x)` representa os acessos de leitura do campo `x` da classe `Point`, assim como a construção `set(Point.x)` representa os acessos de escrita do campo `x` (TIRELO et al., 2004).

A interceptação de tratadores de exceções a partir de aspectos se dá através do operador `handler`, o qual permite definir *join points* que tratam a execução dos blocos `catch` (GRADECKI; LESIECKI, 2003; LADDAD, 2003). Como exemplo podemos citar o tratamento da exceção `NumberFormatException` a qual será definido através do operador `handler(NumberFormatException)` (TIRELO et al., 2004).

Após a definição dos pontos de junção serão inseridos nesses pontos os *advices*, estes definem como interferir nos *join points* através de `before`, `after` e `around`, e qual o código do interesse que deve ser executado. Uma regra `before` é executada imediatamente antes da execução do *join point* e a regra `after` é executada imediatamente após, a regra `around` define uma execução que será executada no lugar do *join point*, podendo definir se a execução continuará normalmente ou não (GRADECKI; LESIECKI, 2003; LADDAD, 2003).

Com AspectJ é possível modularizar os requisitos transversais de sistemas Java através de aspectos, porém AspectJ dá somente as ferramentas para essa implementação. Os arquitetos de software e programadores precisam conhecer tanto da linguagem quanto do sistema para tomar as decisões inerentes a quais requisitos devem ser modularizados, quais requisitos serão implementados por um mesmo *advice* e até que ponto a refatoração desse interesse transversal apresenta vantagens ao sistema como um todo, ou seja, é necessário o conhecimento técnico aliado a uma análise e decisão de projeto para um bom resultado da refatoração para aspectos de um software.

2.3 Métricas de Software

Medição é um processo imprescindível para o controle e avaliação de projetos em todas as áreas, na engenharia de software esse processo também não é diferente (PRESSMAN, 2005), principalmente com a crescente necessidade de melhoria e controle constante dos projetos de software. Segundo Park, Goethert e Florac (1996), as quatro razões

para medir software são caracterizar, avaliar, prever e melhorar. Conforme Fenton e Neil (1999), o foco da medição é fornecer informações para apoiar gerencialmente a tomada de decisão durante o processo de desenvolvimento, possibilitar a melhoria contínua, auxiliar na estimativa, controle de qualidade e avaliação do software (PRESSMAN, 2005).

Porém, mesmo sendo tão importante e tendo uma gama de estudos na área, as métricas de software não são efetivamente utilizadas (FENTON; NEIL, 1999). Isso talvez se justifique porque em muitas empresas os processos não são organizados ou não estão maduros o bastante para fazer o uso de medições (SOMMERVILLE, 2006). Quando a medição é feita nem sempre seguem boas práticas de medição, o que pode impactar no resultado final, além disso, métricas serão sempre uma sobrecarga em projetos de software, essa sobrecarga está em torno de 4 a 8% do projeto (HALL; FENTON, 1997). Segundo Kaner, Member e Bond (2004), pode-se interpretar esses fatos como prova da imaturidade e falta de profissionalismo do campo ou da resistência ao alto custo dos programas de métricas. Em outros casos porém, os programas de métricas são rejeitados, porque sua aplicação pode trazer resultados negativos ao invés de positivos ao projeto, isso ocorre porque muitas vezes nos projetos de software não se sabe ao certo o que medir e se o que está sendo medido é o que se deseja realmente.

O processo de medição se preocupa em atribuir um valor numérico ou simbólico para algum atributo de um produto de software ou um processo de software de acordo com regras claramente definidas (SOMMERVILLE, 2006; FENTON, 1994). A partir da coleta desses valores é possível aplicar métricas, processo que, simplificando, consiste em relacionar esses valores para gerar resultados.

As métricas de software podem ser tanto de controle, que são normalmente associadas com os processos de software, como por exemplo o esforço médio e tempo necessário para reparar defeitos relatados, ou métricas de previsão, que são associadas a produtos de software, por exemplo a complexidade de um módulo e o número de atributos (SOMMERVILLE, 2006). As métricas de previsão que estão diretamente relacionadas com as características do software em si são as métricas de produtos. Estas se dividem em: (i) métricas dinâmicas, que são coletadas durante a execução do software, como por exemplo a quantidade de erros gerados, número de acessos, processamento, etc; (ii) métricas estáticas, que são coletadas diretamente no sistema, como o projeto, código fonte ou documentação (SOMMERVILLE, 2006).

Para um processo de medição não existem regras pré-definidas, porém alguns estudos sobre o assunto foram feitos com o intuito de estabelecer conceitos para direcionar

(PARK; GOETHERT; FLORAC, 1996), analisar (EJIOGU, 1991), caracterizar (ROCHE, 1994) e definir (EJIOGU, 1991) o processo de medição, formulação e análise da medição. Em seu estudo sobre Métricas de Software e Princípios de Medição, Roche (1994) define alguns princípios para medição de software. Conforme esse estudo, um processo de medição pode se dividir em (1) formulação da medição, (2) coleta e (3) análise dos dados, (4) interpretação dos resultados e (5) *feedback* dos resultados encontrados.

Há ainda alguns processos definidos para garantir a validação de métricas de software, pois métricas são úteis apenas se forem caracterizadas efetivamente e validadas de forma que seu valor seja aprovado (PRESSMAN, 2005). Desta forma devem aderir à ciência da medição para que ganhem ampla aceitação e validade (FENTON; NEIL, 1999).

Para uma eficiente formulação de métricas, Roche (1994) define alguns princípios: (1) Os objetivos da medição devem ser estabelecidos antes da coleta de dados, (2) a definição da métrica deve ser clara e inequívoca, (3) métricas devem ser construídas dentro do domínio da aplicação e (4) métricas para serem eficientes devem ser adaptadas para determinados produtos e processos.

Com o intuito de validar métricas, existem atributos que essas devem possuir, conforme os estudos de Pressman (2005) e Ejiogu (1991): (1) Uma métrica deve ser simples e computável. (2) Uma métrica deve ser consistente no uso de unidades e dimensões e deve sempre estar dentro de um intervalo significativo, por exemplo seu valor deve estar dentro do intervalo 0 e 1 . (3) Uma métrica deve ter características empíricas e intuitivas, por exemplo seu valor deve aumentar ou diminuir de acordo com a presença maior ou menor do atributo que está sendo avaliado pela métrica. (4) Uma métrica deve ser amplamente avaliada para então ser publicada e utilizada para tomar decisões. (5) Uma métrica deve medir o fator de interesse independente de outros fatores . (6) Uma métrica deve ser independente da linguagem de programação.

As características citadas atendem a formulação, caracterização e validação das métricas, mas segundo Pressman (2005) a coleta e análise são as atividades que guiam o processo de medição. Para direcionar a essas duas atividades, Roche (1994) sugere algumas diretrizes: (1) Sempre que possível a coleta de dados e análise devem ser automatizadas; (2) técnicas estatísticas válidas devem ser aplicadas para estabelecer relações entre os atributos internos do produto e as características externas de qualidade; (3) deve-se garantir a independência dos fatores utilizados na formulação da métrica; (4) diretrizes e recomendações interpretativas devem ser estabelecidas para cada métrica; (5) métricas exigem validações de construção adequadas e deve seguir um método científico.

Em alguns casos, métricas de software não satisfazem à todos os atributos citados nesta seção, porém, não se deve rejeitar essas métricas porque não atendem a um ou dois atributos, afinal podem oferecer entendimento útil e valor importante (PRESSMAN, 2005). Dessa forma, conclui-se que os conceitos apresentados direcionam a definição e análise de métricas de software, sendo na maioria das vezes atendidos pelas métricas, mas a análise de qualidade desta métrica deve ser feita também a partir de sua aplicabilidade no mundo real e resultados gerados.

O foco desta dissertação é baseado em métricas de produtos de software, mais especificamente métricas estáticas, visto que as métricas propostas têm por interesse prover resultados para ajudar na avaliação da refatoração de sistemas orientados por objetos para aspectos, com base no código original fonte do sistema. Para isso foram utilizadas as teorias de métricas estáticas, pois avaliam o código fonte diretamente, seus métodos e atributos. Além disso, as diretrizes propostas nesta seção serão utilizadas como guia para a proposta das métricas, assim como fatores para a avaliação de sua eficiência.

2.4 Métricas de Software Orientado por Objetos

Os objetivos de uma métrica de software não diferem conforme seu paradigma de programação, as métricas de softwares convencionais têm o intuito de avaliar a qualidade, eficácia e melhoria contínua do software (PRESSMAN, 2005). Ao se comparar as métricas de softwares convencionais e softwares orientados por objetos, é possível assimilar algumas medidas que são comuns entre elas, por exemplo medida de tamanho e complexidade, porém como isso é medido em cada tipo de software que difere as métricas. Além disso, há características que são medidas específicas para projetos orientados por objetos, as quais não se aplicam ou não estão presente nos demais paradigmas.

Conforme Pressman (2005), softwares orientados por objetos são fundamentalmente diferentes de softwares desenvolvidos à partir dos demais paradigmas de programação, por essa razão, as métricas devem ser ajustadas para atender a características distintas como: (1) Localização: em softwares convencionais estão em métodos procedurais, em softwares orientados por objetos estão em classes ou objetos; (2) Encapsulamento: em softwares convencionais se encontram em sub-programas, funções, entre outros, em softwares orientados por objetos o encapsulamento envolve as responsabilidades de uma classe incluindo atributos e operações; (3) Ocultação de Informações: característica presente somente em projetos orientados por objetos, onde é possível ocultar informações dentro de uma classe que não serão acessadas pelos demais objetos do sistema; (4) Herança: em

geral herança não é suportada por linguagens convencionais, na linguagem orientada por objetos porém, é uma característica fundamental; (5) Abstração: característica também predominante das linguagens orientadas por objetos.

As métricas podem ser especializadas no projeto, gerência, em nível de classes e testes do software orientado por objetos (PRESSMAN, 2005), porém o foco desta dissertação é a avaliação dos interesses transversais espalhados e entrelaçados pelo sistema, sendo assim, o detalhamento de métricas de software será focado em métricas específicas de classe. Os conjuntos de métricas de classe mais amplamente referenciados, segundo Pressman (2005), são o conjunto de métricas CK proposto por Chidamber e Kemerer (1994), o conjunto de métricas MOOD proposto por Harrison, Counsell e Nithi (1998) e o conjunto de métricas de Lorenz e Kidd definido por Lorenz e Kidd (1994).

O conjunto de métricas CK, são aplicadas nas classes e têm por objetivo avaliar características fundamentais de sistemas orientados por objetos como complexidade, coesão e acoplamento (CHIDAMBER; KEMERER, 1994).

A métrica *Weighted Methods per Class* (WMC), ou métodos ponderados por classe, do conjunto de métricas CK, conta os métodos e sua complexidade (CHIDAMBER; KEMERER, 1994), obtendo o valor para a complexidade dos métodos de um sistema, assim como uma medida de tamanho da classe (PRESSMAN, 2005).

A métrica *Depth of the Inheritance Tree* (DIT), ou tamanho da árvore de herança, mede o comprimento da árvore de herança da raiz até o nó folha de maior tamanho. Pode-se considerar que, quanto maior o valor de DIT, maior será a dificuldade de prever o comportamento das classes herdeiras e maior será complexidade do sistema (CHIDAMBER; KEMERER, 1994).

A métrica *Number of Children* (NOC), ou número de filhos, é uma métrica que conta o número de classes filhas que herdem imediatamente de uma determinada classe, ou seja, somente as classes filhas no nível abaixo na hierarquia de classes. Essa métrica tem o intuito de medir o grau de reutilização de uma classe (CHIDAMBER; KEMERER, 1994).

A medida de acoplamento de classes para um sistema, do conjunto de métricas CK, é feita a partir da métrica *Coupling Between Objet Classes* (CBO), ou acoplamento entre as classes de objetos. A métrica CBO avalia o acoplamento de uma classe, onde uma classe A está acoplada a uma classe B quando a classe A utiliza métodos ou variáveis da classe B (CHIDAMBER; KEMERER, 1994).

A métrica *Response For a Class* (RFC), ou resposta de uma classe, mede a complexidade de uma classe, que aumenta proporcionalmente ao valor de RFC. A métrica RFC conta o número de métodos do conjunto resposta de uma classe, sendo que o conjunto resposta de uma classe é um conjunto de métodos que podem potencialmente ser executados em resposta a uma mensagem recebida por um objeto da classe (CHIDAMBER; KEMERER, 1994).

Uma medida de coesão por sua vez é feita com a métrica *Lack of Cohesion in Methods* (LCOM), ou falta de coesão em métodos. A métrica LCOM avalia a similaridade entre métodos de uma classe, onde similaridade entre dois métodos é o acesso aos mesmos atributos da classe. Quanto maior a similaridade entre métodos maior será a coesão da classe e menor será o valor de LCOM (CHIDAMBER; KEMERER, 1994).

As métricas MOOD é um conjunto de métricas para sistemas orientados por objetos criado por Harrison, Counsell e Nithi (1998). Estas métricas se baseiam nos métodos e atributos da classe, e têm como objetivo principal medir a herança e acoplamento.

O conjunto de métricas MOOD mede o fator de acoplamento para métodos e atributos de um sistema através da métrica *Coupling Factor* (CF), ou fator de acoplamento. A métrica CF é calculada considerando todos os possíveis conjuntos de pares de classes, isso é feito combinando todas as possíveis duplas de classes e verificando se estão relacionadas, esse relacionamento pode ser pela passagem de parâmetros ou por referência direta de um atributo ou método de uma classe por outra (HARRISON; COUNSELL; NITHI, 1998).

O cálculo para herança de métodos e atributos de uma classe é feita por meio das métricas *Method Inheritance Factor* (MIF), ou fator de herança de métodos, e *Attribute Inheritance Factor* (AIF), ou fator de herança de atributos. Para o cálculo de MIF, conta-se em todas as classes de um sistema, a quantidade de métodos que são herdados. Esse total será dividido pelo total de métodos do sistema. Para cálculo de AIF a regra é semelhante a MIF, porém deve-se contar os atributos, não os métodos de uma classe. As métricas MIF e AIF definem um percentual de métodos e atributos herdados no sistema (HARRISON; COUNSELL; NITHI, 1998; PRESSMAN, 2005).

O conjunto de métricas de Lorenz e Kidd foi proposto por Lorenz e Kidd (1994). Eles propuseram algumas métricas baseadas em classes divididas entre as categorias: (1) tamanho: baseada na contagem de atributos e operações da classe individualmente e na média para um valor do sistema como um todo; (2) herança: avaliação de como as operações são reutilizadas através da hierarquia de classe; (3) características internas:

avaliação de coesão; e (4) características externas: acoplamento e reutilização (PRESSMAN, 2005).

Conforme Lorenz e Kidd (1994), o tamanho de uma classe é medido a partir da métrica *Class Size* (CS). Essa métrica conta o número total de operações e número de atributos que são encapsulados dentro de uma classe. Grandes valores de CS podem significar que uma classe possui grande responsabilidade, o que pode ocasionar baixa reusabilidade, alta complexidade de implementação, manutenção e testes (PRESSMAN, 2005).

A métrica proposta por Lorenz e Kidd (1994) para a medida de herança da classe é a métrica *Number of Operations Overridden by a Subclass* (NOO), ou número de operações substituídas por uma subclasse, ou seja, uma subclasse substitui uma operação herdada por uma especialização da operação. Altos valores de NOO podem indicar problema de modelagem, fraca hierarquia de classes e dificuldade de manutenção e testes do software (PRESSMAN, 2005).

A métrica *Number of Operations Added by a Subclass* (NOA), ou número de operações adicionadas por uma subclasse, mede a especialização de subclasses. É calculada a partir do número de operações privadas e atributos adicionados à subclasse com o intuito de especializá-la (LORENZ; KIDD, 1994). Quando são identificados altos valores de NOA, pode significar o afastamento da abstração da superclasse (PRESSMAN, 2005).

A métrica *Specialization Index* (SI), ou índice de especialização, é uma métrica para prover um índice do grau de especialização das subclasses do sistema, ou seja, mede individualmente o valor de NOO de cada uma das subclasses. Para sua medida é feito o cálculo de NOO multiplicado pelo nível da árvore hierárquica onde se encontra a subclasse, o resultado é dividido pelo número total de métodos da classe.

Através das métricas para software orientado por objetos apresentadas, é possível se obter uma medida direta de tamanho para um sistema a partir do cálculo das métricas WMC e CS por exemplo, outras calculam valores relacionados a herança de classes como DIT, NOC, MIF, AIF, NOO e NOA, assim como provêem valores relacionados ao acoplamento e coesão, entre outros cálculos proporcionados pelas métricas. A partir desses valores, pode-se obter respostas características que são relacionadas a outros fatores como por exemplo complexidade, dificuldade de manutenção, testes e reutilização.

Como exemplo pode-se citar a medida de complexidade, que em geral não é uma medida exata, caso um sistema apresente altos valores de WMC que conta o número de

métodos e sua complexidade e altos valores para métricas de herança de classes como por exemplo DIT e NOC pode-se presumir que o sistema apresenta alta complexidade, o que ocasionará diretamente dificuldade de manutenção e testes. Outro exemplo é o cálculo de métricas de acoplamento como CF e CBO, que, se apresentarem altos valores, representam um alto acoplamento no sistema que pode ocasionar dificuldade de reutilização e problemas na modelagem. Para se obter resultados refinados em sistemas orientados por objetos é necessária a avaliação de uma ou mais métricas e uma análise em conjunto de fatores que os ocasionem.

2.5 Métricas de Software Orientado por Aspectos

As mesmas métricas que foram desenvolvidas para sistemas orientados por objetos, nem sempre podem ser aplicadas a sistemas orientados por aspectos, devido à arquitetura e às necessidades de medição diferenciadas. Isso não quer dizer, porém, que as métricas de software orientado por objetos não possam ser aplicadas, algumas até em sua forma original e outras sofrendo pequenas adaptações, em sistemas orientados por aspectos. Além disso, vêm sendo propostas diversas métricas específicas para medição de softwares orientados por aspectos, algumas delas são apresentadas nesta seção.

No seu estudo sobre o reúso e manutenção de softwares orientados por aspectos, Sant'Anna et al. (2003) definem um quadro para avaliar a reutilização e manutenção de software através de um conjunto de métricas e de um modelo de qualidade. As métricas são centradas na separação de interesses e avaliação de demais atributos como acoplamento, coesão e tamanho.

Neste estudo, Sant'Anna et al. (2003) propuseram os seguintes requisitos que uma métrica adequada para software orientado por aspectos deve satisfazer: (1) medir atributos de software conhecidos como separação de interesses, acoplamento, coesão e tamanho; (2) basear-se tanto quanto possível em métricas tradicionais e métricas de software orientado por aspectos; (3) capturar diferentes dimensões de acoplamento e coesão de software orientada por aspectos; e (4) apoiar a identificação das vantagens e desvantagens na utilização de aspectos em um projeto de software, quando comparado com uma solução orientada por objetos para o mesmo problema. Baseado nesses princípios, o trabalho de Sant'Anna et al. (2003) propõem algumas métricas novas e adaptam métricas tradicionais de software orientado por objetos para software orientado por aspectos.

As métricas propostas por Sant'Anna et al. (2003) para separação de interesses

são *Concern Diffusion over Operations* (CDO), ou difusão de interesses sobre operações, que se baseia na medida de operações (métodos e *advices*) e *Concern Diffusion over Components* (CDC), ou difusão de interesses sobre componentes, que se baseia na medida de componentes (classes e aspectos). A métrica CDO conta o número de operações cuja finalidade principal é contribuir para a implementação de um interesse e conta o número de métodos e *advices* que os acessem por meio de chamadas de métodos ou utilizando-os como parâmetros formais, tipos de retorno, declarações e variáveis locais. A métrica CDC, por sua vez, conta o número de componentes cujo objetivo principal é contribuir para a implementação de um interesse e o número de classes e aspectos que os acessem. A partir de CDC e CDO, é possível desenhar um mapa de componentes (classes e aspectos) e operações (métodos, atributos e *advices*) que um interesse transversal influencia no sistema como um todo.

A métrica *Coupling Between Components* (CBC), ou acoplamento entre componentes (SANT'ANNA et al., 2003), foi derivada da métrica CBO do conjunto de métricas CK (CHIDAMBER; KEMERER, 1994). O cálculo de CBC é feito verificando-se o acoplamento entre componentes, sendo que um componente (classe ou aspecto) A está acoplado a outro componente (classe ou aspecto) B se A utiliza métodos, *advices* ou atributos de B. Essa métrica foi adaptada para medir todas as variações de acoplamento possíveis em sistemas orientados por aspectos (SANT'ANNA et al., 2003). Outra métrica de acoplamento derivada das métricas CK é a métrica *Depth of Inheritance Tree* (DIT), que mede o comprimento máximo de um nó para a raiz da árvore, contando quão baixo a hierarquia de herança de uma classe ou aspecto é declarada. A partir da medida do tamanho da árvore de herança do sistema, é medido o acoplamento e a complexidade do sistema.

