Estimativa de Métricas de Separação de Interesses em Processos de Refatoração para Extração de Aspectos

César Francisco de Moura Couto[1], **Jaqueline Faria de Oliveira**[2], **Marco Tulio Valente**[2]

[1]Departamento de Computação, CEFET Minas
[2]Instituto de Informática, PUC Minas
cesar@decom.cefetmg.br, jaquefari@gmail.com, mtov@pucminas.br

Abstract. *This paper presents a tool that allows software maintainers to build a logical map of crosscutting concerns tangled in the implementation of object-oriented systems. Based on this map, the tool estimates the values for classical separation of concerns metrics supposing that aspects would be used to physically modularize the mapped crosscutting concerns. Therefore, the central goal of the tool is to help maintainers to evaluate the benefits of using AOP in their existing systems, without requiring the real implementation of aspects.*

Resumo. *Neste trabalho, apresenta-se uma ferramenta que permite que mantenedores de software criem um mapeamento lógico dos interesses transversais presentes em um sistema orientado por objetos. A partir desse mapeamento, a ferramenta realiza uma estimativa dos valores que seriam obtidos para métricas clássicas de separação de interesses caso o código relativo a tais interesses venha a ser fisicamente extraído para aspectos. O objetivo final é auxiliar mantenedores a avaliar os benefícios do emprego de aspectos em sistemas existentes, sem que eles tenham que concretamente partir para implementação de aspectos.*

1 Introdução

A implementação de interesses transversais (*crosscutting concerns*) em sistemas orientados por objetos ocorre de forma entrelaçada e espalhada, o que prejudica a manutenção e evolução dos mesmos. Refatorações orientadas por aspectos podem ser usadas para modularizar em aspectos a implementação de tais interesses [8, 1]. Linguagens de programação orientadas por aspectos – como AspectJ – viabilizam essas refatorações. AspectJ estende Java com novas abstrações, incluindo pontos de junção (*join points*), conjuntos de junção (*pointcuts*), adendos (*advices*) e aspectos. Em AspectJ, um aspecto permite a especificação de conjuntos de pontos bem definidos da execução de um programa (pontos de junção) e a exposição de seus contextos para a implementação de adendos. Adendos são métodos anônimos que são implicitamente invocados antes, após e no lugar dos pontos de junção capturados.

Apesar dos benefícios normalmente atribuídos a orientação por aspectos, desenvolvedores podem se deparar com a seguinte questão: vale a pena migrar um sistema existente, orientado por objetos, para a tecnologia de aspectos? Mais especificamente, quais os reais ganhos que serão obtidos caso aspectos sejam usados para modularizar os interesses transversais de um sistema existente? Na literatura sobre aspectos, essas questões são investigadas em diversos trabalhos [6, 7], os quais normalmente comparam duas versões

de um mesmo sistema – orientada por objetos e orientada por aspectos – usando métricas clássicas para acoplamento, coesão, separação de interesses, etc [2]. Apesar de terem o mérito de abordar o problema de uma forma quantitativa – o que certamente favorece a realização de comparações – os resultados obtidos são específicos dos sistemas analisados. No máximo, eles podem ser extrapolados para sistemas que utilizem a mesma arquitetura e tecnologias, incluindo linguagens de programação e os mais diversos tipos de *frameworks*, para persistência, distribuição, interfaces gráficas, etc.

Em resumo, mantenedores de sistemas com uma arquitetura diferente daquela dos sistemas analisados nesses trabalhos irão continuar sem respostas claras e convincentes para as perguntas mencionadas. Evidentemente, não vale a pena para esses mantenedores migrar seus sistemas para a tecnologia de aspectos, para só então poder comparar a nova versão com a versão existente (e, eventualmente, chegar à conclusão de que o uso de aspectos não se mostrou tão vantajoso).

Assim, neste trabalho apresenta-se uma ferramenta, chamada ConcernMetrics, que: (1) permite que mantenedores de software criem um mapeamento lógico dos interesses transversais presentes em um sistema orientado por objetos (para isso reutiliza-se um *plug-in* para a plataforma Eclipse, chamado ConcernMapper); (2) realiza uma estimativa do impacto em métricas clássicas de separação de interesses caso o código relativo a tais interesses venha a ser fisicamente extraído para aspectos, usando AspectJ. Em sua atual versão, a ferramenta ConcernMetrics se baseia em duas métricas de separação de interesses: *Concern Diffusion over Operations* (CDO) e *Concern Diffusion over Components* (CDC). A ferramenta é capaz de calcular os valores dessas métricas para o código existente (orientado por objetos) e também seus novos valores caso aspectos sejam utilizados para modularizar os interesses transversais mapeados por meio do ConcernMapper. Como afirmado, o objetivo final é auxiliar mantenedores a responder às perguntas mencionadas anteriormente, sem que eles tenham que concretamente partir para implementação de aspectos.

Além de descrever a ferramenta ConcernMetrics, o trabalho apresenta também um estudo de caso, envolvendo um sistema para recuperação e análise de páginas Web, com cerca de catorze mil linhas de código, chamado JSpider[1]. Na versão orientada por objetos desse sistema, *logging* é um interesse transversal, estando sua implementação espalhada e entrelaçada por diversas classes. O sistema JSpider possui ainda uma versão na qual o código de *logging* foi modularizado por meio de aspectos. No trabalho, mostra-se que o ConcernMetrics foi capaz de mapear o código de *logging* presente na versão orientada por objetos (OO) desse sistema e estimar o valor das métricas CDO e CDC relativas à versão orientada por aspectos (OA). A fim de validar a estimativa produzida, comparou-se o valor inferido pelo ConcernMetrics