A métrica LCOM, do conjunto de métricas CK (CHIDAMBER; KEMERER, 1994), foi estendida por Sant'Anna et al. (2003) para utilização em sistemas orientados por aspectos, resultou na métrica *Lack of Cohesion in Operations* (LCOO), e propõe uma medida similar a LCOM, a medida de falta de coesão entre operações (métodos e *advices*) do aspecto. Quanto maior o valor de LCOO, menor será a coesão. Dessa forma, quanto mais altos valores de LCOO, maior será a dificuldade para reutilização e manutenção.

A métrica *Weighted Operations per Component* (WOC), ou operações ponderadas por componente, estende a métrica WMC do conjunto de métricas CK (CHIDAMBER; KEMERER, 1994). Semelhante a WMC, WOC conta os métodos e *advices* e sua complexidade para os aspectos do sistema. Consequentemente, altos valores de WOC significam maior complexidade do sistema.

Em outro estudo, Garcia et al. (2005) definem a métrica *Concern Lines of Code* (CLC), ou linhas de código do interesse, que tem por objetivo medir as linhas de código de um determinado interesse no código fonte orientado por aspectos e orientado por objetos. A métrica CLC conta o número de linhas de código cujo objetivo principal é contribuir para a implementação de um interesse, mas desconsiderando comentários, linhas em branco e chamadas de *import*. Na versão orientada por objetos, CLC conta a chamada de métodos que contêm os interesses transversais em questão. Na versão orientada por aspectos, porém, a contagem é do aspecto criado para implementar o interesse transversal, incluindo a descrição do *pointcut*, a assinatura, *advice* e o código de implementação do interesse.

No trabalho de Ceccato e Tonella (2004), foi apresentado um estudo onde as métricas CK foram refinadas para que essas sejam utilizadas em sistemas orientados por aspectos. Conforme o estudo, algumas métricas do conjunto de métricas CK podem facilmente ser adaptadas para utilização em sistemas orientados por aspectos, por exemplo, adaptando as medidas das métricas de classes e métodos para aspectos e *advices*. Outras porém, precisam de um maior refinamento, por exemplo a métrica CBO, que mede o acoplamento, foi refinada em 2 métricas orientadas por aspectos. Outras duas novas métricas de acoplamento foram criadas para atender a todas as formas de acoplamento com aspectos e uma nova métrica foi proposta para medir o grau de transversalidade de um aspecto.

A métrica DIT, que mede a complexidade de uma classe a partir da medida do tamanho da sua árvore hierárquica, foi refinada para considerar também aspectos que possam impactar na hierarquia desta classe por meio de chamadas de inter-tipos, considerando os aspectos também quando o cálculo de DIT é feito (CECCATO; TONELLA, 2004). No seu estudo, Sant'Anna et al. (2003) também criaram uma variação da métrica DIT para medição de software orientado por aspectos. A diferença destas duas métricas é que a métrica DIT proposta por Sant'Anna et al. (2003), mede onde o aspecto se encontra na árvore de herança, enquanto a métrica DIT de Ceccato e Tonella (2004), propõe a medida de possíveis alterações na árvore hierárquica através de inter-tipos.

A métrica RFC do conjunto de métricas CK (CHIDAMBER; KEMERER, 1994), também foi refinada para sistemas orientados por aspectos por meio da métrica *Response For a Module* (RFM), ou resposta de um módulo. Similar à métrica RFC, a métrica RFM mede o número de métodos e *advices* que potencialmente serão executados como resposta de um módulo (CECCATO; TONELLA, 2004).

A métrica CBO foi refinada em duas novas métricas, *Coupling on Method Call* (CMC), ou acoplamento em chamadas de métodos, e *Coupling on Field Access* (CFA), ou acoplamento em acesso a atributos. A métrica CMC conta o número de módulos ou interfaces que declarem métodos que possam ser interceptados por aspectos, e CFA considera os atributos ao invés de métodos (CECCATO; TONELLA, 2004).

Duas novas métricas de acoplamento foram definidas, *Coupling on Advice Execution* (CAE), ou acoplamento na execução de *advice*, e *Coupling on Intercepted Modules* (CIM), ou acoplamento na interceptação de módulos. A métrica CAE conta o número de aspectos que contêm *advices* que possivelmente serão chamados por execuções do sistema, CIM por sua vez conta o número de módulos e interfaces que estão explicitamente declarados nos *pointcuts* para serem interceptados (CECCATO; TONELLA, 2004).

A métrica *Crosscutting Degree of an Aspect* (CDA), ou grau de transversalidade de um aspecto, conta o número de módulos afetados pelos *pointcuts* e pela introdução de um determinado aspecto (CECCATO; TONELLA, 2004). A partir da medida de CDA pode-se estimar o impacto global que tem um aspecto sobre os outros módulos.

Foram propostas duas novas métricas de separação de interesses para sistemas orientados por aspectos por Eaddy et al. (2008), (1) *Degree of Scattering Across Classes* (DOSC), ou grau de dispersão entre classes, e (2) *Degree of Scattering Across Methods* (DOSM), ou grau de dispersão entre métodos. A métrica DOSC calcula o grau em que o código do interesse está distribuído entre as classes. Quando o valor de DOSC é igual a 0 (zero) todo o código está em uma classe, quando o valor é igual a 1 (um) o código é dividido igualmente entre todas as classes. A métrica DOSM calcula o grau em que o código do interesse é distribuído através de métodos, varia de 0-1, semelhante ao DOSC.

Outro estudo feito para propor métricas para sistemas orientados por aspectos foi feito por Apel (2010). Neste trabalho foram propostas cinco métricas para tratar aspectos divididos pelos seguintes grupos: homogêneo e heterogêneo, estático e dinâmico, e básico e avançado.

A métrica *Classes, Interfaces, and Aspects* (CIA) conta as classes, interfaces e aspectos de um programa. Pode ser utilizada para avaliar se um aspecto implementa uma parte significante de um sistema. Quanto maior a quantidade de aspectos, maior será considerada sua importância no sistema.

A fim de definir quais interesses são homogêneos ou heterogêneos, Apel (2010) propõe a métrica *Heterogeneous and Homogeneous Crosscuts* (HHC), ou transversalidade

heterogênea e homogênea. Com HHC, é feita uma análise de cada parte de um *advice* e declaração inter-tipo, se o número de *join points* que ele impacta for maior que 1 (um) então é considerado homogêneo, caso contrário, ele é heterogêneo.

A métrica *Code Replication Reduction* (CRR), ou redução de replicação de código, tem por interesse quantificar o benefício da redução do número de linhas de código (APEL, 2010). A métrica CRR conta o número de linhas de código que podem ser reduzidos por meio da implementação de aspectos. Para o seu cálculo é feita a multiplicação do número de linhas dos *advices* homogêneos e declarações de inter-tipos pelo número de *join points* afetados (menos um). Esse valor é definido para todo o sistema e medido no código fonte orientado por objetos, calculando dessa forma a quantidade de linhas que serão reduzidas por meio de aspectos.

A métrica *Static and Dynamic Crosscuts* (SDC), ou interesses dinâmicos e estáticos, conta o número de linhas de código, ou *Lines of Code* (LOC), de declarações de inter-tipos e *advices*. Esse total é comparado com o número de linhas do código fonte original (APEL, 2010). A métrica SDC mostra até que ponto aspectos estendem o código dinâmico do programa, o que não pode ser bem expresso em linguagens orientadas por objetos.

Para definir um interesse básico ou avançado, Apel (2010) propõe a métrica *Basic and Advanced Dynamic Crosscuts* (BAC), ou interesses dinâmicos básicos ou avançados. A métrica BAC avalia as linhas de código associadas a partes de um *advice* básico ou avançado. Foi considerado um *advice* avançado se o *pointcut* envolve mais do que simplesmente uma combinação de execução (ou `call`) e argumentos.

Em seu trabalho, Sant'Anna et al. (2003) propõem duas métricas de separação de interesses, CDO e CDC. Essas provêem informações importantes para modularização dos interesses transversais, possibilitando o mapeamento de classes, aspectos, métodos, atributos e *advices* que um interesse transversal impacta. Nesse trabalho o cálculo das métricas foi feito em projetos já refatorados para aspectos, mas sua aplicação também pode ser feita em sistemas orientados por objetos puramente, porém avaliando somente classes, métodos e atributos, que pode traçar um mapeamento da importância do interesse antes de sua refatoração para aspectos.

Nos trabalhos de Sant'Anna et al. (2003) e Ceccato e Tonella (2004), foram estendidas algumas métricas CK para sistemas orientados por aspectos. No contexto do trabalho de Sant'Anna et al. (2003), que visa a reutilização e manutenção de sistemas orientados por aspectos, aplicar essas métricas em código já refatorados para aspectos

é justificável. Porém, vale ressaltar que as métricas propostas provêem informações de tamanho, acoplamento, coesão e complexidade de sistemas orientados por aspectos, os quais seriam interessantes de se obter previamente à refatoração para aspectos de um sistema, pois são fatores de decisão importantes.

A métrica CDA, proposta por Ceccato e Tonella (2004), informa o número de módulos afetados por um aspecto. Em um outro trabalho, Apel (2010) propõe as métricas HHC, SDC e CRR. A métrica HHC define interesses homogêneos e heterogêneos (embora seu conceito de homogeneidade seja diferente do conceito utilizado nesta dissertação), a métrica SDC mostra até que ponto aspectos estendem o código dinâmico do programa, e a métrica CRR calcula a redução de replicação de código. Essas métricas provêem uma avaliação da importância e o impacto de um aspecto para o sistema, mas nos estudos apresentados foram aplicadas a sistemas já na versão orientada por aspectos.

As métricas DOSC e DOSM, propostas por Eaddy et al. (2008), calculam o grau em que o código do interesse está distribuído entre classes e métodos. Essas métricas são interessantes pois podem ser utilizadas em um sistema que ainda não foi migrado para aspectos, possibilitando uma avaliação do espalhamento dos interesses transversais para uma possível decisão de refatoração. Esses valores vinculados a informações de redução de código, de acoplamento, coesão e complexidade, caso estejam disponíveis para sistemas ainda na versão orientada por objetos, podem prover informações relevantes para a decisão da refatoração, ou não, de um sistema para aspectos.

2.6 Avaliações em Projetos Orientados por Aspectos

As métricas para sistemas orientados por aspectos apresentadas na seção anterior possibilitam a avaliação de atributos específicos em um projeto de software. Nesta seção, serão apresentados alguns trabalhos de avaliação de sistemas orientados por aspectos, os quais realizaram medições de atributos, tais como, tamanho, coesão, acoplamento, complexidade e quantificação. Esses atributos são medidos normalmente por meio da aplicação de métricas de software, em sua maioria, em sistemas já refatorados para aspectos. As avaliações podem ser feitas ainda, a partir da comparação da implementação de um determinado interesse, implementado no código orientado por objetos e também no código orientado por aspectos.

No trabalho de Filman e Friedman (2005), é feita uma avaliação para determinar características distintivas de sistemas passíveis de refatoração para aspectos. A conclusão

neste trabalho é que a quantificação é um fator importante para a caracterização de um sistema "aspectizável". Um interesse é quantificável quando sua declaração tem efeitos em diversos locais do sistema, ou seja, um número grande de *join points*. A partir dessa avaliação, Filman e Friedman (2005) conclui que um sistema com interesses transversais amplamente quantificáveis apresenta uma característica positiva para a decisão de refatoração para aspectos.

Usando as métricas de separação das interesses, acoplamento, coesão e tamanho, métricas essas que foram criadas ou adaptadas das métricas CK para sistemas orientados por aspectos, Sant'Anna et al. (2003) têm investigado o uso de aspectos em uma ampla variedade de domínios e sistemas, incluindo padrões de projeto (GARCIA et al., 2005), manipulação de exceções (FILHO et al., 2006), sistemas de informação baseados na Web (KULESZA et al., 2006) e as linhas de produto de software (FIGUEIREDO et al., 2008).

Em outro estudo, Garcia et al. (2005) utilizaram as métricas definidas por Sant'Anna et al. (2003) para avaliar a implementação de padrões de projeto por meio de aspectos. Foi observado que na maioria dos padrões houve vantagem na utilização de aspectos para modular os interesses transversais, porém alguns padrões apresentaram maior acoplamento, complexidade e tamanho que as soluções orientadas por objetos.

Em seu estudo sobre manipulação de exceções, Filho et al. (2006) fizeram uma avaliação da utilização de aspectos para modularização de exceções e concluiu que é eficiente quando se trata de exceções homogêneas, ou seja, quando um ou poucos *advices* atingem todos os *join points*. Porém, quando se trata de exceções heterogêneas e dependentes do contexto do sistema, podem causar mais danos do que benefícios, pois, pode tornar o código mais complexo e os interesses frágeis.

No estudo de Kulesza et al. (2006), foi apresentada a implementação de um sistema de informação web utilizando aspectos e sua comparação com uma implementação somente orientada por objetos. Constatou-se na versão orientada por aspectos uma diminuição da coesão e aumento de operações e componentes, ou seja, baixa quantificação. Porém, no sistema em geral, a implementação orientada por aspectos apresentou superioridade quanto a linhas de código, separação de interesses, baixo acoplamento e menor complexidade.

No trabalo de Figueiredo et al. (2008), foi apresentado um estudo baseado em linhas de produtos utilizando-se de aspectos. Nesse trabalho foram desenvolvidos dois exemplos de linhas de produto utilizando-se aspectos com o intuito de avaliar a modularidade, estabilidade, manutenabilidade e acoplamento. Foi constatado que, com a utilização de aspectos, houve um aumento da manutenabilidade, estabilidade, modula-

ridade e diminuição do acoplamento. Porém apresentou dificuldade para introdução de novas implementações, o que é uma característica forte em projetos de linhas de produto.

As experiências na refatoração de *features* da Oracle Berkeley DB com aspectos foram relatadas por Kastner, Apel e Batory (2007). Neste trabalho observou-se que os elementos extraídos, em geral, apresentam um pequeno grau de quantificação, ou seja, número reduzido de *advices* que podem afetar mais de um ponto de execução (*join point*). Por exemplo, a partir de 482 *advices* extraídos apenas 7 afetam mais de um ponto comum. Além disso, a maioria dos *pointcuts* definidos estão intimamente ligados ao programa de base e, portanto, são especialmente frágeis para modificações no programa, pois, alterações no programa base afetarão os *advices* e podem ocasionar erros.

No trabalho de Apel (2010) foi apresentada uma análise do uso de AspectJ em onze sistemas, usando as métricas CIA, HHC, CRR, SDC e BAC. O estudo mostrou que 86% do código é considerado orientada por objetos, 12% utiliza mecanismos de base transversal (declarações inter-tipos ou estensões de um método único), e apenas 2% utilizam mecanismos transversais avançados (incluindo *advices* que afetam conjuntos inteiros de *join points*). Uma vez que na maioria dos sistemas avaliados, AspectJ foi usado para extrair características heterogêneas a partir do código fonte, os resultados reforçam as conclusões derivadas da refatoração de Oracle Berkeley DB (KASTNER; APEL; BATORY, 2007).

Em geral, em cada um desses estudos empíricos foram identificados efeitos positivos e negativos da utilização de aspectos, e, na maioria dos estudos, as situações em que eles não recomendam o uso de aspectos pode estar ligado a um reduzido grau de quantificação, onde há o aumento de linhas de código e de componentes. Os trabalhos apresentados exemplificam as avaliações de sistemas orientados por aspectos.

Observa-se que, para que as avaliações apresentadas fossem feitas, foi necessário a refatoração dos sistemas para aspectos. Isso implica em custo e tempo, podendo, caso observe-se que não é interessante a utilização de aspectos, significar prejuízo para o projeto. A incerteza do resultado da refatoração pode ser considerado um limitador para a utilização de programação orientada por aspectos comercialmente. Caso seja possível essa avaliação previamente à refatoração para aspectos, a utilização de aspectos seria mais segura e possivelmente mais disseminada.

2.7 Ferramentas de Automação de Cálculo de Métricas

Conforme relatado na Seção 2.3, em um processo de medição o processo de coleta de dados e análise deve ser, preferencialmente, automatizados (ROCHE, 1994). Na medição em projetos de software normalmente são criadas ferramentas para automatizar parte, ou todo o processo do cálculo de métricas, conforme serão relatados alguns exemplos nesta seção. Os exemplos de ferramentas que serão apresentadas foram selecionadas por apresentarem automatização de métricas de software para avaliação de interesses transversais.

A ferramenta *ConcernMapper*[1] é utilizada como uma ferramenta de apoio para coleta de dados. Foi desenvolvida por Robillard e Weigand-Warr (2005) com duas propostas distintas. A primeira delas é criar uma ferramenta para técnicas avançadas para a separação de interesses; a segunda é disponibilizar uma plataforma para criar, armazenar e consultar mapas de *concerns* para atender a diferentes abordagens.

A ferramenta *ConcernMapper* apoia o desenvolvimento e as tarefas de manutenção de software que envolvam interesses espalhados pelo sistema. Permite aos desenvolvedores organizar e visualizar o código de um projeto vinculado a uma abstração de alto nível, permitindo uma visão modularizada dos interesses transversais de um sistema sem que ele sofra qualquer tipo de alteração estrutural.

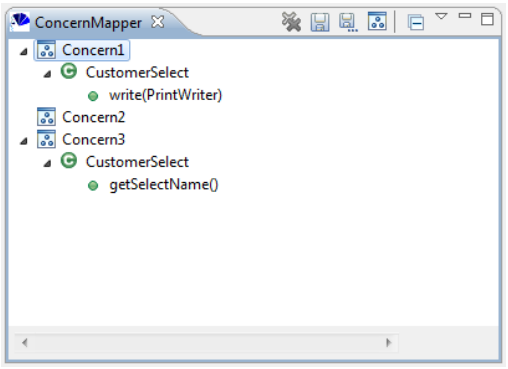


Figura 3: *Concern Maps* da ferramenta *ConcernMapper*.

Através da ferramenta *ConcernMapper* é possível organizar atributos e métodos correspondentes a interesses transversais em uma estrutura em forma de árvore, denominada *Concern Maps*, conforme ilustrado na Figura 2.7. *ConcernMapper* permite montar a árvore de *concerns*, o qual é uma estrutura visual, semelhante por exemplo ao *Package Explorer* da plataforma Eclipse[2]. Para isso, basta selecionar o interesse no *Package Ex-*

[1]http://www.cs.mcgill.ca/ martin/cm/
[2]http://www.eclipse.org

plorer e arrastá-lo até o *concern* no *Concern Maps*, soltá-lo e ele será automaticamente incluído na árvore de *concerns*. Atualmente a ferramenta *ConcernMapper* está disponível em forma de *plugin* do Eclipse e é estendida para criação de outras ferramentas, inclusive a ferramenta *ConcernMetrics* que foi desenvolvida nesta dissertação.

A ferramenta *ConcernTagger*[3] é uma ferramenta desenvolvida a partir da extensão de *ConcernMapper*. A ferramenta *ConcernTagger* foi desenvolvida para automatizar as métricas de software orientado por aspectos, incluindo CDC, CDO, DOSC e DOSM, propostas por Eaddy et al. (2008). Permite ao usuário criar uma hierarquia de *concerns* e associar aos mesmos requisitos, código fonte e bugs, através da interface de *ConcernMapper* e obter o resultado dos cálculos das métricas. Essas métricas são calculadas com base no código fonte orientado por objetos.

A ferramenta *ConcernMorph* é uma ferramenta para automatização de cálculo de métricas de software. A ferramenta implementa diversas métricas, entre elas a métrica de separação de interesses CDC (SANT'ANNA et al., 2003) e as métricas NOA e NOO que contam, respectivamente, o número de atributos e operações de um sistema (KICZALES et al., 1997). Através de *ConcernMorph* é possível a identificação de interesses transversais e a classificação destes em uma série de padrões transversais pré-definidos através da aplicação das métricas de software. Esses padrões foram baseados no projeto de Figueiredo, Whittle e Garcia (2009), onde é definido um grupo de 12 padrões transversais para qualificação de interesses transversais. *ConcernMorph* possui a funcionalidade de identificação de padrões transversais com base em métricas coletadas diretamente a partir do código orientado por objetos, sua interface é estendida da ferramenta *ConcernMapper* para montagem da árvore de *concerns*.

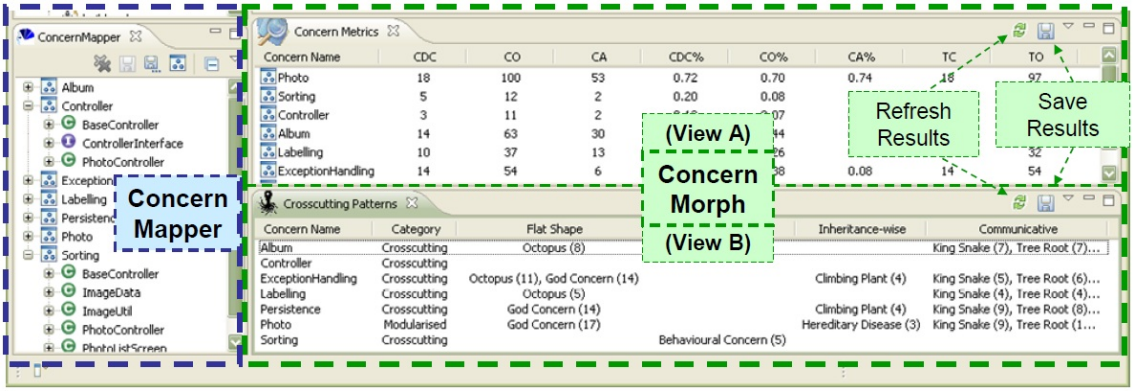


Figura 4: Ferramenta *ConcernMorph* (FIGUEIREDO; WHITTLE; GARCIA, 2009).

[3]http://www.cs.columbia.edu/ẽaddy/concerntagger

A Figura 4 apresenta duas visões da ConcernMorph: Métricas (Visão A) e Padrões Transversais (Visão B), onde são mostrados respectivamente o cálculo das métricas e as instâncias dos padrões transversais encontrados.

2.8 Considerações Finais

Neste capítulo, foi apresentada uma revisão da literatura diretamente relacionada ao desenvolvimento desta dissertação. A revisão apresentada incluiu conceitos básicos sobre o paradigma de programação orientada por aspectos e AspectJ, que é uma linguagem para implementar aspectos em Java. Foram apresentados conceitos de medição e métricas de software, métricas para projetos orientados por objetos e orientados por aspectos. Foram apresentadas ainda ferramentas de automação de cálculo de métricas de software.

A partir da análise dos trabalhos apresentados, nota-se que ainda não é feita uma larga utilização de medição em sistemas orientados por objetos, e ainda em menor escala em projetos orientados por aspectos. Pode-se talvez justificar essa menor utilização em projetos orientados por aspectos pela características das métricas apresentadas, que, para analisar o projeto, normalmente há a necessidade da refatoração do código fonte base para aspectos, o que despende um grande esforço e custo. Foi verificado também uma quantidade limitada de ferramentas para automatizar os cálculos de métricas, visto que esses cálculos são trabalhosos e tediosos, o que pode ocasionar uma maior abertura a erros na sua aplicação manual.

3 MÉTRICAS PARA AVALIAÇÃO DO GRAU DE QUANTIFICAÇÃO

Neste capítulo, serão apresentados alguns estudos de quantificação em sistemas orientados por aspectos e a proposta de duas novas métricas de software. Como validação da proposta desta dissertação, será exposto um estudo sobre uma análise quantitativa utilizando-se métricas de separação de interesse já amplamente utilizadas em sistemas orientados por aspectos. Essas métricas serão aplicadas em sistemas nas versões orientadas por objetos e versões refatoradas para aspectos, a partir do resultado da aplicação das métricas para cada versão, esses serão comparados quantitativamente. Será apresentada a proposta de uma métrica para medir o grau de quantificação de um interesse transversal em um sistema, a métrica *Quantification Degree* (QD), e uma sub-métrica para medir a redução do espalhamento caso os interesses venham a ser modularizados através de aspectos, a métrica *Scattering Reduction* (SR). Serão apresentados ainda alguns exemplos da aplicação das métricas e a análise de cada aplicação.

Na Seção 3.1, serão apresentadas avaliações quantitativas de métricas convencionais de separação de interesses nos sistemas JSpider e JAccounting. Na Seção 3.2 serão apresentadas as definições do *Modelo de Concerns* proposto e a formalização das métricas QD e SR. Na Seção 3.3 serão expostos exemplos da aplicação das métricas propostas em exemplos da aplicação das métricas em sistemas hipotéticos. Na Seção 3.4, por fim, serão apresentadas as considerações finais.

3.1 Avaliações Quantitativas

Será apresentada nesta seção uma avaliação dos sistemas JAccounting[1] e JSpider[2] já refatorados para aspectos. Ambos os sistemas possuem código aberto, versões orientadas por objetos e também versões orientadas por aspectos que foram desenvolvidas por Binkley et al. (2006), com o intuito de ilustrar a utilização de aspectos aspectos para modularizar interesses transversais.

[1] http://jaccounting.dev.java.net
[2] http://j-spider.sourceforge.net

Os aspectos implementados nos sistemas mencionados fazem usos opostos de quantificação. Por exemplo em JAccounting, declarações quantificadas são amplamente utilizadas a fim de modularizar o interesse de controle de transações, por outro lado, a refatoração da preocupação de *log* de JSpider faz o uso mínimo de quantificação. Inicialmente será ilustrado o uso de aspectos para modularização de requisitos transversais, em seguida será apresentada uma avaliação feita nos sistemas JAccouting e JSpider das vantagens da quantificação através da análise da aplicação manual das métricas de separação de interesses transversais CDC e CDO (SANT'ANNA et al., 2003).

Para a avaliação quantitativa apresentada nesta seção, os projetos JAccounting e JSpider foram selecionados por apresentarem um histórico de utilização em avaliações de refatoração para aspectos (MALTA; OLIVEIRA; VALENTE, 2009; MALTA; VALENTE, 2009). Além disso, esses sistemas possuem versões orientadas por objetos e versões orientadas por aspectos (BINKLEY et al., 2006), o que possibilita a comparação quantitativa do resultado das métricas aplicadas em ambas as versões.

No trabalho de Binkley et al. (2006), os sistemas JAccounting e JSpider foram selecionados para serem os estudos de caso na aplicação de sua proposta de passos para refatoração de projetos orientados por objetos e de sua ferramenta de automatização desse processo chamada *AOP-Migrator*. Nos trabalhos Malta, Oliveira e Valente (2009) e Malta e Valente (2009), são apresentadas propostas de transformações de código para extração de interesses transversais por meio de aspectos. Esses trabalhos utilizam os sistemas JAccounting e JSpider como estudos de caso, propondo modificações no código para a melhor refatoração possível dos respectivos interesses transversais *transaction* e *logging*. No estudo de Ceccato (2008), os sistemas JAccounting e JSpider também são utilizados para validação de seu trabalho sobre identificação e transformação de código para refatoração de interesses transversais para aspectos.

O sistema JAccounting é um sistema web contábil de médio porte e tem como principais processos a automatização de faturamento e controle de contas (BINKLEY et al., 2006). O interesse transversal selecionado no trabalho de Binkley et al. (2006), para exemplo de refatoração em JAccounting, foi o controle de transações *transaction*. Esse interesse se encontra espalhado pelo sistema em 44 pontos de junção e sua implementação está entrelaçada aos interesses funcionais do sistema. O interesse transversal *transaction* é o controle de transações que deve ser feito a cada acesso ao banco de dados, conforme exemplo na Figura 5. Ao iniciar o acesso deve ser chamado o método `beginTransaction()` (linha 01), no fim o método `commit()` (linha 12), e, caso ocorra alguma exceção, deve

chamar o método `rollback()` (linha 07).

```
01: tx= sess.beginTransaction();      // starts a transaction
02: try {
03:   ...                             // database operations
04: }
05: catch (...) {                     // handles database exceptions
06:   if (tx != null)  {
07:      tx.rollback();               // performs a rollback
08:      tx= null;
09:   }
10: }
11: finally {
12:   if (tx != null) tx.commit();    // commits
13: }
```

Figura 5: Chamada do interesse transversal *transaction* de JAccounting.

O sistema JAccounting apresenta um alto grau de quantificação do interesse transversal *transaction*, ou seja, foi possível fazer a modularização do interesse transversal através de um número reduzido de *advices* e que atenda a todos os *join points*. Isso porque o interesse transversal *transaction* apresenta um comportamento homogêneo, ou seja, suas chamadas são idênticas (MALTA; OLIVEIRA; VALENTE, 2009; MALTA; VALENTE, 2009). A partir dessa avaliação, constatou-se que as 44 chamadas do interesse *transaction* foram modularizadas em um único aspecto e em três *advices*, conforme mostrado na Figura 6, o *advice* `p0` (linha 04), o *advice* `p1` (linha 08) e o *advice* `p2` (linha 14). Avaliando quantitativamente, esse resultado pode ser considerado um bom exemplo do benefício da quantificação.

O sistema JSpider, por sua vez, é um sistema de médio porte que funciona como um robô que permite recuperar e validar páginas Web. Na versão orientada por objetos desse sistema, *logging* é um interesse transversal, estando sua implementação espalhada e entrelaçada por diversas classes, exemplos de chamadas do interesse são mostrados na Figura 7. Na versão orientada por aspectos, implementada em AspectJ, o código de *logging* foi devidamente modularizado por meio de aspectos.

Há porém casos que os interesses transversais não são homogêneos, como por exemplo as chamadas do método `info()` mostrado na Figura 7, que implementam o interesse transversal *logging* do sistema JSpider. As chamadas de `info()` não são homogêneas pois apresentam variações nos parâmetros, embora sejam chamadas do mesmo método.

Outros métodos que implementam o interesse *logging* de JSpider apresentam essa

```
01: aspect TransactionManagement {
02:   ...
03:   // pointcut p0 captures when database sessions must be opened
04:   after(): p0() {
05:     tx = sess.beginTransaction();
06:   }
07:   // pointcut p1 captures when database exceptions must be handled
08:   before(): p1() {
09:     if (tx != null)  {
10:        tx.rollback(); tx= null;
11:     }
12:   }
13:   // pointcut p2 captures when database sessions must be closed
14:   before(): p2() {
15:     if (tx != null) tx.commit();
16:   }
17: }
```

Figura 6: Aspecto com modularização do interesse *transaction* em JAccounting.

```
01: log.info("Loading " + pluginCount + " plugins.");
02: ...
03: log.info("Loading plugin configuration '" + pluginInstance + "'...");
04: ...
05: log.info("Plugin class ... not found");
06: .....
07: log.info("Plugin uses local event filtering");
08: ...
09: log.info("Plugin not configured for local event filtering");
10: ...
11: log.info("Plugin Name    : " + plugin.getName());
```

Figura 7: Chamada do interesse transversal *Logging* de JSpider.

mesma característica, interesses heterogêneos. Dessa forma não é possível em JSpider modularizar o interesse transversal *logging* em um único, ou mesmo em poucos, *advices*, sendo necessário criar um *advice* para atender a cada variação de passagem de parâmetro dos métodos que implementam o interesse `logging`. No sistema JSpider foram necessários 176 *advices* para modularizar as 190 chamadas do interesse. Esse é um exemplo de um grau de quantificação baixo, onde os *advices* atingem poucos *join points*.

Os sistemas JAccounting e JSpider apresentam características de modularização dos interesses transversais distintas. Os interesses do sistema JAccounting apresentam uma característica homogênea, enquanto no sistema JSpider os interesses transversais

apresentam características heterogêneas. Esses dois projetos, em conjunto, apresentam características que devem ser examinadas durante o processo de migração de sistemas para aspectos e possibilitam uma avaliação eficiente das propostas.

Para o desenvolvimento da análise quantitativa nos sistemas JAccounting e JSpider, foram aplicadas manualmente métricas de separação de interesses e feita uma análise da comparação dos resultados das métricas. Na Tabela 1 são apresentadas informações relevantes dos sistemas já na versão orientada por aspectos, como número de aspectos, *join point shadows* (JPS) e *advices*.

	JAccounting	JSpider
Aspects	1	1
JPS	44	190
Advices	3	176

Tabela 1: Informações sobre versão orientada por aspectos dos sistemas JAccounting e JSpider.

Uma das métricas aplicadas foi a CLC, que conta o número de linhas de código que implementam um determinado interesse (GARCIA et al., 2005). No sistema JAccounting, o valor de CLC na versão orientada por objetos é 105, após sua refatoração para aspectos o valor de CLC foi reduzido para 72, ou seja, reduziu o número de linhas que implementam o interesse em 32%. Isso se justifica pela modularização do interesse *transaction* em um único aspecto e em somente três *advices*, pois seus interesses são homogêneos em todo o sistema.

No sistema JSpider o valor de CLC na versão orientada por objetos é de 244, na versão refatorada para aspectos aumentou para 2400, havendo um aumento de 884% no valor da métrica. Basicamente, essa diferença pode ser explicada pelo número significativo de linhas extras necessárias para implementar os aspectos para atender às variações do interesse *logging*. Como exemplo, na Figura 8, para modularizar uma única chamada de `error()` (linha 6), seis linhas extras de código foram utilizadas: quatro linhas para declarar o *pointcut* (linhas 1-4) e duas linhas que delimitam o corpo do *advice* (linha 5 e linha 7).

A métrica CDO também foi utilizada para avaliação dos sistemas JAccounting e JSpider. A métrica CDO conta o número de métodos e *advices* que contribuem para a implementação de um interesse transversal e outros métodos e *advices* que os acessam. Outra métrica aplicada foi a métrica CDC, que é semelhante à métrica CDO, porém em CDC é contado o número de classes, não de métodos, que contribuem na implementação

```
1: pointcut p_27(DBUtil _this, SQLException e ):
2:   this(_this)
3:   && execution(void DBUtil.sqlException(SQLException))
4:   && args(e);
5:   before(DBUtil _this, SQLException e): p_27(_this, e) {
6:      _this.log.error("SQL Exception during JDBC Connect", e);
7:  }
```

Figura 8: Exemplo da descrição do *pointcut* e implementação do *advice* em JSpider

de um interesse e outras classes e aspectos que acessam as mesmas.

Ao aplicar a métrica CDO em JAccounting foi obtido o valor 11 para a versão orientada por objetos e 7 para a versão orientada por aspectos. Através da refatoração para aspectos foi obtida uma redução de 36% no valor de CDO, ou seja, houve a redução de 36% dos métodos ou *advices* que implementam ou acessam o interesse *transaction*.

Em JSpider, porém, foi observado um aumento significativo do valor de CDO em 125%, sendo o valor na versão orientada por objetos igual a 108, enquanto que na versão refatorada para aspectos aumentou para 243. Esse aumento é resultado do baixo grau de quantificação dos interesses de JSpider, isso porque, em sua maioria, uma chamada de *logging* é movida para um *advice*, apenas movendo a chamada de lugar no sistema de um método para dentro de um aspecto.

Um outro fator é que no sistema JSpider, na versão orientada por objetos, às vezes um único método tem diversas chamadas de métodos que implementam o interesse *logging*, neste caso é contado somente uma vez. Na versão orientada por aspectos porém, as chamadas são implementadas em *advices* diferentes, aumentando conseqüentemente o valor de CDO, pois cada um desses *advices* é contado separadamente.

A aplicação da métrica CDC no sistema JAccounting obteve o valor 10 na versão orientada por objetos e o valor 3 para a versão orientada por aspectos, diminuindo assim o valor de CDC em 70%. Semelhantes ao sistema JAccounting, o sistema JSpider também teve uma redução significativa, diminuiu o valor de CDC em 91%, passando o valor de CDC da versão orientada por objetos de 39 para 3 da versão refatorada para aspectos.

A redução no valor de CDC para ambos os sistemas ocorreu, porque para a implementação de *advices*, é necessário apenas um aspecto, sendo que quando se utiliza mais de um é por decisão dos desenvolvedores, dessa forma diminuindo a quantidade de componentes que implementam e acessam determinado interesse. Os resultados da aplicação

das métricas CLC, CDO e CDC em JAccounting e JSpider são apresentados na Tabela 2.

Métricas	**JAccounting**			**JSpider**		
	OO	OA	%	OO	OA	%
CLC	105	72	-32%	244	2400	+884%
CDO	11	7	-36%	108	243	+125%
CDC	10	3	-70%	39	3	-92%

Tabela 2: Resultado da aplicação das métricas CLC, CDO e CDC na versão orientada por objetos e orientada por aspectos dos sistemas JAccounting e JSpider.

A partir da avaliação quantitativa dos exemplos apresentados, pode-se presumir que quanto maior a quantificação, menor será o número de linhas de código necessárias para implementar interesses transversais (CLC), também contribui para reduzir o número de *advices* necessários para atender a todos os *join points* (CDO), não impacta diretamente na redução do número de aspectos necessários no processo de refatoração de um determinado sistema (CDC), porém com a redução da quantidade de *advices* consequentemente o tamanho de tais aspectos diminui.

3.2 Definições das Métricas de Avaliação do Grau de Quantificação

Nesta seção será apresentada a proposta da métrica de quantificação, *Quantification Degree* (QD), e da sub-métrica de medida da redução do espalhamento, *Scattering Reduction* (SR). Essas métricas foram desenvolvidas com base nos estudos apresentados na Seção 3.1 deste capítulo e têm a proposta de definir valores para o grau de quantificação de um interesse no sistema, sendo uma medida para ajudar aos mantenedores de software a decidir se uma refatoração para aspectos irá prover resultados positivos em um sistema, isto ainda na sua versão orientada por objetos. Inicialmente será apresentado o *Modelo de Concerns*, depois serão definidas as métricas QD e SR e por fim serão apresentados exemplos de sua aplicação e análise dos resultados.

3.2.1 Modelo de Concerns

O *Modelo de Concerns* definido neste trabalho foi inspirado em um modelo já existente proposto por Eaddy et al. (2008) para relacionar interesses transversais com defeitos. O modelo organiza hierarquicamente os interesses transversais em uma árvore, onde os nós internos são os interesses principais e os nós folha são os métodos que implementem esse interesse.

Por exemplo, no estudo de caso JAccounting, o interesse transversal é o controle de transações (*transaction*), que é implementado através dos seguintes métodos, que serão os nós folha do modelo: `begin`, `commit` e `rollback`. No estudo de caso JSpider o interesse transversal analisado é o *logging*, que foi definido como o nó principal e os métodos que o implementam serão mapeados como nós folha do modelo: `debug, error, fatal, info` e `warn`. Os modelos dos estudos de caso JAccounting e JSpider são mostrados nas Figuras 9 e 10 respectivamente.

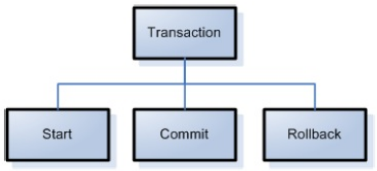


Figura 9: *Modelo de Concerns* do interesse transversal *transaction* do estudo de caso JAccounting.

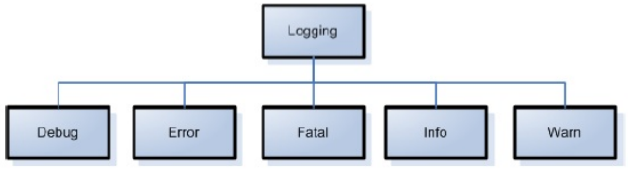


Figura 10: *Modelo de Concerns* do interesse transversal *logging* do estudo de caso JSpider.

Para a aplicação das métricas QD e SR serão considerados os interesses atômicos, isto é, os métodos que correspondem ao nó folha do *Modelo de Concers*. Essa definição se justifica pelo motivo que, quando um interesse é implementado por dois métodos distintos dificilmente serão modularizados em uma única instrução. Por exemplo, o interesse transversal de JAccounting *transaction* dificilmente poderá ser modularizado em um único *advice*, visto que possui os seguintes nós folha `beginTransaction`, `commit` e `rollback` e entre essas chamadas pode haver códigos diversos do sistema. Porém é plausível considerar que todas as chamadas do método `rollback` possam ser confinados em um único *advice*.

Além disso, as preocupações podem ser mapeadas para elementos do programa estático, ou seja, nós da *Abstract Syntax Tree* (AST) do programa de destino. Desta forma assume-se que há uma relação que possibilita o mapeamento dos nós do *Modelo de Concerns* para os nós da AST. Por exemplo, o interesse `rollback` pode ser mapeado para

os nós da AST quando uma operação de `rollback` for acionada. Segundo essa definição, uma preocupação transversal é um interesse que é mapeado para vários elementos do programa (EADDY et al., 2008).

3.2.2 Formalização

Suponha que a implementação do interesse transversal C, representado pelo nó folha no *Modelo de Concerns*, esteja relacionado a ***jps*** *join point shadows* do programa base. Suponha também que ***adv*** *advices* foram usados para implementar C em um sistema AspectJ. Logo é apresentado a primeira relação:

$$\frac{\text{Nº de advices que implementam C}}{\text{Nº de JPS afetados pelos advices}} = \frac{adv}{jps}$$

O melhor caso é quando o interesse é modularizado em um único *advice* que afete os ***jps*** *join point shadows*, e o pior caso é quando são necessários ***jps*** *advices* para referenciar os ***jps*** *join point shadows*. Desta forma a relação descrita está no intervalo $[1/jps, 1]$.

Porém, observa-se que quanto maior o valor pior é o resultado. Assim, para atender à premissa da definição de métricas (EJIOGU, 1991; PRESSMAN, 2005), onde o valor 0 (zero) deve representar o pior caso, foi incluída na fórmula uma função linear para ajustar ao intervalo desejado.

$$1 - \frac{adv}{jps}$$

Essa nova normalização está no intervalo de [0, 1-$(1/jps)$]. Porém o intuito é transpor a razão para o intervalo [0,1], que é um intervalo mais significativo (PRESSMAN, 2005). Para isso, será necessária a normalização do resultado, considerando que o seu valor máximo é igual a 1 - $(1/jps)$. O resultado é a fórmula da métrica *Quantification Degree* (QD), ou Grau de Quantificação:

$$QD(C) = \frac{1 - \frac{adv}{jps}}{1 - \frac{1}{jps}} = \frac{jps - adv}{jps - 1}, \quad jps > 1$$

Nesta fórmula, C é o interesse que será implementado usando ***adv*** *advices* que afetam um total de ***jps*** *join point shadows* do código orientado por objetos. Como pode ser observado, QD(C) varia de 0 (no pior caso) até 1 (no melhor caso), sendo que o pior caso acontece quando ***jps*** = ***adv***, ou seja, cada *advice* implementado afeta somente um

join point shadow no programa base. No melhor caso porém, teremos o melhor benefício da quantificação, ou seja, é necessário um *advice* ($adv = 1$) para afetar todos os ***jps*** *join points shadows* do sistema base.

Na métrica QD, o cálculo (*jps* - *adv*) mede os benefícios da quantificação em termos de redução de espalhamento. Em implementações orientadas por objetos, o código relacionado ao interesse deve ser colocado em cada um dos ***jps*** *join point shadows* do programa *C*. Na implementação orientada por aspectos é possível confinar o código em ***adv*** *advices* (enquanto $jps > 1$ e $adv \geq 1$), visto que o valor de ***jps*** deve ser maior que 1 (um), pois caso seu valor seja igual a 1 (um) não será considerado um interesse transversal. Em outras palavras, usando Java, o interesse estará espalhado ao longo dos ***jps*** *join point shadows* do sistema base, usando AspectJ, o espalhamento será reduzido para ***adv*** *advices*.

A partir da métrica QD foi definida uma sub-métrica, a métrica *Scattering Reduction* (SR). Essa métrica mede a redução do espalhamento de código caso um determinado interesse C venha a ser refatorado para aspectos, utilizando a seguinte fórmula:

$$SR(C) = jps - adv$$

A partir da definição de SR(C) a fórmula de QD(C) pode ser reescrita assim:

$$QD(C) = \frac{SR(C)}{jps - 1}, \quad jps > 1$$

O conceito de interesses homogêneos e heterogêneos utilizados nesta dissertação seguem as seguintes premissas, as chamadas de um interesse são homogêneas quando os parâmetros passados nesses métodos possuem valores idênticos e podem ser modularizados através de um único *advice*, essas chamadas serão consideradas heterogêneas quando os parâmetros passados nas chamadas do interesse tiverem valores diferentes. A partir dessa definição, um interesse transversal que possua n chamadas no código do sistema, pode ter chamadas homogêneas e heterogêneas. Por exemplo, suponha as chamadas a seguir para o método $m : t_1.m(arg_1)$ e $t_2.m(arg_2)$, enquanto t_i indica o destino e arg_i indica o argumento das chamadas ($i = 1$ ou $i = 2$). Essas chamadas do método `m` serão homogêneas quando os valores de arg_1 e arg_2 forem idênticos; caso esses valores sejam diferentes as chamadas serão heterogêneas.

Através de QD é possível definir um aspecto de homogeneidade, ou seja, se QD(C) = 1, o interesse é homogêneo, se QD(C)= 0 o interesse é heterogêneo, os valores entre 0 e 1 serão indicadores para suportar decisões de desenvolvimento. Dessa forma é importante ressaltar que não se pode afirmar que um interesse é considerado homogêneo se tem um valor de QD superior a um valor pré-definido ou heterogêneo caso contrário, esse valor não existe.

3.3 Exemplos

Nesta seção, serão apresentados exemplos da aplicação das métricas propostas, assim como uma análise para cada exemplo.

Exemplo 1: Suponha o *concern* C mapeado para 20 *join point shadows*. Na Figura 11 são apresentados os possíveis valores que QD pode assumir quando o número de *advices* usados para implementar C varia de 1 até 20. Nesse exemplo, é possível verificar que QD possui um valor escalável e seu valor é normalizado entre 0 e 1.

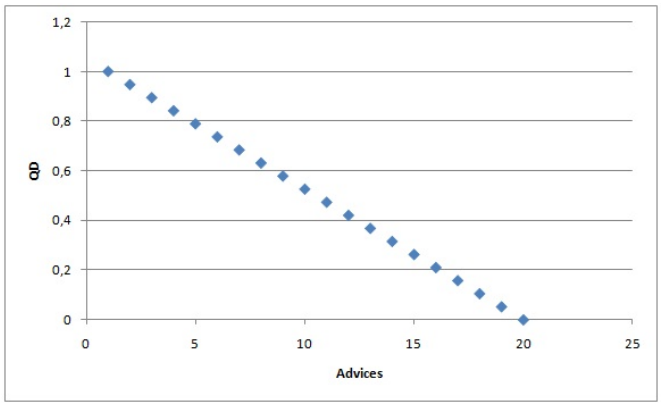


Figura 11: Valores de QD assumidos para *concerns* mapeados para 20 *join point shadows*.

Exemplo 2: Considere o sistema hipotético *Graficos* que possibilita a geração de gráficos a partir de informações definidas pelo usuário. Os seguintes métodos fazem parte da classe `Ponto`, `setX(int x)` e `setY(int y)`, os dois métodos, em conjunto, são responsáveis por desenhar um determinado ponto na tela. Esses métodos são os nós folha do *Modelo de Concerns* do interesse transversal *Ponto*. Na Figura 12 são exemplificadas algumas chamadas dos métodos no sistema.

Os valores calculados para as métricas QD e SR para os interesses `setX` e `setY` são

```
public class Linha{
   ...
   public Linha() {
      Ponto.setX(0);
      ...
      Ponto.setY(0);
   }

   void DesenhaLinha(){
      ...
      int x = 10;
      Ponto.setX(x);
      ...
      Ponto.setY(retornaY());
      ...
   }
   ...
}
```

```
public class Matriz{
   ...
   public Matriz() {
      Ponto.setX(10);
      ...
      Ponto.setY(50);
   }

   void MontaMatriz(){
      ...
      Ponto.setX(x);
      ...
      Ponto.setY(retornaY());

      ....
   }
   ...
}
```

Figura 12: Exemplos de chamadas do interesse *Ponto* no sistema *Graficos*.

mostrados na Tabela 3. Avaliando os resultados identifica-se que os valores de QD e SR foram altos para o interesse `setY` e baixos para o interesse transversal `setX`. Isso se justifica porque foi identificado que é possível implementar as 67 chamadas do interesse transversal `setY` em apenas 3 *advices*, ou seja, uma grande quantidade de chamadas homogêneas desse interesse. Porém, para o método `setX`, serão necessários 65 *advices* para implementar as 67 chamadas, isso porque apresentam um comportamente heterogêneo. Os resultados para a métrica SR acompanham o resultado da métrica QD, ou seja, para o interesse `setY` serão reduzidas 64 chamadas no código orientado por objetos, enquanto para o interesse `setX` somente 2 chamadas serão reduzidas.

Concern	jps	adv	SR	QD
setX(int x)	67	65	2	0,03
setY(int y)	67	3	64	0,97

Tabela 3: Cálculo de QD e SR para os interesses `setX(int x)` e `SetY(int y)`.

A partir dos resultados apresentados, pode-se afirmar que é mais indicada a refatoração do interesse transversal `setY` do que do interesse transversal `setX`. O interesse transversal `setY` apresenta alto grau de quantificação, o que proporcionará a redução das linhas de código, do espalhamento, do entrelaçamento e da replicação de código do interesse no sistema. Para o interesse `setX`, o baixo valor de quantificação não significa

diretamente que este não deva ser modularizado por meio de aspectos, porém o valor baixo do grau de quantificação indica que serão necessários muitos *advices* para atender a todos os *join points* do sistema. Dessa forma, a refatoração pode não resultar em vantagens ao sistema. Por exemplo, ao se analisar o número de linhas de código necessários para a implementação dos interesses, haverá o aumento das linhas necessárias para a implementação dos *pointcuts* e *advices*. Sendo um valor muito superior da quantidade utilizada para implementar as chamadas dos métodos no sistema, isso pode ocasionar dificuldade de manutenção e abertura a erros.

Exemplo 3: Suponha os sistemas hipotéticos *S1* e *S2* e respectivamente as preocupações *C1* e *C2* dos sistemas conforme mostrado na Tabela 4.

Sistema	Concern	jps	adv	SR	QD
S1	C1	101	51	50	0.5
S2	C2	11	6	5	0.5

Tabela 4: Comparação de valores de QD e SR.

Em ambos os casos o valor de QD é o mesmo, 0.5. Este valor significa que o número extraído de *advices* é exatamente a média entre o valor mínimo e máximo de *advices* que podem ser utilizados nesses sistemas. Por exemplo, no sistema S1 o melhor cenário para modularizar o interesse C1 seria 1 *advice* e o pior cenário seria necessitar de 101 *advices*. Conforme o exemplo do sistema S1 o cálculo de QD é apresentado a seguir:

$$QD(C1) = \frac{101 - 51}{100} = 0.5$$

Similar em S2:

$$QD(C1) = \frac{11 - 6}{10} = 0.5$$

Analisando os exemplos dos sistemas de exemplo, observa-se em S1 que a quantificação contribuiu para reduzir de 101 para 51 o número de locais que requerem a presença do código relacionado ao interesse C1, isto é, o valor de SR = 50.

$$SR(C1) = 101 - 51 = 50$$

Já no sistema S2 a quantificação contribuiu para reduzir de 11 para 6 o número de locais que requerem o código relacionado ao interesse C2, ou seja, SR = 5.

$$SR(C1) = 11 - 6 = 5$$

Portanto, mesmo os sistemas S1 e S2 terem os mesmos valores de QD, a modularização dos interesses através de aspectos provê mais benefícios de redução de espalhamento para o sistema S1 (50) do que para o sistema S2 (5). É importante ressaltar que na avaliação dos benefícios da quantificaçao é necessário avaliar o resultado tanto de QD quanto SR, pois fornecem informações distintas. A métrica QD apresenta um índice de melhor uso possível dos aspectos em um determinado cenário, SR porém apresenta o valor absoluto de locais onde não será necessário inserir o código do interesse transversal caso decidam utilizar aspectos.

3.4 Considerações Finais

Neste capítulo inicialmente foi apresentada uma análise quantitativa da aplicação das métricas de separação de interesses CDC e CDO e da métrica CLC em sistemas reais que foram refatorados para aspectos, a partir dessa análise pode-se concluir que os melhores resultados da refatoração para aspectos também apresentaram os melhores resultados quantitativos.

A partir dessa motivação, foram propostas duas novas métricas de quantificação QD e SR, com o objetivo de prover uma medida direta da quantificação de um interesse transversal em um sistema e o valor da redução do espalhamento de código caso esse seja refatorado para aspectos. Após a definição formal dessas métricas, foram apresentados alguns exemplos de aplicação das métricas e as avaliações dos resultados.

Conforme os exemplos apresentados, a partir dos valores de QD e SR, pode-se criar a relação da quantificação com o sucesso da refatoração de um determinado interesse, ou seja, altos valores de QD e SR significam bons resultados da refatoração de interesses transversais para aspectos, assim como quando os valores de QD e SR são baixos não é indicado a refatoração, pois a implementação destes através de aspectos não vai ocasionar redução de número de linhas, de componentes e pode dificultar a manutenção e testes dos sistemas.

4 FERRAMENTA *CONCERNMETRICS*

Neste capítulo, será apresentada a ferramenta *ConcernMetrics* que foi desenvolvida para possibilitar a modelagem do *Modelo de Concerns*, a automatização do cálculo das métricas QD e SR, bem como das métricas convencionais de separação de interesses CDC e CDO. Na Seção 4.1 são apresentados os objetivos da implementação da ferramenta *ConcernMetrics*. Na Seção 4.2 o projeto da ferramenta *ConcernMetrics* é apresentado, detalhando a interface, análise do código fonte e algoritmos de decisão. Na Seção 4.3 são apresentadas as limitações da versão atual da ferramenta. Finalmente, na Seção 4.4 são feitas as considerações finais.

4.1 Objetivos

Para a aplicação manual das métricas QD e SR, é necessário definir um *Modelo de Concerns* com o detalhamento da árvore de *concerns* até os níveis folha dos métodos que implementem o interesse transversal selecionado. A partir do *Modelo de Concerns*, é necessário identificar em todo o sistema os *join points* e definir quais chamadas são homogêneas e heterogêneas para consequentemente ser modularizadas através do mínimo de *advices*. Somente após essa análise do sistema como um todo, as métricas QD e SR serão efetivamente calculadas. Essas métricas foram formalizadas e exemplificadas, porém o processo para a sua aplicação pode se tornar exaustivo, complexo e aberto a erros quando aplicado manualmente, crescendo conforme o tamanho do sistema e a quantidade de *join points* espalhados pelo código fonte aumentam.

Outro fator importante é a avaliação quantitativa da comparação dos resultados das métricas de separação de interesses CDC e CDO aplicadas a uma versão do sistema orientado por objetos e também em sua versão já refatorada para aspectos, conforme foi apresentado no estudo do Capítulo 3. Como foi verificado, o valor quantitativo da comparação dessas métricas pode ser uma medida importante na avaliação da refatoração de um sistema para aspectos. Essa avaliação pode se tornar muito dispendiosa visto

que, para o cálculo das métricas CDC e CDO, precisa-se que sejam identificados os interesses transversais a serem analisados, os métodos, classes, *advices* e aspectos que os implementem e que façam chamadas a esses interesses. Outro fator importante é o fato de que é necessário que exista uma versão do sistema já refatorada para aspectos, o que pode tomar tempo e esforço em vão, caso se identifique que os interesses transversais não poderão ser devidamente modularizados através de aspectos.

O objetivo do desenvolvimento da ferramenta *ConcernMetrics* é automatizar o processo de modelagem do *Modelo de Concerns*, o cálculo da métrica de quantificação QD e da sub-métrica de redução do espalhamento SR. A ferramenta *ConcernMetrics* propõe ainda o cálculo das métricas convencionais de separação de interesses CDC e CDO para o código orientado por objetos e também a estimativa dos valores destas métricas caso o interesse venha a ser refatorado para aspectos.

Para o cálculo das métricas propostas para a ferramenta *ConcernMetrics* as seguintes funcionalidades foram implementadas:

- Mapeamento lógico de interesses transversais possibilitando a modelagem do *Modelo de Concerns*;
- Identificação dos *join points* no código fonte;
- Identificação de chamadas homogêneas e heterogêneas do interesse e decisão do agrupamento dos *join points* em *advices*;
- Identificação de métodos e classes que implemenem e acessem um determinado interesse.

A ferramenta *ConcernMetrics* automatiza esses processos e calcula as métricas totalmente baseada no código fonte orientado por objetos, isto sem que o sistema sofra qualquer alteração estrutural ou comportamental. A ferramenta *ConcernMetrics* foi desenvolvida na linguagem Java e disponibilizada como um *plugin* da ferramenta de desenvolvimento Eclipse.

4.2 Implementação

Nesta seção, será detalhado todo o processo de desenvolvimento da ferramenta *ConcernMetrics*, suas funcionalidades, tecnologias utilizadas e algoritmos de decisão.

4.2.1 Interface

A ferramenta *ConcernMetrics* estende a interface da ferramenta *ConcernMapper*, proposta por Robillard e Weigand-Warr (2005). *ConcernMapper* é um *plugin* para o Eclipse que permite acesso à estrutura do sistema criando uma comunicação direta entre o código fonte e a ferramenta.

A ferramenta *ConcernMetrics* adicionou algumas funcionalidades específicas à ferramenta *ConcernMapper*, porém manteve suas principais funcionalidades. Por exemplo, *ConcernMapper* possui a funcinalidade de criação de um modelo lógico de interesses que é apresentado em forma de uma árvore hierárquica. A ferramenta *ConcernMapper* possui ainda a possibilidade de identificação no código fonte a chamada de métodos e atributos que foram vinculados ao *concern* inserido no seu modelo lógico, um exemplo é demonstrado na Figura 13.

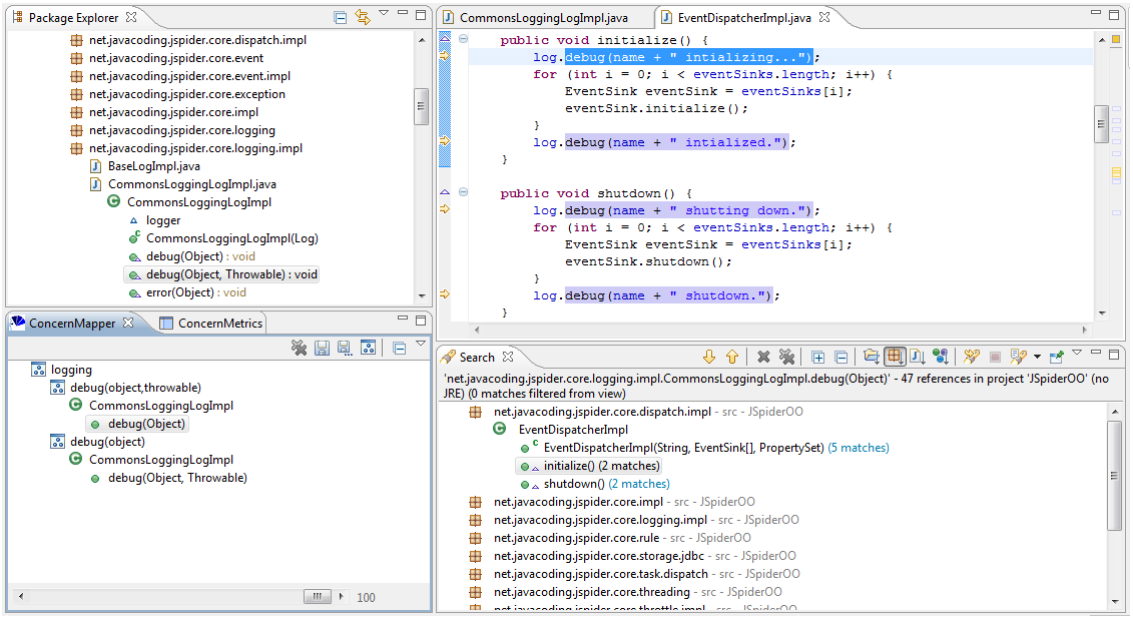


Figura 13: Identificação dos métodos dos interesses a partir da visão da ferramenta *ConcernMapper*.

A ferramenta *ConcernMetrics* estendeu a funcionalidade de mapeamento de interesses transversais de *ConcernMappers* que permite a modelagem do *Modelo de Concerns*, conforme definido no Capítulo 3. De acordo com a definição do modelo proposto, é necessário a adição de *concerns* ao modelo até o seu nível atômico, ou seja, até que não seja possível dividir o interesse em mais de um método que o implementa. Um exemplo da modelagem do *Modelo de Concerns* é mostrado na Figura 14.

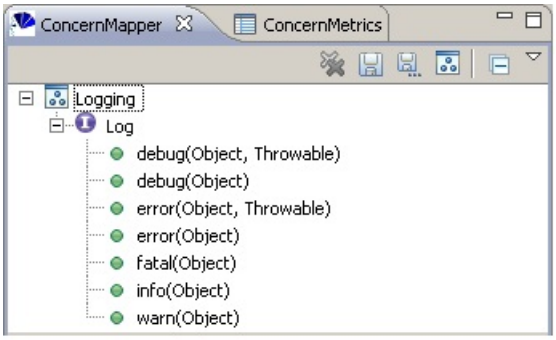


Figura 14: Ferramenta *ConcernMetrics*: *Modelo de Concerns* do interesse transversal *logging* do sistema JSpider.

O *Modelo de Concerns* é criado na interface de *ConcernMapper*. Para montar o modelo, é necessário selecionar os métodos associados ao interesse transversal a partir da visão *Package Explorer* do Eclipse, arrastando-os e soltando-os no nó correspondente ao interesse no *Modelo de Concerns*. Na Figura 15, é apresentado um diagrama de sequência da utilização da ferramenta *ConcernMetrics*.

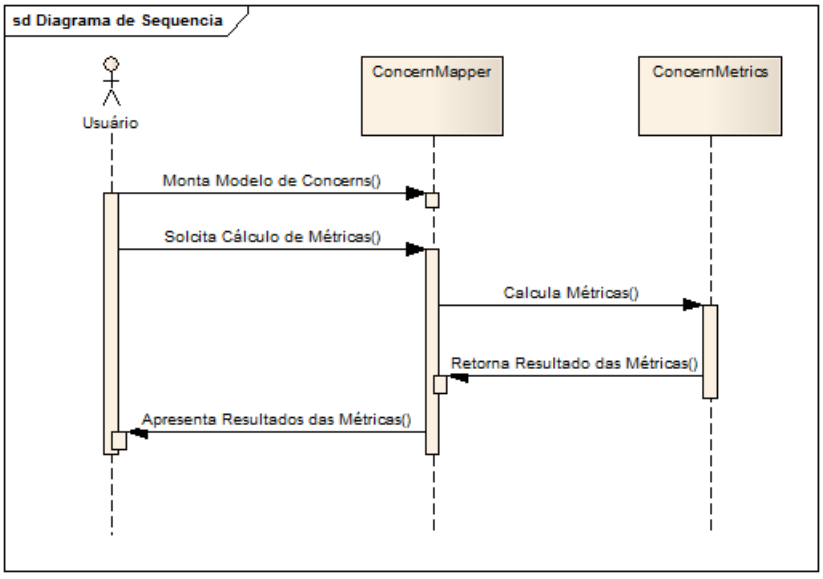
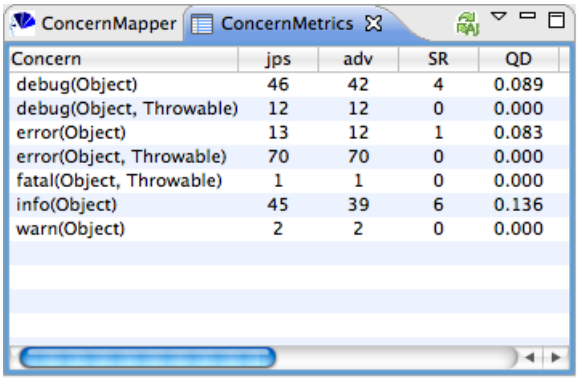


Figura 15: Diagrama de sequência da ferramenta *ConcernMetrics*.

Após a definição do *Modelo de Concerns*, o cálculo das métricas pode ser solicitado na tela *ConcernMetrics*. A tela *ConcernMetrics* do *plugin* foi desenvolvida para ser a interface do usuário para solicitação do cálculo das métricas para os interesses detalhados no *Modelo de Concerns* e para apresentação dos resultados das métricas QD, SR, CDC e CDO para o código orientado por objetos e das métricas CDC e CDO para o código

orientado por aspectos. Os resultados são apresentado em uma inteface em forma de tabela, possibilitando uma visualização e análise dos resultados, conforme mostrado na Figura 16.



Concern	jps	adv	SR	QD
debug(Object)	46	42	4	0.089
debug(Object, Throwable)	12	12	0	0.000
error(Object)	13	12	1	0.083
error(Object, Throwable)	70	70	0	0.000
fatal(Object, Throwable)	1	1	0	0.000
info(Object)	45	39	6	0.136
warn(Object)	2	2	0	0.000

Figura 16: Ferramenta *ConcernMetrics*: apresentação do resultado do cálculo das métricas para o interesse transversal *logging* do sistema JSpider.

4.2.2 Análise do Código Fonte

A leitura do código fonte do sistema que está sendo avaliado é feita utilizando-se o *framework* Java ASM[1] para manipulação e análise de *bytecode*. O *framework* ASM foi desenvolvido por Bruneton, Lenglet e Coupaye (2002) e possibilita a análise estrutural do sistema, modificações de código, declaração de classes, métodos e atributos, entre outras manipulações mais complexas.

A ferramenta *ConcernMetrics* necessita analisar a estrutura do código fonte para o cálculo das métricas, para isso implementa as interfaces `Visitor` do ASM. As intefaces `Visitor` são responsáveis pela análise estrutural das classes (`ClassVisitor`), métodos (`MethodVisitor`) e atributos (`FieldVisitor`) do código fonte. Dessa forma, o código fonte do sistema é atravessado identificando declaração de classes, métodos, atributos, instanciação de objetos, identificação de chamadas de métodos e parâmetros, assim como a identificação dos tipos e valores dos parâmetros, atributos e objetos. Um exemplo da interface `ClassVisitor` do *framework* ASM é mostrado na Figura 17. O método `visit` (linha 02) é chamado a cada declaração de uma classe, podendo identificar através dessa chamada o seu nome, assinatura, classe mãe e interfaces que essa classe implementa. O método `visitMethod` (linha 12), por exemplo, é chamado nas declarações e chamadas de

[1] As siglas ASM não tem nenhum significado específico, é apenas uma referência para a palavra chave –asm– na linguagem C, que permite que algumas funções sejam implementadas em linguagem *assembly* (BRUNETON; LENGLET; COUPAYE, 2002).

métodos nesta classe. A partir dessas interfaces é possível fazer uma leitura completa do código fonte do sistema.

```
01: public interface ClassVisitor {
02:    void visit(int version, int access, String name, String signature,
03:       String superName, String[] interfaces);
04:    void visitSource(String source, String debug);
05:    void visitOuterClass(String owner, String name, String desc);
06:    AnnotationVisitor visitAnnotation(String desc, boolean visible);
07:    void visitAttribute(Attribute attr);
08:    void visitInnerClass(String name, String outerName, String innerName,
09:       int access);
10:    FieldVisitor visitField(int access, String name, String desc,
11:       String signature, Object value);
12:    MethodVisitor visitMethod(int access, String name, String desc,
13:       String signature, String[] exceptions);
14:    void visitEnd();
15: }
```

Figura 17: Interface `ClassVisitor` do *framework* ASM (BRUNETON; LENGLET; COUPAYE, 2002).

As informações coletadas através da classe `Visitor` da ferramenta *ConcernMetrics*, que implementa as interfaces `ClassVisitor, MethodVisitor` e `FieldVisitor`, são tratadas para que sejam armazenadas em uma estrutura em árvore dentro do módulo *Tree*. Esse módulo trata as informações e as armazena conforme os seguintes objetos:

- Objeto *Tree*: contém uma estrutura em árvore de todas as classes do sistema.
- Objeto *Class*: contém as informações de herança da classe e seus métodos.
- Objeto *Method*: contém o valor dos parâmetros e a informação se são dinâmicos ou estáticos, tipo dos parâmetros e chamadas de outros métodos.

Para o armazenamento das informações, os objetos se relacionam de acordo com a estrutura modelada no diagrama de classes da ferramenta *ConcernMetrics*, Figura 18.

O objeto *Tree* pode ser considerado o nó principal da estrutura em árvore que contém as informações do código fonte do sistema, esse contém uma lista de objetos *Class*, que é a lista de classes do sistema. Cada objeto *Class* por sua vez possui uma lista de objetos *Method* que representam os métodos implementados nessa classe. Um objeto *Method* contém uma lista de objetos do tipo *Param*, que contém os parâmetros desse método, seu tipo e valor; uma lista dos objetos declarados no método, seu tipo e valor; e uma lista de

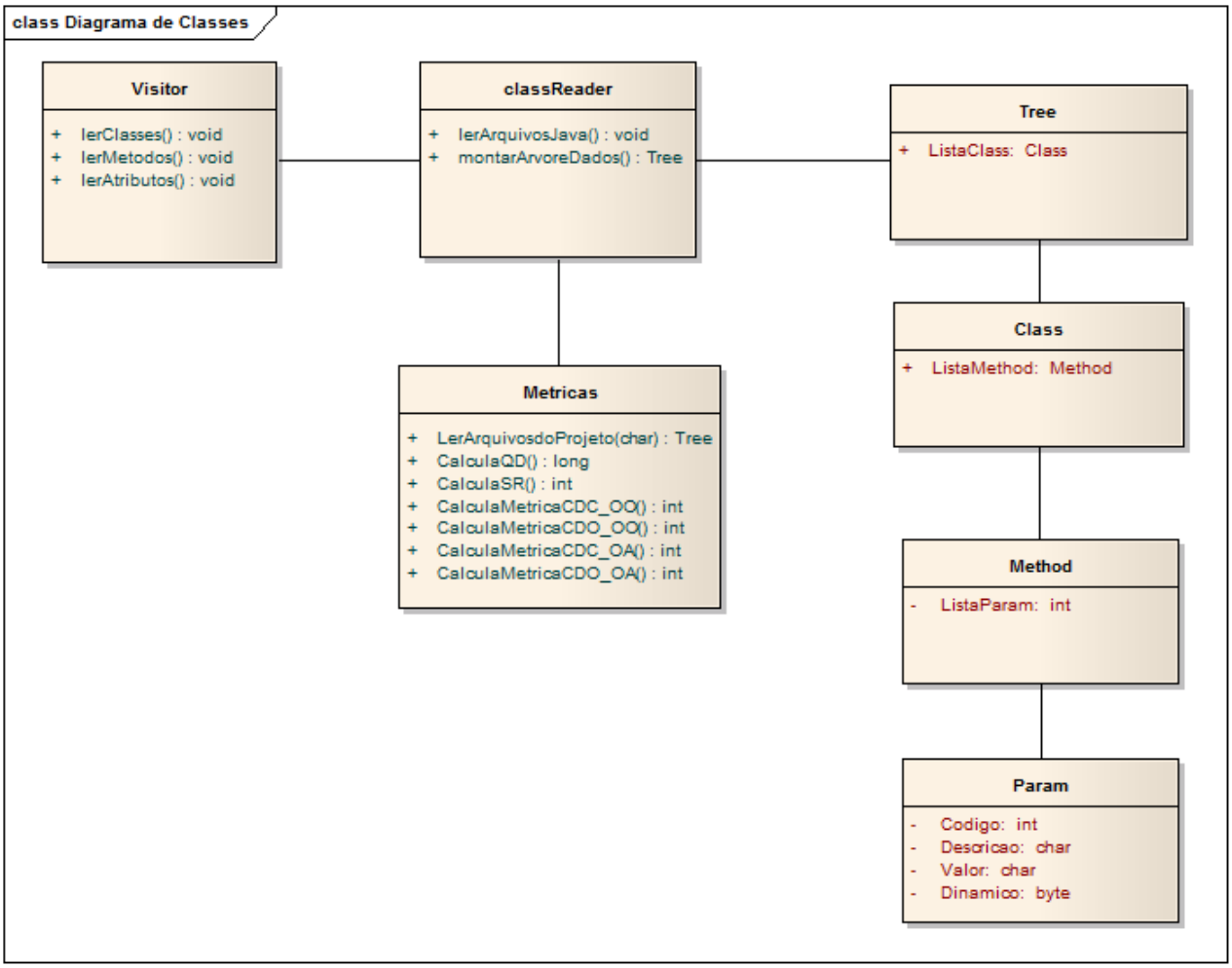


Figura 18: Diagrama de classes da ferramenta *ConcernMetrics*.

outros métodos que este método acessa. A partir desta estrutura, a ferramenta *ConcernMetrics* possui as informações necessárias para a localização dos *join points*, definição dos *advices* e análise de quais classes e métodos implementam determinados interesses contidos no *Modelo de Concerns*.

4.2.3 Algoritmos de Decisão

O módulo *Metrics* da ferramenta é o responsável pela identificação e análise das informações dos interesses na estrutura *Tree* e pelo cálculo das métricas QD, SR, CDC e CDO. Para efetuar a análise e cálculo das métricas foram definidos alguns algoritmos de decisão, estes possibilitam a identificação de *advices* homogêneos e heterogêneos, identificam os *join points* e calculam as métricas propostas para cada interesse do *Modelo de Concerns*.

A decisão mais importante no momento do cálculo das métricas é a identificação dos interesses homogêneos e heterogêneos, ou seja, identificar quais chamadas são idênticas e quais não são para a definição da assinatura e da quantidade de *advices* necessários para atender a todos os *join points*. Essas definições são feitas no módulo *Metrics* e obedecem aos algoritmos de decisão detalhados a seguir.

- **Algoritmo 1**: Suponha as chamadas a seguir para o método m : $t_1.m(arg_1)$ e $t_2.m(arg_2)$, t_i indica o destino e arg_i indica o argumento das chamadas ($i = 1$ ou $i = 2$). Essas chamadas serão movidas para o mesmo *advice* nas seguintes condições:

  1. Caso sejam métodos estáticos, t_1 e t_2 devem denotar a mesma classe ou classes que herdem da mesma superclasse. Em caso de métodos dinâmicos, t_1 e t_2 devem denotar que são do mesmo tipo ou que são derivados de um tipo comum.
  2. arg_1 e arg_2 são o mesmo valor constante ou campos tenham o mesmo tipo (ou são derivados de um tipo comum).

  Suponha as chamadas de `start` e `log` nas classes A e B da Figura 19. Neste exemplo, o interesse `start` pode ser movido para um único *advice* considerando-se as definições do Algoritmo 1. Isso se verifica pois, o objeto `tx` em ambas as classes é do tipo `Transaction`, logo atende à regra número 1, e o parâmetro passado na chamada do método nas duas classes é estático e tem o valor igual 1, logo atende à regra número 2.

```
class A {
  Transaction tx;
  void foo(){
    tx.start(1);
    ....
    Logger.log("finished");
    ....
  }
}
```

```
class B {
  Transaction tx;
  void bar(){
    tx.start(1);
    ....
    Logger.log("panic");
    ....
  }
}
```

Figura 19: Exemplo de chamadas do interesse transversal *log*.

O método `log`, do mesmo exemplo da Figura 19, é um método estático da classe `Logger`, dessa forma atende à regra número 1 do algoritmo. Porém, os argumentos passados nas chamadas desse método são dinâmicos e não têm o mesmo valor. O valor do parâmetro passado para `log` na classe A é "finished", enquanto o valor de `log`

na classe B é "panic", desta forma não atendem à regra número 2, consequentemente não é possível confiná-los em um único *advice*.

- **Algoritmo 2**: Suponha as chamadas a seguir para os métodos $m_i : x.m_1(arg)$ e $y.m_2(arg)$, enquanto m_i indica os *advices* contidos no *Modelo de Concerns*, sendo ($i = 1$ ou $i = 2$). Essas chamadas serão movidas para o mesmo *advice* nas seguintes condições:

  1. m_1 e m_2 façam parte do mesmo *Modelo de Concerns*;
  2. m_1 e m_2 sejam chamadas em sequência, ou sejam, sem nenhum outro código da implementação do sistema entre as duas chamadas.

  Considere o *Modelo de Concerns* do interesse *logging*, Figura 10 na página 51 do Capítulo 3. Esse *Modelo de Concerns* tem o nó principal `logging` e os seguintes nós folha: `debug, error, fatal, info` e `warn`. Considere o trecho do código fonte da classe `VelocityPlugin` do sistema JSpider apresentado na Figura 20.

```
class VelocityPlugin
{
   ...
   log.debug("opened trace file '" + traceFileName + "'");
   log.info("Writing to trace file: " + traceFileName)
   ....
}
```

Figura 20: Exemplo de chamada de interesses transversais do sistema JSpider.

No exemplo da Figura 20 há a chamada de dois interesses do *Modelo de Concerns* de *logging*, `debug` e `info`. Ao analisar esses dois interesses, segundo o Algoritmo 2, é considerado que ambos possam ser modularizados por um único *advice*, visto que ambos fazem parte do mesmo *Modelo de Concerns* e ambos estão sendo chamados em sequência.

Considere as classes hipotéticas X e Y da Figura 21. Neste exemplo, na classe X, as chamadas dos interesses `error` e `debug` podem ser modularizados em um mesmo *advice* pois atendem ao Algoritmo 2. Isso porque fazem parte do mesmo *Modelo de Concers* e são chamados em sequência, sem nenhum outro código entrelaçado entre as chamadas. O mesmo ocorre para o exemplo da classe Y e as chamadas dos interesses `error` e `warn`.

Além disso, há ainda outro fator para análise neste mesmo exemplo, os interesses atendem ao Algoritmo 1 que identifica as chamadas de um mesmo *advice*, ou seja, as chamadas dos interesses são chamadas estáticas do mesmo tipo, tanto para a chamada do interesse `error` e `warn` da Classe X, quanto para as chamadas dos mesmos interesse na Classe Y, e os parâmetros são estáticos e idênticos nas chamadas dos mesmos métodos. Dessa forma, será possível modularizar as quatro chamadas dos interesses em um único *advice* pois atendem às regras dos dois algoritmos definidos.

```
class X                              class Y
{                                    {
   void a()                             void b()
   {                                    {
      ...                                  ...
      log.error("Error.");                 log.error("Error.");
      log.warn("Caution");                 log.warn("Caution");
      ...                                  ...
   }                                    }
}                                    }
```

Figura 21: Exemplo de chamada de interesses transversais chamados em sequência.

Os algoritmos detalhados acima foram feitos para a identificação de interesses homogêneos e heterogêneos e a partir desses a quantidade de *advices* necessários para implementá-los. A partir da identificação dos *advices* necessários para implementar os interesses transversais, é possível efetuar o cálculo das métricas QD e SR. Para o cálculo das métricas CDC e CDO do código orientado por objetos não é necessário nenhum dos algoritmos, visto que esses valores são obtidos diretamente do código fonte e essa identificação é feita pela ferramenta baseada somente nas informações armazenadas no módulo *Tree*.

A ferramenta *ConcernMetrics* estima o valor das métricas CDC e CDO caso o sistema seja refatorado para aspectos, esses cálculos são totalmente baseados no código orientado por objetos e na quantidade de *advices* e aspectos que serão necessários para modularizar os interesses transversais que estão sendo analisados. A partir dessas informações as seguintes fórmulas foram definidas para a estimativa dos cálculos de CDC e CDO:

- **CDC**: O cálculo de CDC, conforme já detalhado anteriormente, é feito somando-se

a quantidade de classes e aspectos que implementam ou acessam um determinado interesse. Para o cálculo de CDC para o código orientado por aspectos, é feita a soma de C (quantidade de classes) que implementam o interesse e A (quantidade de aspectos) que implementam os *advices*. Consideram-se para esse cálculo somente a classe que implementa o interesse transversal, isto porque é previsto que todos os *join points* do sistema serão modularizados através de aspectos. A partir disso, a fórmula para o cálculo de CDC para o sistema orientado por aspectos é definida:

$$CDC(OA) = C + A$$

Um exemplo do cálculo para CDC é o método `debug(object)` do interesse transversal *logging* do sistema JSpider apresentado na Figura 22.

```
class DistributedLoadThrottleProvider
{
   ...
   log.debug("throttle interval set to " + interval + " ms.");
   ....
}
```

Figura 22: Exemplo de chamada de interesse transversal do sistema JSpider.

Esse interesse transversal no código orientado por objetos do sistema possui o valor de CDC igual a 16, para o sistema orientado por aspectos, conforme a fórmula apresentada, o valor de CDC será igual a 2.

O valor de C é a quantidade de classes que implementam o interesse transversal, esse valor é obtido a partir da estrutura *Tree*, o valor de A é a quantidade de aspectos necessários para implementar os *advices*, este será sempre considerado igual a 1, pois todos os *advices* do sistema podem ser implementados em um único aspecto, caso seja separado em mais aspectos será feito por opção dos desenvolvedores do sistema.

- **CDO**: O cálculo de CDO é feito com base na quantidade de métodos e *advices* que implementam ou acessam determinado interesse. Para o cálculo de CDO para o código orientado por aspectos considera-se o total de **adv** *advices* somado com o valor m, que é a quantidade de métodos que implementam o interesse. A fórmula do cálculo de CDO é definida a sequir:

$$CDO(OA) = adv + m$$

O exemplo apresentado na Figura 22 pode ser utilizado também para exemplificar o cálculo de CDO. Para o sistema orientado por objetos o valor de CDO é igual a 47, para o sistema orientado por aspectos, para o método `debug(object)`, o resultado será igual a 43.

Para estimar o valor de CDO para o sistema orientado por aspecto deve-se conhecer o valor de *advices* necessários para modularizar todos os *join points* do interesse que está sendo avaliado, neste exemplo foram necessários 42 *advices* para atender aos 46 *join points*. O valor de *advices* foi somado ao valor de m, baseado na definição do *Modelo de Concerns* utilizado, onde um interesse transversal deve ser detalhado até serem encontrados seus nós folha que são unidades indivisíveis, ou seja, um único método, o valor de m será sempre igual a 1 (um).

A partir das definições feitas nesta seção a ferramenta *ConcernMetrics* calcula o valor da métrica de quantificação QD e a sub-métrica de redução do espalhamento SR. Calcula ainda, com base informações coletadas e de acordo com a definição das fórmulas apresentadas nesse capítulo, os valores das métricas de separação de interesses CDC e CDO para o código orientado por objetos e também estima os valores para essas mesmas métricas considerando uma possível refatoração do sistema para aspectos. A ferramenta *ConcernMetrics* possibilita a geração do *Modelo de Concerns* e a partir deste facilmente solicitar o cálculo das métricas. Os resultados são apresentados em uma interface tabulada com os resultados das métricas para cada interesse, tornando fácil a análise dos resultados para tomadas de decisões de refatoração do sistema.

4.3 Limitações da Ferramenta *ConcernMetrics*

A ferramenta *ConcernMetrics* apresenta ainda algumas limitações na sua versão atual, conforme detalhado a seguir:

- A fim de avaliar os benefícios da quantificação, a ferramenta *ConcernMetrics* apenas trata de interesses transversais dinâmicos, ou seja, interesses transversais que podem ser modulados por meio de *advices* (KICZALES et al., 2001). Além disso, a ferramenta assume que interesses transversais (dinâmicos) sempre correspondem à chamadas de métodos únicos. Portanto, as preocupações associadas a outras

declarações (atribuições, loops, etc) ou interesses associados a várias instruções, devem ser primeiro extraídos do método, utilizando a *Extract Method Refactoring* (FOWLER et al., 1999). No entanto, a exigência da aplicação prévia de extração para um método, nesses casos, não representa um esforço infrutífero. Mesmo que os desenvolvedores decidam não utilizar aspectos, os métodos de extração, na maioria dos casos, contribuem para eliminar código repetido e para aumentar a inteligibilidade.

- AspectJ permite utilização de aspectos somente em pontos bem definidos na execução de sistemas, ou seja, antes, durante ou depois de alguma chamada que possa ser identificada por um *pointcut*. Em casos que não é possível a identificação desses *join points*, é necessária uma transformação de código. Geralmente, essas transformações de código podem ser divididas em Reordenação da Declaração ou Método de Extração (MALTA; VALENTE, 2009; BINKLEY et al., 2006). A ferramenta *ConcernMetrics* não considera eventuais necessidades de transformações no código, considera que os *join points* encontrados são possíveis de ser identificados através de *pointcuts*. A ferramenta avalia somente se é possível extrair as chamadas associadas a interesses transversais para o mesmo *advice*. No entanto, é aconselhável, para o cálculo das métricas e identificação de *join points* e *advices* através de *ConcernMetrics*, a avaliação se há a necessidade de eventuais transformações de código previamente à sua utilização.

4.4 Considerações Finais

Neste capítulo foi apresentada a ferramenta *ConcernsMetrics*, que é uma ferramenta para automatização do processo de modelagem do *Modelo de Concerns* e do cálculo das métricas propostas nesta pesquisa, a métrica de quantificação QD e a sub-métrica de redução do espalhamento SR. A ferramenta *ConcernMetrics* ainda calcula o valor das métricas de separação de interesses CDC e CDO para o código orientado por objetos e estima os valores dessas métricas caso o sistema seja refatorado para aspectos.

A ferramenta é capaz de identificar de informações no código fonte como *join points*, métodos e classes que implementem e acessem o interesse em análise. A ferramenta possui ainda a funcionalidade de estimativa do menor número necessário de *advices* para modularizar os interesses transversais que estão sendo analisados. Foram apresentados detalhes da interface, da estrutura para leitura do código fonte, algoritmos de decisão, e por fim as limitações presentes na versão atual da ferramenta.

5 AVALIAÇÕES DO GRAU DE QUANTIFICAÇÃO

Neste capítulo é apresentada uma avaliação das métricas de quantificação QD e SR através da aplicação destas em três sistemas, JAccounting e JSpider refatorados por Binkley et al. (2006) e JHotDraw refatorado por Binkley et al. (2005). As métricas QD e SR foram aplicadas manualmente nos sistemas selecionados como estudos de caso e esses valores serão comparados com os resultados obtidos pela avaliação quantitativa apresentada no Capítulo 3. Ainda neste capítulo serão apresentados os resultados obtidos para as métricas QD, SR, CDC e CDO para o código orientado por objetos e também os resultado das métricas CDC e CDO para o código orientado por aspectos, feitos para os sistemas JSpider, JAccounting e JHotDraw através da ferramenta *ConcernMetrics*. Os valores obtidos através da ferramenta serão comparados com os resultados das análises já efetuadas manualmente e serão discutidas divergências encontradas.

A organização do capítulo é feita como a seguir. São apresentadas as aplicações em estudos de caso das métricas QD e SR na Seção 5.1, na Seção 5.2 serão detalhados os testes feitos com a ferramenta *ConcernMetrics* em três estudos de caso e avaliação dos resultados, na Seção 5.3 é apresentada uma discussão sobre a quantificação e na Seção 5.4 finalmente serão apresentadas as considerações finais.

5.1 Avaliação Manual das Métricas QD e SR

Os sistemas JSpider e JAccounting já foram apresentados no Capítulo 3, onde foram utilizados para uma avaliação quantitativa através das métricas CDC e CDO aplicadas no código orientado por objetos e também no código orientado por aspectos. O sistema JHotDraw por sua vez é um *framework* orientado por objetos da 2D *graphics*[1], foi originalmente desenvolvido por Erich Gamma e Thomas Eggenschwiler e é um *framework* GUI para Java. O sistema JHotDraw possui ainda uma versão que tem alguns interesses transversais implementados através de aspectos que é o AJHotDraw[2], desenvolvido por

[1] http://www.jhotdraw.org, version v.54b1
[2] http://ajhotdraw.sourceforge.net/

Binkley et al. (2005) e juntamente com JHotDraw é utilizado em diversos estudos sobre desenvolvimento de software orientado por aspectos (ANBALAGAN; XIE, 2007; MARIN; DEURSEN; MOONEN, 2007; MARIN et al., 2009). Esses sistemas foram selecionados como estudos de caso por terem versões originais orientadas por objetos e também versões refatoradas para aspectos, dessa forma é possível avaliar os resultados obtidos pelas métricas QD e SR e compará-los aos resultados da refatoração desses sistemas.

5.1.1 Exemplo 1 - JAccounting

O sistema JAccounting contém o interesse transversal *transaction* que pode ser decomposto nos interesses atômicos: iniciar a transação (`beginTransaction`), salvar a transação (`commit`) e desfazer a transação (`rollback`).

Foi feita uma análise manual no sistema JAccounting para a localização dos *join points* do interesse *transaction* do código fonte orientado por objetos. Os *advices* que modularizam os *join points* foram identificados com base no código fonte já refatorado para aspectos. Na Tabela 5 são apresentados os valores obtidos das métricas QD e SR, a quantidade de *join point shadows* (jps) encontrados no sistema e o número de *advices* (adv) necessários para atender a todos os *jps* do sistema JAccounting.

Interesses	**jps**	**adv**	**SR**	**QD**
beginTransaction	15	1	14	1
commit	15	1	14	1
rollback	14	1	13	1

Tabela 5: Valores de QD e SR para o sistema JAccounting.

Análise dos resultados: No estudo de caso JAccouting foi obtido o melhor resultado possível da quantificação através da utilização de aspectos, ou seja, os três interesses avaliados obtiveram o máximo possível de quantificação (QD = 1). Além disso, foram removidas 14 chamadas do código base do sistema relacionado ao interesse `beginTransacion` (SR = 14), 14 chamadas do interesse `commit` (SR = 14) e 13 chamadas do interesse `rollback` (SR = 13). Dessa forma, pode-se afirmar que o resultado da quantificação e redução do espalhamento será satisfatório caso esse sistema seja refatorado para aspectos. Conforme a análise apresentada no Capítulo 3, com base no código fonte orientado por objetos e também no código fonte orientado por aspectos, esse resultado está correto, pois foi constatado que as 44 chamadas do interesse *transaction* foram efetivamente modularizadas por 3 *advices*. Além disso o valor das métricas CDC e CDO,

calculados diretamente no código orientado por objetos e no código orientado por aspectos, mostrou que a refatoração dos interesses através de aspectos reduziu em 36% o valor de CDO e 70% o valor de CDC, resultado tão positivo quanto o verificado através das métricas QD e SR, validando desta forma que o resultado da quantificação encontrada por QD e SR é compatível com o resultado original, conforme analisado no código já refatorado.

5.1.2 Exemplo 2 - JSpider

No sistema JSpider, o interesse transversal utilizado para o estudo de caso foi o interesse *logging*. Este se decompõe nos interesses atômicos: `debug`, `info`, `warn`, `error` e `fatal`. Foi feita uma inspeção manual no sistema JSpider para identificação dos interesses e localização dos *join points* no código fonte orientado por objetos. A identificação dos *advices* que foram utilizados para modularizar os interesses foi feita diretamente no código fonte orientado por aspectos. A avaliação manual do código fonte do sistema JSpider foi bem dispendiosa, visto que existem 190 *join points* do interesse e essas chamadas apresentam características heterogêneas, o que dificulta a identificação dos *concerns*. Por fim, as métricas QD e SR foram calculas e os resultados são apresentado na Tabela 6.

Interesses	**jps**	**adv**	**SR**	**QD**
debug(Object)	45	44	1	0.02
debug(Object,Throwable)	12	12	0	0.00
error(Object)	12	11	1	0.09
error(Object,Throwable)	68	68	0	0.00
fatal(Object,Throwable)	1	1	0	0.00
info(Object)	44	38	6	0.14
warn(Object)	2	2	0	0.00

Tabela 6: Valores de QD e SR para o sistema JSpider.

Análise dos resultados: Através dos valores calculados para a métrica QD, pode-se pressupor que caso o sistema seja refatorado para aspectos irá utilizar poucas declarações quantificadas, uma vez que os valores são zero ou muito próximos a zero. Em outras palavras, esses valores expressam o modo como os *advices* são utilizados na versão orientada por aspectos de JSpider. Na maioria dos casos, as chamadas dos interesses foram simplesmente transferidos para um *advice*, ou seja, foram retiradas as chamadas dos *join points* e foram implementados os *advices* para atendê-las. Foi identificado que o sucesso da refatoração para aspectos é proporcional ao grau de quantificação de um

interesse, ou seja, quanto mais alto o grau de quantificação melhores resultados serão obtidos através da refatoração. O valor de QD também implica nos valores de SR, pois quando o valor da quantificação é baixo, consequentemente o valor de SR também será baixo e menos indicada será essa refatoração.

Na análise quantitativa do Capítulo 3, o sistema JSpider não mostrou bons resultados da quantificação, onde os valores para CDO aumentaram 125%, ou seja, foi necessário um número muito maior de *advices* do que o número de *join points* espalhados no sistema para refatorar esses interesses, sendo esse valor compatível com os resultados de QD e SR. A partir dessa análise pode-se afirmar que o resultado da refatoração do interesse transversal *logging*, com base na quantificação e na redução do espalhamento, não se mostra vantajosa.

5.1.3 Exemplo 3 - JHotDraw

No sistema JHotDraw os valores de QD e SR foram calculados para dois métodos relacionados a diferentes interesses transversais: `fireSelectionChanged`: faz parte do interesse *selection picture*; `setUndoActivity`: faz parte do interesse *undo*. Foi feita uma inspeção manual no código fonte orientado por objetos e no código orientado por aspectos. A Tabela 7 apresenta os valores calculados para as métricas QD e SR para o sistema JHotDraw.

Interesses	**jps**	**adv**	**SR**	**QD**
fireSelectionChanged()	4	2	2	0.67
setUndoActivity(Undoable)	10	9	1	0.11

Tabela 7: Valores de QD e SR para o sistema JHotDraw.

Análise dos resultados: Foi encontrado um valor médio para o cálculo de QD para o interesse `fireSelectionChanged` (QD = 0,67) e o resultado de SR foi igual a 2. Isso significa que o grau de quantificação do interesse é intermediário, porém ao se analisar juntamente QD e SR observa-se que o valor de SR é muito pequeno, podendo neste caso não ser indicada sua refatoração pois a redução do espalhamento será pequena e pode causar maior complexidade no sistema pela sua implementação através de aspectos. Para o interesse `fireSelectionChanged` o grau de quantificação foi baixo (QD = 0.11) assim como o valor de SR (SR = 1), caracterizando um interesse pouco quantificável e não sendo indicada sua refatoração.

5.2 Avaliação da Ferramenta *ConcernMetrics*

Com o intuito de validar os cálculos feitos pela ferramenta *ConcernMetrics* das métricas de quantificação QD e SR e também das métricas de separação de interesses CDC e CDO para o código orientado por objetos e também para o código orientado por aspectos, foram utilizados os sistemas JAccounting, JSpider e JHotDraw como estudos de caso.

Em geral, os resultados sugeridos pela ferramenta *ConcernMetrics* se igualam aos valores calculados manualmente a partir das versões orientadas por objetos e orientadas por aspectos destes sistemas. Em alguns casos, porém, os resultado não foram correspondentes. Isso se deve a novos requisitos implementados ou a falta de chamadas desses requisitos nas versões orientadas por aspectos. Os detalhes são mais bem discutidos nos exemplos a seguir.

5.2.1 Exemplo 1 - JAccounting

O sistema JAccounting foi utilizado como estudo de caso da aplicação das métricas de quantificação QD e SR, assim como das métricas de separação de interesses CDC e CDO através da ferramenta *ConcernMetrics*. Na Figura 23, é apresentado o *Modelo de Concerns* modelado na ferramenta *ConcernMetrics*.

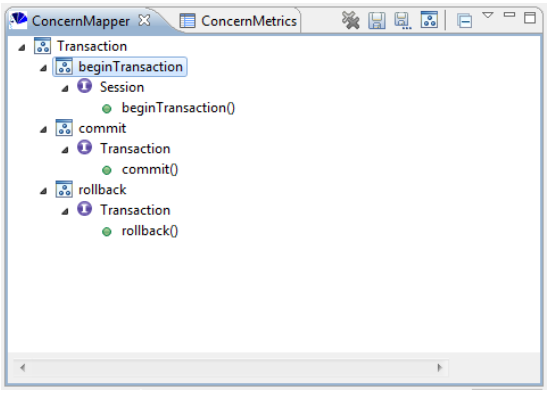


Figura 23: *Modelo de Concerns* do interesse transversal *transaction* do sistema JAccounting.

Baseado no *Modelo de Concerns* de JAccounting, a ferramenta calcula os valores das métricas na aba *ConcernMetrics*. A Tabela 8 apresenta os resultados de QD e SR

calculados a partir da ferramenta *ConcernMeetrics* na coluna **CM**, esses valores foram calculados com base no código orientado por objetos do sistema. A coluna **AM** mostra os valores da análise manual apresentada na Seção 5.1 deste capítulo para o interesse transversal *transaction*.

Interesses	**AM**		**CM**	
	SR	**QD**	**SR**	**QD**
beginTransaction	14	1	14	1
commit	14	1	14	1
rollback	13	1	13	1

Tabela 8: Valores de SR e QD para o interesse transversal *transaction* de JAccounting.

Análise dos resultados: Comparando os resultados atuais calculados pela ferramenta *ConcernMetrics*, coluna **CM**, e os resultados apresentados na análise manual, coluna **AM**, observa-se que a ferramenta obteve precisamente os mesmos valores medidos com base no código fonte orientado por aspectos. Conforme já avaliado na Seção 5.1, e agora mostrado pela ferramenta *ConcernMetrics*, através dos cálculos de QD e SR, o interesse transversal *transaction* apresenta uma característica homogênea e foi possível a modularização dos seus métodos `beginTransaction`, `commit` e `rollback` através de 3 *advices*. Isso significa que a quantificação desse interesse transversal é grande e indica uma característica positiva a ser considerada para decisão da refatoração.

A ferramenta *ConcernMetrics* calcula as métricas de separação de interesses CDC e CDO para o código orientado por objetos. No Capítulo 3 foi feita uma análise manual da aplicação dessas métricas na versão do sistema orientada por objetos. Os resultados dos cálculos destas métricas para o sistema JAccounting feito pela ferramenta *ConcernMetrics* é apresentado na Tabela 9 coluna **CM**, esse resultado é comparado ao resultado obtido pela análise manual na coluna **AM**.

Interesses	**AM**		**CM**	
	CDC	**CDO**	**CDC**	**CDO**
beginTransaction	8	8	8	8
commit	8	8	8	8
rollback	7	7	7	7

Tabela 9: Valores de CDC e CDO para o interesse transversal *transaction* de JAccounting do código fonte orientado por objetos.

Análise dos resultados: Conforme observado na Tabela 9, o resultado das métri-

cas CDC e CDO obtidos pela ferramenta *ConcernMetrics* foram idênticos aos resultados da aplicação manual dessas métricas do sistema JAccounting. Através de *ConcernMetrics* é possível a identificação dos interesses transversais, assim como das classes e métodos que acessam esses interesses, diretamente no código fonte do sistema.

A ferramenta *ConcernMetrics* foi utilizada ainda neste estudo de caso para prever os valores para as métricas CDC e CDO, caso o sistema seja refatorado para aspectos. O resultado desse cálculo é demonstrado na Tabela 10, coluna **CM**. Esse cálculo é baseado no código orientado por objetos e é feita uma análise dos interesses transversais estimando-se o valor dessas métricas com base nas fórmulas definidas no Capítulo 4 desta dissertação. Os valores obtidos pela ferramenta são comparados com os valores calculados manualmente, com base no código fonte original orientado por aspectos, apresentados na coluna **AM**.

Interesse	**AM**		**CM**	
	CDC	**CDO**	**CDC**	**CDO**
beginTransaction	1	1	1	1
commit	1	1	1	1
rollback	1	1	1	1

Tabela 10: Valores de CDC e CDO para o interesse transversal *transaction* de JAccounting do código fonte orientado por aspectos.

Análise dos resultados: Conforme a Tabela 10 as estimativas do cálculo das métricas CDC e CDO feitos pela ferramenta *ConcernMetrics* para o sistema JAccounting foram idênticos aos cálculos feitos diretamente no código orientado por aspectos. A ferramenta *ConcernMetrics*, a partir do código orientado por objetos do sistema, obteve valores corretos para o cálculo das métricas de separação de interesses.

A ferramenta *ConcernMetrics*, aplicada ao sistema JAccounting, apresentou os resultados corretos do cálculos das métricas QD e SR propostas nesta dissertação e também das métricas convencionais de separação de interesses CDC e CDO, sendo essas últimas calculadas para o código orientado por objetos e também a estimativa dos valores para o código orientado por aspectos. O resultado obtido pela ferramenta *ConcernMetrics* mais uma vez confirma que o interesse transversal *transaction* contém um alto grau de quantificação, ou seja, o resultado de sua quantificação será positivo pois consegue retirar diversas chamadas do interesse transversal espalhadas e entrelaçadas pelo código fonte e modularizá-las em apenas 3 *advices*, tornando mais fácil a manutenção, reutilização e modularização.

5.2.2 Exemplo 2 - JSpider

O sistema JSpider é o segundo estudo de caso utilizado para validação dos cálculos da ferramenta *ConcernMetrics*. O interesse transversal que foi analisado é o *logging*, este interesse se divide nos seguintes interesses atômicos: `debug, error, fatal, info` e `warn`, seu *Modelo de Concerns* é apresentado na Figura 14. A ferramenta *ConcernMetrics* foi utilizada para calcular o valor das métricas de quantificação de QD e SR, o resultado do cálculo dessas métricas é apresentado na Tabela 11, coluna **CM**, que compara os valores obtidos pela ferramenta *ConcernMetrics* com os resultados obtidos do cálculo manual apresentado na Seção 5.1, coluna **AM**.

Interesse		**AM**		**CM**	
		SR	**QD**	**SR**	**QD**
1	debug(Object)	1	0.02	4	0.09
2	debug(Object,Throwable)	0	0.00	0	0.00
3	error(Object)	1	0.09	1	0.08
4	error(Object,Throwable)	0	0.00	0	0.00
5	fatal(Object,Throwable)	0	0.00	0	0.00
6	info(Object)	6	0.14	6	0.14
7	warn(Object)	0	0.00	0	0.00

Tabela 11: Cálculo de QD e SR para o interesse transversal *logging* do sistema JSpider.

Análise dos resultados: Como podemos observar na Tabela 11, os valores indicados pelo *ConcernMetrics* se igualam aos valores reais medidos no código fonte orientado por aspectos, em cinco dos sete métodos considerados na avaliação (mais especificamente, os métodos 2, 4, 5, 6 e 7). Porém alguns resultados apresentaram divergências, estes são explicados da seguinte forma:

- A versão orientada por aspectos de JSpider não contém algumas chamadas do interesse transversal *logging*, ou seja, há chamadas de *logging* que existem na versão orientada por objetos, mas não são implementadas em nenhum *advice* na versão orientada por aspectos. Por exemplo, há 46 chamadas do método `debug(Object)`, na versão orientada por objetos analisada pela ferramenta *ConcernMetrics*, no entanto, na versão orientada por aspectos apenas 45 de tais chamadas são implementadas pelos *advices*. Esse fato justifica a diferença do valor calculado pela ferramenta *ConcernMetrics* encontrado para esse interesse.

- Duas novas chamadas do método `error(Object)` foram implementados na versão

orientada por aspectos, essas não existiam no código orientado por objetos. Não há justificativa para sua inclusão, porém com essa verificação se justifica a diferença dos valores calculados manualmente e pela ferramenta.

- Em algumas situações, há chamadas consecutivas para o mesmo método de *logging* no código orientado por objetos. Por exemplo, a Figura 24 mostra três chamadas consecutivas para `debug(Object)` no construtor da classe `EventDispatcherImpl`. O algoritmo para estimativas de métricas utilizado em *ConcernMetrics*, definido no Capitulo 4, considera que chamadas consecutivas para métodos incluídos no mesmo *Modelo de Concerns* pode ser extraído para um mesmo *advice*. No entanto, no caso particular da Figura 24, os desenvolvedores de JSpider decidiram implementar essas chamadas em *advices* diferentes, justificando a diferença entre os valores encontrados.

```
class EventDispatcherImpl {
   public EventDispatcherImpl (...) {
     ...
     log.debug("EventFilter for engine events = " + ...);
     log.debug("EventFilter for monitor events = " + ...);
     log.debug("EventFilter for spider events = " + ...);
     ...
}
```

Figura 24: Chamadas consecutivas de `debug` em JSpider.

A ferramenta *ConcernMetrics* foi usada para calcular o valor das métricas CDC e CDO para a versão original do sistema orientado por objetos, e também para realizar uma estimativa de qual seriam os valores dessas métricas caso aspectos sejam usados para modularizar o código de *logging*. Por fim, as estimativas produzidas pela ferramenta foram comparadas com o valor efetivo das métricas CDC e CDO calculados manualmente no código orientado por objetos e no código refatorado para aspectos. A Tabela 12 apresenta o resultado dos cálculos para a métrica CDC.

Análise dos resultados: Conforme mostrado na Tabela 12, em relação à métrica CDC, pode-se observar que não houve diferença entre a estimativa calculada pela ferramenta *ConcernMetrics* e o valor efetivamente medido manualmente nas versões orientadas por objetos e orientadas por aspectos do sistema JSpider. Para todos os métodos avaliados o valor de CDC, na versão orientada por aspectos, foi igual a 2 (dois). O motivo

Interesse		CDC				
		AM-OO	**AM-OA**	**CM-OO**	**CM-OA**	**Comp.**
1	debug(Object)	16	2	16	2	0
2	debug(Object,Throwable)	2	2	2	2	0
3	error(Object)	6	2	6	2	0
4	error(Object,Throwable)	24	2	24	2	0
5	fatal(Object,Throwable)	2	2	2	2	0
6	info(Object)	16	2	16	2	0
7	warn(Object)	3	2	3	2	0

Tabela 12: Métrica CDC para o interesse *logging* nas versões orientadas por objetos e orientadas por aspectos do sistema JSpider, bem como estimativa para essas métricas produzida pela ferramenta *ConcernMetrics* (CM).

é que essa métrica conta o número de componentes que implementam e acessam a funcionalidade *logging*, no caso do JSpider apenas a classe `Log` e um aspecto, o qual passa a confinar todo o código de *logging* do sistema.

Os cálculos da métrica CDO para o sistema JSpider apresentaram algumas divergências em relação à avaliação feita manualmente do sistema. Os valores obtidos para o cálculo CDO são apresentados na Tabela 13.

Interesse		CDO				
		AM-OO	**AM-OA**	**CM-OO**	**CM-OA**	**Comp.**
1	debug(Object)	47	43	47	43	0
2	debug(Object,Throwable)	13	14	13	13	-1
3	error(Object)	14	12	14	13	+1
4	error(Object,Throwable)	71	68	71	71	+3
5	fatal(Object,Throwable)	2	2	2	2	0
6	info(Object)	46	39	46	40	+1
7	warn(Object)	3	3	3	3	0

Tabela 13: Métrica CDO para o interesse *logging* calculado a partir do código orientado por objetos e do código orientado por aspectos do sistema JSpider, bem como estimativa para essas métricas produzida pela ferramenta *ConcernMetrics*(CM).

Análise dos Resultados: Como pode ser observado na Tabela 13, a estimativa para a métrica CDO para o código orientado por objetos calculado pela ferramenta *ConcernMetrics* apresentou um resultado idêntico à análise manual do sistema. Porém a estimativa do valor dessa mesma métrica para o código orientado por aspectos apresentou algumas divergências. Os valores coincidiram em três dos sete métodos avaliados (métodos 1, 5 e 7), para os métodos onde houve uma diferença, foi realizada uma inspeção

manual com o objetivo de tentar explicar as diferenças. Os resultados dessa inspeção são reportados abaixo:

- O valor de CDO do método `debug(Object,Throwable)`, calculado pela ferramenta *ConcernMetrics*, é menor em uma unidade do que o valor medido na versão orientada por aspectos. Essa diferença se justifica pelo fato de uma chamada extra a esse método ter sido adicionada ao código da versão refatorada para aspectos, chamada essa que não existe na versão orientada por objetos do sistema. Consequentemente, um adendo a mais teve que ser criado.

- Foram observadas também diferenças nos valores de CDO para os métodos `error(Object)` e `error(Object,Throwable)`. Essas diferenças se explicam pelo fato de uma chamada ao método `error(Object)` e três chamadas ao método `error(Object,Throwable)` terem sido eliminadas do código da versão orientada por aspectos.

- O valor de CDO do método `info(Object)` calculado pela ferramenta *ConcernMetrics* é maior em uma unidade do que o valor medido na versão orientada por aspectos. Essa diferença é justificada pelo fato de uma chamada a esse método ter sido removida do código orientado por objetos.

A partir das análises do estudo de caso JSpider, pode-se concluir que os cálculos feitos pela ferramenta *ConcernMetrics* foram aqueles esperados em uma ferramenta que trabalha analisando apenas o código orientado por objetos do sistema. Na verdade, constatou-se que as refatorações realizadas no sistema JSpider para modularização do código de *logging* para aspectos implicaram em alterações no comportamento original do sistema em determinados pontos. Como descrito anteriormente, chamadas a *logging* foram acrescentadas em alguns casos e removidas em outros. Essas alterações podem ter ocorrido devido a novos requisitos que foram levantados durante a implementação dos aspectos ou então por um descuido ou esquecimento dos responsáveis por essa tarefa.

Assim, conclui-se que a ferramenta *ConcernMetrics* estimou corretamente os valores das métricas de quantificação QD e SR e também os valores das métricas de separação de interesse CDC e CDO, tanto no código orientado por objetos, quanto caso refatorações para extração de aspecto venham a ser aplicadas para modularizar o interesse de *logging* no sistema JSpider. Através dos resultados apresentados por *ConcernMetrics* pode-se afirmar ainda que os valores quantitativos obtidos pelas métricas foram baixos, o que significa que a refatoração desses interesses para aspectos não irá apresentar grandes van-

tagens, pois na maioria das vezes as chamadas irão somente ser trocadas de lugar, sendo necessário uma grande quantidade de *advices* para poder atingir a todos os *join points*.

5.2.3 Exemplo 3 - JHotDraw

O sistema JHotDraw é utilizado como o terceiro estudo de caso para validação da ferramenta *ConcernMetrics*. Conforme já relatado na Seção 5.1, dois métodos são utilizados para estudo de caso da aplicação das métricas QD e SR, estes também serão utilizadas agora para validação da ferramenta *ConcernMetrics*. Na Figura 25 é apresentado o *Modelo de Concerns* feito através da ferramenta *ConcernMetrics* para modelagem dos interesses a serem analisados.

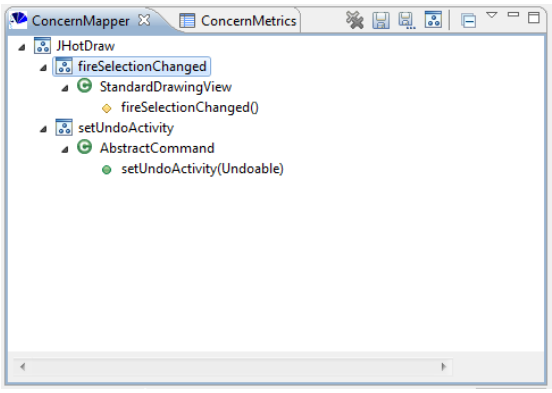


Figura 25: *Modelo de Concerns* do sistema JHotDraw.

A ferramenta *ConcernMetrics* calculou as métricas QD e SR para o sistema JHotDraw, conforme apresentado na Tabela 14. A coluna **AM** mostra os valores obtidos da avaliação manual diretamente feita no código fonte orientado por aspectos do sistema e a coluna **CM** os valores estimados pela ferramenta *ConcernMetrics*.

Interesse		**AM**		**CM**	
		SR	**QD**	**SR**	**QD**
1	fireSelectionChanged()	2	0.67	3	1
2	setUndoActivity(Undoable)	1	0.13	11	0.92

Tabela 14: Resultado de QD e SR para JHotDraw.

Análise dos resultados: Conforme os resultados apresentados na Tabela 14 algumas diferenças foram encontradas entre a avaliação feita diretamente no código orientado por aspectos e pelas estimativas da ferramenta *ConcernMetrics*. Essas são explicadas a seguir.

- As chamadas para `fireSelectionChanged` poderiam ter sido confinadas em um *advice* único, porém na versão refatorada para aspectos estas chamadas foram implementadas em dois *advices*.
- No caso de `setUndoActivity(Undoable)` na versão orientada por aspectos do sistema, não foram extraídas quatro chamadas do interesse para aspectos, nove chamadas foram refatoradas para aspectos e uma nova chamada foi inserida. Foi verificado no código fonte orientado por aspectos que 10 *join points* foram modularizados por 9 *advices*. A partir de uma inspeção no código fonte orientado por aspectos, com o intuito de explicar os motivos dessas implementações, foi verificado que na refatoração do interesse `setUndoActivity(Undoable)`, foram utilizados diferentes *pointcuts* para atender aos *join points* do interesse. Desta forma, foram necessários nove *advices* para implementar as 10 chamadas desse interesse transversal. Uma análise baseada no código fonte orientado por objetos e no código fonte orientado por aspecto foi feita e concluiu-se que caso o interesse transversal `setUndoActivity(Undoable)` tivesse sido implementado através de um único *pointcut*, a quantidade correta de *advices* necessários para modularizá-los seriam 2 *advices* e iriam atingir a todos os 13 *join points*, conforme foi calculado pela ferramenta *ConcernMetrics*.

A partir do estudo de caso JHotDraw, pode-se verificar que a ferramenta *ConcernMetrics* obteve os valores corretos para as métricas QD e SR. Além da avaliação de QD e SR para JHotDraw, também foram feitas as avaliações do cálculo das métricas CDC e CDO para o código orientado por objetos, de acordo com o cálculo da ferramenta *ConcernMetrics*. Os resultados são demonstrados na Tabela 15. Os valores de CDC e CDO, conforme análise manual do código fonte orientado por objetos e o resultado obtido através da ferramenta *ConcernMetrics*, foram idênticos, mostrando mais uma vez a eficácia da ferramenta.

As métricas CDC e CDO, quando foram calculados manualmente no código fonte orientado por aspectos, resultaram nos valores apresentados na Tabela 16, os resultados por sua vez foram diferentes dos resultados estimados pela ferramenta *ConcernMetrics*.

Interesse		AM		CM	
		CDC	CDO	CDC	CDO
1	fireSelectionChanged()	1	5	1	5
2	setUndoActivity(Undoable)	13	13	13	13

Tabela 15: Resultado de CDC e CDO para os interesses transversais de JHotDraw do código fonte orientado por objetos.

Interesse		AM		CM	
		CDC	CDO	CDC	CDO
1	fireSelectionChanged()	2	3	2	2
2	setUndoActivity(Undoable)	10	9	2	3

Tabela 16: Resultado de CDC e CDO para os interesses transversais de JHotDraw do código fonte orientado por aspectos.

Análise dos resultados: Algumas diferenças foram encontradas para os valores de CDC e CDO para o cálculo do código Orientado por Aspectos. Essas divergências se justificam por modificações encontradas no ato da implementação dos interesses através de aspectos. Desta forma, ocasionou as diferenças encontradas para os valores de CDC e CDO e também as divergências verificadas nos cálculos QD e SR, conforme já apresentado na Tabela 14. As diferenças são mais bem detalhadas abaixo:

- No código orientado por aspectos, foi verificado que o interesse `fireSelectionChanged()` foi modularizado através de dois *advices*, resultando um valor para CDO igual a 3. A ferramenta *ConcernMetrics* por sua vez, considerou que seria necessário somente 1 *advice* para modularizar os 4 *join points*, logo o resultado estimado para a métrica CDC foi igual a 2.

- O método `setUndoActivity(Undoable)` apresentou maiores diferenças, isso se deve ao fato que somente dez chamadas de `SetUndoActivity(Undoable)` foram refatoradas para aspectos, estas chamadas moram modularizadas através de nove *advices*, causando a diferença no valor de CDC e CDO para o valor calculado com base no código fonte orientado por aspectos e o estimado pela ferramenta.

A ferramenta *ConcernMetrics* apresentou resultados corretos para a avaliação do sitema JHotdraw, e, de acordo com os resultados apresentados, pode-se verificar que os interesses analisados, quando refatorados separadamente de quaisquer interesses do sistema, irão apresentar alto grau de quantificação, ou seja, um número grande de chamadas

serão refatoradas em uma quantidade pequena de *advices*. Isso se identifica tanto pelo cálculo das métricas de quantificação QD e SR quanto pela análise das métricas CDC e CDO.

5.2.4 Análise ConcernMetrics

Conforme os estudos de caso apresentados a ferramenta *ConcernMetrics* apresentou uma estimativa correta para os interesses transversais dos sistemas JAccounting, JSpider e JHotDraw. Os valores estimados pela ferramenta foram comparados com os valores avaliados manualmente diretamente no código fonte orientado por objetos e também nas versões orientada por aspectos dos sistemas. Isso possibilitou uma validação do cálculo das métricas, tanto para a validação das métricas de quantificação QD e SR, quanto para as métricas já conhecidas de separação de interesses CDC e CDO.

Como análise geral, os resultados foram totalmente corretos para o cálculo das métricas QD e SR, concluindo que a ferramenta *ConcernMetrics* é eficiente para a avaliação quantitativa de interesses transversais de sistemas orientados por objetos Java na ferramenta Eclipse. A ferramenta *ConcernMetrics* apresentou também 100% de acerto no cálculo de CDC e CDO para o código orientado por objetos, para a estimativa de CDC e CDO para o código orientados por aspectos identificou-se algumas diferenças entre os resultados de *ConcernMetrics* e os resultados obtidos a partir da análise feita no código fonte refatorado para aspectos. Essas divergências, porém, foram justificadas porque nas versões refatoradas para aspectos foram feitas implementações diferentes acrescentando ou retirando chamadas dos interesses do sistema, além de tomarem decisões de implementação dos interesses em mais *advices* do que seriam necessários para refatoração. A ferramenta *ConcernMetrics* estimou os valores corretos do cálculo das métricas CDC e CDO para o código orientado por aspectos, analisando somente o código fonte orientado por objetos, seguindo-se os algoritmos de decisão descritos no Capítulo 4 e analisando os interesses atômicos conforme definido pelo *Modelo de Concerns*, ou seja, análise de somente uma interesse sem influência de demais interesses que não estejam incluídos no *Modelo de Concerns*.

A ferramenta *ConcernMetrics* possibilita a modelagem do *Modelo de Concerns* que serve para raciocinar sobre a separação de interesses de um sistema, calcula com eficácia o grau de quantificação dos interesses selecionados no *Modelo de Concers*, possui uma interface amigável para cálculo e visualização do resultado dos cálculos das métricas, facilitando a avaliação dos valores de QD, SR, CDC e CDO para tomada de decisões para

refatoração de sistemas para orientação por aspectos.

5.3 Discussão

Nesta seção, é discutida a importância da quantificação para a refatoração de interesses transversais em sistemas orientados por objetos para aspectos. É analisado como essa quantificação possibilita uma avaliação feita diretamente no código fonte original do sistema, sem a necessidade de qualquer alteração estrutural do sistema, possibilitando uma análise do resultado que será obtido previamente à refatoração.

De acordo com o resultado da métrica de quantificação QD para os sistemas JAccounting, JSpider e JHotDraw apresentados nas Tabelas 5, 6 e 7 respectivamente, é possível visualizar o valor quantitativo de cada interesse analisado. O sistema JAccouting contém um alto grau de quantificação, o sistema JSpider apresenta um grau de quantificação baixo e o sistema JHotDraw apresenta também um alto grau de quantificação.

A quantificação é um fator de avaliação importante para a decisão da refatoração de interesses transversais para aspectos. A análise quantitativa apresentada no Capítulo 3 mostra o cálculo de métricas de separação de interesses para sistemas orientados por aspectos. Quando seus resultados são analisados quantitativamente, demonstram valores que espelham os resultados reais da refatoração, ou seja, os sistemas que apresentaram melhor resultado da refatoração para aspectos, também apresentaram os melhores resultados da quantificação.

Essa mesma interpretação é feita a partir da análise dos resultados obtidos da métrica QD, onde mostra o grau de quantificação dos sistemas JAccouting, JSpider e JHotDraw. Esses resultados são compatíveis com a análise quantitativa feita baseada nas métricas CDC e CDO. Outro fator para validação da quantificação como critério de decisão da refatoração de sistemas para aspectos é o cálculo da métrica SR. Este retorna o valor específico da redução do espalhamento de código pelo sistema, o qual é um dos principais problemas causados pelos interesses transversais (KICZALES et al., 1997; SOMMERVILLE, 2006).

É importante ressaltar porém que a quantificação e a redução do espalhamento não devem ser os únicos fatores a ser analisados na decisão de refatoração para aspectos de interesses transversais. Fatores como transformação de código (MALTA; VALENTE, 2009), separação de interesses (SANT'ANNA et al., 2003), complexidade, entre outros, também devem ser avaliados, além de necessidades específicas para cada projeto de software.

Exemplos de situações que não podem ser previstas são mostrados nos estudos de caso dos sistemas JSpider e JHotDraw. Nestes exemplos, alguns interesses transversais homogêneos foram implementados em mais de um *advice*, enquanto poderiam ter sido modularizados através de um único *advice*. Desta forma, a implementação feita resultou em um valor diferente do que a análise de uma ferramenta baseada somente no código fonte pudesse identificar.

A análise quantitativa porém se mostrou eficaz nos estudos de caso apresentados, resultando em valores similares aos comparados com a análise feita diretamente no código refatorado desses sistemas, assim como o cálculo automatizado dessa quantificação em sistemas Java pela ferramenta *ConcernMetrics*. Isso mostra que a quantificação deve ser um requisito importante para a análise da tomada de decisões para a refatoração de sistemas para aspectos.

5.4 Considerações Finais

Neste capítulo, foram apresentados os resultados de três estudos de caso de sistemas reais realizados para avaliação da aplicação das métricas QD e SR propostas no Capítulo 3, esses mesmos estudos de caso também serviram para a validação da ferramenta *ConcernMetrics*. Nas avaliações foram aplicadas manualmente as métricas QD e SR nos sistemas JAccounting, JSpider e JHotDraw, os resultados obtidos foram comparados com análise quantitativa apresentada no Capítulo 3, baseado na aplicação das métricas de quantificação CDC e CDO nos sistemas orientados por objetos e também nas versões dos mesmos orientadas por aspectos.

Através da ferramenta *ConcernMetrics* foram feitos os cálculos para as métricas QD e SR para os estudos de caso, o cálculo das métricas CDC e CDO para o código orientado por objetos e também a estimativa dos valores destas métricas caso o sistema fosse refatorado para aspectos. Esses resultados foram analisados separadamente e discutidos nesse capítulo divergências encontradas. A ferramenta demonstrou eficácia no cálculo das métricas, facilidade para a montagem do *Modelo de Concerns* e possibilita uma avaliação totalmente baseada no código orientado por objetos sem a necessidade de nenhuma alteração estrutural do sistema.

A partir das análises manuais feitas para validação dos estudos de caso dessa dissertação, constatou-se a real necessidade de ferramentas para automatização do cálculo de métricas de software. Esse fato se justifica pelo grau de dificuldade encontrado para a

inspeção manual dos sistemas JAccounting, JSpider e JHotDraw. Embora a ferramenta Eclipse facilite a identificação das chamadas de métodos através da funcionalidade *Call Hierarchy*, ainda são necessários diversos outros passos para o cálculo das métricas. É necessária a avaliação dos *join points*, a definição dos interesses homogêneos e heterogêneos, a análise e decisão dos *advices* para modularizarem todos os *join points*, identificação de classes e métodos que implementam e acessam determinado interesse, e finalmente o cálculo das métricas.

A ferramenta *ConcernMetrics* mostrou eficácia na automatização do processo de análise, definição e cálculo de métricas, baseado somente nos interesses identificados no *Modelo de Concers* e no código fonte orientado por objetos. Desta forma, tornando mais fácil a avaliação quantitativa de sistemas orientados por objetos previamente à refatoração para aspectos, além de evitar possíveis erros de avaliação e identificação de interesses transversais que podem ocorrer através de uma análise manual.

As discussões por sua vez abordam a importância da quantificação para a tomada de decisões para a refatoração de sistemas para aspectos, também discute fatores externos à análise quantitativa como decisões de alteração estrutural de projeto no momento da refatoração, que podem influenciar tanto quanto a quantificação nessa tomada de decisão.

6 CONCLUSÕES

O paradigma de programação orientada por aspectos se consolida cada vez mais como a melhor tecnologia utilizada para separação e modularização de interesses transversais. Esse fato pode se confirmar a partir de diversas pesquisas sobre o assunto, onde são relatados trabalhos sobre refatorações orientadas a aspectos que descrevem como modularizar interesses transversais (BINKLEY et al., 2005, 2006; COLE; BORBA, 2005; LADDAD, 2006; MONTEIRO; FERN, 2006), e também trabalhos que apresentam estudos de caso sobre a extração e modularização de interesses transversais por meio de aspectos (GARCIA et al., 2005; HANNEMANN; KICZALES, 2002; APEL; LEICH; SAAKE, 2006; FIGUEIREDO et al., 2008; KASTNER; APEL; BATORY, 2007; MENDHEKAR; KICZALES; LAMPING, 1997). No entanto, esses estudos, em geral, não abordam a questão sobre como decidir quando as refatorações propostas irão trazer benefícios ao sistema. Nos trabalhos citados, observa-se que alguns apresentaram benefícios na refatoração, porém outros praticamente retiraram as chamadas do interesse do código e foram transferidas para o aspecto, isso pode trazer dificuldades para testes, manutenção, complexidade e inteligibilidade. Nesses trabalhos não foram feitas avaliações prévias sobre os reais benefícios da refatoração e somente alguns apresentaram uma avaliação após o processo, o que poderia ter evitado o esforço da refatoração caso fosse constatado que não se obteria um resultado positivo.

Para tratar da necessidade de análise de projetos de refatoração para aspectos, alguns trabalhos são propostos com o intuito de avaliar as vantagens e desvantagens das refatorações (SANT'ANNA et al., 2003; CECCATO; TONELLA, 2004; EADDY et al., 2008; LOPEZ-HERREJON; APEL, 2007; APEL, 2010; KULESZA et al., 2006; STEIMANN, 2006). Essas avaliações são feitas em sua maioria através de métricas de software e atributos específicos para aspectos. Também são utilizadas medidas comuns para projetos de software convencionais ou orientados por objetos que são aplicáveis a sistemas orientados por aspectos. Nos estudos apresentados, poucos utilizam o conceito de quantificação, que conforme Filman e Friedman (2005) é uma das características mais favoráveis para a constatação da eficácia da utilização de aspectos. Outro fator a ser considerado nos tra-

balhos avaliados é que esses, em sua maioria, são feitos com base no código já refatorado para aspectos, nenhum se propondo a prever uma possível avaliação prévia da refatoração totalmente feita com base no código fonte original.

A motivação principal para a pesquisa desenvolvida nesta dissertação surgiu a partir da observação de que trabalhos na área de refatoração de interesses transversais dificilmente utilizavam-se de uma análise para a avaliação do resultado da refatoração. O fator proposto para essa medida é a quantificação, que é uma medida imprescindível quando se trata da modularização de interesses transversais através de aspectos e que é pouco utilizada nos trabalhos atuais de propostas de avaliações destes. Além disso, trabalhos feitos para prover essa avaliação não apresentam um resultado prévio à migração do sistema para aspectos, sendo necessário dispensar tempo e esforço para o processo sem saber ao certo se essa refatoração pode ocasionar um resultado positivo ou negativo ao projeto.

6.1 Contribuições

A quantificação permite aos desenvolvedores implementar através de um único *advice* o código espalhado em diversas posições estáticas do sistema orientado por objetos, desde que esses sejam interesses homogêneos. Sendo assim, a quantificação é um mecanismo chave para a avaliação e tomada de decisão da refatoração de sistemas para aspectos.

A fim de mais bem avaliar as vantagens da quantificação, foram propostas duas métricas: *Quantification Degree* (QD) e *Scattering Reduction* (SR), que medem respectivamente o grau de quantificação e a redução do espalhamento de código de determinado interesse. Essas métricas foram descritas e definidas formalmente e seu uso exemplificado utilizando-se sistemas reais. De acordo com estudos de caso se validou a eficácia das métricas para medir o grau de quantificação de interesses transversais.

Como mais uma contribuição desta dissertação, foi implementada a ferramenta *ConcernMetrics*, um *plugin* para a ferramenta de desenvolvimento Java Eclipse, que calcula as métricas QD e SR e também as métricas de separação de interesse CDC e CDO. Esses cálculos são feitos sem requerer a extração física de interesses transversais do código fonte para aspectos, com o intuito de ser utilizada para calcular antecipadamente as métricas propostas de quantificação, fornecendo informações para a avaliação das vantagens de usar aspectos em sistemas Java. Apresentou-se uma descrição do projeto desta ferramenta,

suas principais funcionalidades e as limitações de sua atual implementação. A ferramenta *ConcernMetrics* foi utilizada para a avaliação em três sistemas de médio porte em Java, JAccounting, JSpider e JHotDraw. De acordo com pesquisas feitas, *ConcernMetrics* é a primeira ferramenta que fornece estimativas de métricas para análise de código orientado por aspectos, totalmente baseado no código orientado por objetos.

Com o apoio de *ConcernMetrics*, foram realizados estudos de caso envolvendo os sistema JSpider, JAccounting e JHotDraw. Foram aplicadas as métricas QD e SR, CDC e CDO para o código orientado por objetos e estimados os valores dessas métricas para o código orientado por aspectos. As principais lições aprendidas nesses estudos de caso são resumidas a sequir:

- os resultados obtidos evidenciam quantificação como uma medida imprescindível para avaliação de extrações de interesses para aspectos;
- a partir da inspeção manual no código dos sistemas, podem ocorrer erros na identificação de interesses transversais, além do alto esforço necessário para esse processo. O uso de uma ferramenta como a *ConcernMetrics* pode evitar essas situações;
- a localização de interesses transversais, assim como seus *join points*, em um sistema é uma tarefa complexa e trabalhosa, além disso a definição de *advices* que atendam à todos os *join points* é complexa e aberta a erros; mais uma vez o uso de uma ferramenta como a *ConcernMetrics* pode evitar esses possíveis problemas.

6.2 Comparação com Outros Trabalhos

A seguir serão apresentados trabalhos relacionados ao tema desta dissertação:

6.2.1 Métricas de Software Orientado por Aspectos

As métricas de separação de interesses CDC e CDO, que foram utilizadas para avaliar os benefícios da quantificação apresentada no Capítulo 3, foram propostas por Sant'Anna et al. (2003). Em outro trabalho, Garcia et al. (2005) utiliza as métricas CDC e CDO para avaliar os benefícios dos aspectos na implementação de padrões de projeto, além disso propôs adaptações para as métricas de acoplamento e coesão do conjunto de métricas CK (CHIDAMBER; KEMERER, 1994), que conforme as métricas CDC e CDO, avaliam características específicas de um projeto já refatorado para aspectos. Esta dissertação

propõe uma abordagem de análise do grau de quantificação de interesses transversais, através das métricas QD e SR, previamente a qualquer alteração estrutural do sistema.

A métrica CDA, proposta por Ceccato e Tonella (2004), utiliza o conceito de quantificação para medir o grau de espalhamento de um aspecto. Esta métrica conta o número de módulos afetados pelos *pointcuts* e pela introdução de um determinado aspecto. Porém ao se comparar a métrica proposta QD com a métrica CDA, verifica-se que CDA não apresenta um comportamento relacionado a nenhum modelo de *concerns*. Além disso, apresenta uma noção básica sobre a quantificação baseada somente no código orientado por aspectos. A métrica QD, por sua vez, apresenta uma medida no intervalo de 0 e 1 para a medida do grau de quantificação com base no código original do sistema, e um relacionamento através do *Modelo de Concerns*, possibilitando o mapeamento dos interesses transversais e o cálculo da métrica.

A métrica CRR é uma métrica proposta para avaliar sistemas em AspectJ, basicamente mede o número de linhas de código que poderia ser reduzido através da utilização de aspectos (APEL, 2010) . Existem duas diferenças principais entre CRR e a métrica SR: (1) em primeiro lugar CRR resulta o número de linhas de código reduzidos baseado no código orientado por aspectos, por outro lado, a métrica SR conta o número de chamadas que serão reduzidas (menos um) do código orientado por objetos. (2) em segundo lugar, CRR dá um número único e global para o sistema como um todo, enquanto a métrica SR é calculada para as preocupações individuais previamente organizadas em um *Modelo de Concerns* hierárquico.

6.2.2 Avaliações Orientadas por Aspectos

Diversos trabalhos são realizados para validação da refatoração de softwares para aspectos utilizando-se as métricas de separação de interesses e as métricas adaptadas pelas métricas CK (SANT'ANNA et al., 2003). Para estas validações foram utilizados exemplos de padrões de projeto (GARCIA et al., 2005; SANT'ANNA et al., 2003), manipulação de exceção (FILHO et al., 2006), sistemas de informação baseados na Web (KULESZA et al., 2006) e as linhas de produto de software (FIGUEIREDO et al., 2008). Em geral, em cada um destes estudos empíricos foram identificados efeitos positivos e negativos da utilização de aspectos. Na maioria dos estudos, as situações em que eles não recomendam o uso de aspectos podem ser relacionados com um reduzido grau de quantificação.

No trabalho de Kastner, Apel e Batory (2007), foram relatadas as experiências na refatoração de *features* da Oracle Berkeley DB para aspectos. Nesse trabalho, foi obser-

vado que os elementos extraídos, em geral, apresentam um pequeno grau de quantificação. Além disso, há o relato de que a maioria dos *pointcuts* definidos estão intimamente ligados ao programa base e, portanto, são especialmente frágeis às modificações no programa. Foi analisado em Apel (2010) o uso de AspectJ em onze sistemas através das métricas CIA, CRR, SDC, entre outras. Constatou-se que apenas 2% utilizam avançados mecanismos transversais, incluindo *advices* que afetam conjuntos inteiros de *join points*.

Acredita-se que através da análise das métricas QD e SR, principalmente suportada pela ferramenta *ConcernMetrics*, seria possível chegar às mesmas conclusões sem extrair qualquer parte do interesse do código original orientado por objetos.

6.2.3 Ferramentas

A ferramenta *ConcernTagger* é uma extensão da ferramenta *ConcernMapper* que calcula as métricas de dispersão, incluindo CDC, CDO e DOSC e DOSM já anteriormente mencionadas (EADDY et al., 2008). No entanto, pelo menos na sua versão atual, as métricas são medidas apenas para o código orientado por objetos.

Por outro lado, uma característica marcante da ferramenta *ConcernMetrics* é sua habilidade de inferir os valores de QD e SR relativos a uma eventual refatoração do código para aspecto. Calcula ainda os valores de CDC e CDO para o código orientado por objetos e estimar os valores dessas métricas para o código orientado por aspectos, sendo esses cálculos baseados totalmente no código fonte orientado por objetos e sem nenhuma alteração estrutural do sistema.

6.3 Trabalhos Futuros

Com o objetivo de validar as métricas de quantificação QD e SR, pretende-se realizar novas avaliações aplicando-as a mais estudos de caso de sistemas reais que apresentem diferentes cenários de interesses transversais. Essas validações podem ser feitas utilizando-se a ferramenta *ConcernMetrics*. Essas avaliações poderão validar a eficácia das métricas QD e SR, além de possibilitar a identificação de *bugs* e a evolução da ferramenta *ConcernMetrics*.

Pretende-se ainda incorporar à ferramenta *ConcernMetrics* o cálculo de outras métricas para avaliação de sistemas orientados por aspectos já difundidas. Métricas como CLC (GARCIA et al., 2005), DOSM e DOSC (EADDY et al., 2008) podem ser incorporadas. A partir de CLC será obtido o número de linhas que implementam um determinado

interesse, e através de DOSC e DOSM será possível uma visão do grau de distribuição do interesse entre as classes e métodos do sistema. Essas métricas poderão ser calculadas a partir do código fonte orientado por objetos e poderão ser estimados os seus valores para o código orientado por aspectos.

Além disso, propõe-se o plano de integração de *ConcernMetrics* com a ferramenta *TransformationMapper* proposta por Malta, Oliveira e Valente (2009). A ferramenta *TransformationMapper* fornece informações sobre transformações necessárias ao código orientado por objetos para permitir a extração de aspectos. Através dessa integração, a ferramenta *ConcernMetrics* irá avaliar também transformações necessárias para a refatoração de um determinado interesse para aspectos para a estimativa de *advices* necessários para essa modularização.

Finalmente, em outro estudo pretende-se abordar a correlação, caso seja possível, entre fragilidade de um *pointcut* e altos valores de QD. Dessa forma, poderia-se verificar se a fragilidade de *pointcuts*, ou seja, a exposição a impacto dos *pointcuts* por alterações no código funcional orientado por objetos, é alta quando existem altos valores de QD.

REFERÊNCIAS

ANBALAGAN, P.; XIE, T. Automated inference of pointcuts in aspect-oriented refactoring. In: *ICSE '07: Proceedings of the 29th international conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007. p. 127–136.

APEL, S. How aspectj is used: An analysis of eleven aspectj programs. *Journal of Object Technology*, v. 9, n. 1, p. 117–142, jan. 2010.

APEL, S.; LEICH, T.; SAAKE, G. Aspectual mixin layers: aspects and features in concert. ACM, New York, NY, USA, p. 122–131, 2006.

BINKLEY, D. et al. Automated refactoring of object oriented code into aspects. IEEE Computer Society, Washington, DC, USA, p. 27–36, 2005.

BINKLEY, D. et al. Tool-supported refactoring of existing object-oriented code into aspects. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 32, p. 698–717, September 2006.

BRUNETON, E.; LENGLET, R.; COUPAYE, T. Asm: A code manipulation tool to implement adaptable systems. In: *Adaptable and extensible component systems*. France: [s.n.], 2002.

CECCATO, M. Automatic support for the migration towards aspects. In: *CSMR '08: Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering*. Washington, DC, USA: IEEE Computer Society, 2008. p. 298–301.

CECCATO, M.; TONELLA, P. 1st workshop on aspect reverse engineering (ware 2004). *Measuring the effects of software aspectization*, 2004.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 20, p. 476–493, June 1994.

COLE, L.; BORBA, P. Deriving refactorings for aspectj. In: *Proceedings of the 4th international conference on Aspect-oriented software development*. New York, NY, USA: ACM, 2005. (AOSD '05), p. 123–134.

EADDY, M. et al. Do crosscutting concerns cause defects? *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 34, p. 497–515, July 2008.

EJIOGU, L. O. *Software engineering with formal metrics*. Wellesley, MA, USA: QED Information Sciences, Inc., 1991.

FENTON, N. Software measurement: A necessary scientific basis. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 20, p. 199–206, March 1994.

FENTON, N. E.; NEIL, M. Software metrics: success, failures and new directions. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 47, p. 149–157, July 1999.

FIGUEIREDO, E. et al. Evolving software product lines with aspects: an empirical study on design stability. ACM, New York, NY, USA, p. 261–270, 2008.

FIGUEIREDO, E. et al. Crosscutting patterns and design stability: An exploratory analysis. *Proc. of Intl Conf. on Program Comprehension (ICPC), Vancouver, Canada*, p. 138–147, 2009.

FIGUEIREDO, E.; WHITTLE, J.; GARCIA, A. F. Concernmorph: metrics-based detection of crosscutting patterns. In 7th International Symposium on Foundations of Software Engineering (FSE), p. 299–300, 2009.

FILHO, F. C. et al. Exceptions and aspects: the devil is in the details. ACM, New York, NY, USA, p. 152–162, 2006.

FILMAN, R. E.; FRIEDMAN, D. P. Aspect-oriented programming is quantification and obliviousness. In: FILMAN, R. E. et al. (Ed.). *Aspect-Oriented Software Development*. Boston: Addison-Wesley, 2005. p. 21–35.

FOWLER, M. et al. *Refactoring: improving the design of existing code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

GARCIA, A. et al. Assessment of contemporary modularization techniques - acom'07: workshop report. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 32, n. 5, p. 31–37, 2007.

GARCIA, A. et al. Modularizing design patterns with aspects: a quantitative study. ACM, New York, NY, USA, p. 3–14, 2005.

GRADECKI, J. D.; LESIECKI, N. *Mastering AspectJ: Aspect-Oriented Programming in Java*. New York, NY, USA: John Wiley Sons, Inc., 2003.

GREENWOOD, P. et al. On the impact of aspectual decompositions on design stability: An empirical study. In: ERNST, E. (Ed.). *ECOOP*. Berlin, Germany: Springer, 2007. (Lecture Notes in Computer Science, v. 4609), p. 176–200.

HALL, T.; FENTON, N. Implementing effective software metrics programs. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 14, p. 55–65, March 1997.

HANNEMANN, J.; KICZALES, G. Design pattern implementation in java and aspectj. ACM, New York, NY, USA, p. 161–173, 2002.

HARRISON, R.; COUNSELL, S. J.; NITHI, R. V. An evaluation of the mood set of object-oriented software metrics. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 24, p. 491–496, June 1998.

HILSDALE, E.; HUGUNIN, J. Advice weaving in aspectj. ACM, New York, NY, USA, p. 26–35, 2004.

IRWIN, J. et al. Aspect-oriented programming of sparse matrix code. Springer-Verlag, London, UK, p. 249–256, 1997.

KANER, C.; MEMBER, S.; BOND, W. P. Software engineering metrics: What do they measure and how do we know? 2004.

KASTNER, C.; APEL, S.; BATORY, D. A case study implementing features using aspectj. IEEE Computer Society, Washington, DC, USA, p. 223–232, 2007.

KICZALES, G.; HILSDALE, E. Aspect-oriented programming. In: *ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*. New York, NY, USA: ACM, 2001. p. 313.

KICZALES, G. et al. An overview of aspectj. In: *ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming*. London, UK: Springer-Verlag, 2001. p. 327–353.

KICZALES, G. et al. Aspect-oriented programming. *In 11th European Conference Object-Oriented Programming (ECOOP)*, v. 1241, p. 220–242, 1997.

KULESZA, U. et al. Quantifying the effects of aspect-oriented programming: A maintenance study. IEEE Computer Society, Washington, DC, USA, p. 223–233, 2006.

LADDAD, R. *AspectJ in Action: Practical Aspect-Oriented Programming*. Greenwich, CT, USA: Manning Publications Co., 2003. ISBN 1930110936.

LADDAD, R. *Aspect Oriented Refactoring*. Greenwich: Addison-Wesley Professional, 2006.

LOPEZ-HERREJON, R. E.; APEL, S. *Measuring and characterizing crosscutting in aspect-based programs: basic metrics and case studies*. Berlin, Heidelberg: Springer-Verlag, 2007. 423–437 p. (FASE'07).

LORENZ, M.; KIDD, J. Object-oriented software metrics: a practical guide. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

MALTA, M. N.; OLIVEIRA, S. d.; VALENTE, M. T. Guidelines for enabling the extraction of aspects from existing object-oriented code. *Jornal Of Object Technology - JOT*, v. 8, n. 3, p. 101 – 119, May - June 2009.

MALTA, M. N.; VALENTE, M. T. Object-oriented transformations for extracting aspects. *Information and Software Technology*, v. 51, p. 138 – 149, May - June 2009.

MARIN, M. et al. An integrated crosscutting concern migration strategy and its semi-automated application to jhotdraw. *Automated Software Engg.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 16, n. 2, p. 323–356, 2009.

MARIN, M.; DEURSEN, A. V.; MOONEN, L. Identifying crosscutting concerns using fan-in analysis. *ACM Trans. Softw. Eng. Methodol.*, ACM, New York, NY, USA, v. 17, n. 1, p. 1–37, 2007.

MENDHEKAR, A.; KICZALES, G.; LAMPING, J. Rg: A case-study for aspect-oriented programming. Palo Alto, CA 94304, p. 21–33, feb 1997.

MONTEIRO, M. P.; FERN, J. M. Towards a catalogue of refactorings and code smells for aspectj. *In T. Aspect-Oriented Software Development I*, Portugal, p. 214–258, 2006.

PARK, R. E.; GOETHERT, W. B.; FLORAC, W. A. *Goal Driven Software Measurement - A Guidebook*. Pittsburgh, PA 15213: Software Engineering Institute, Carnegie Mellon University,, 1996.

PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. 6. ed. New York, NY, USA: McGraw-Hill, Inc., 2005.

ROBILLARD, M. P.; WEIGAND-WARR, F. Concernmapper: simple view-based separation of scattered concerns. ACM, New York, NY, USA, p. 65–69, 2005.

ROCHE, J. M. Software metrics and measurement principles. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 19, p. 77–85, January 1994.

SANT'ANNA, C. et al. On the reuse and maintenance of aspect-oriented software: An assessment framework. *In 17th Brazilian Symposium on Software Engineering (SBES)*, p. 19–34, 2003.

SOMMERVILLE, I. *Software Engineering: (Update) (8th Edition) (International Computer Science Series)*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2006.

STEIMANN, F. The paradoxical success of aspect-oriented programming. *SIGPLAN Not.*, ACM, New York, NY, USA, v. 41, n. 10, p. 481–497, 2006.

TIRELO, F. et al. *Desenvolvimento de Software Orientado por Aspectos*. Salvador, BA, 2004.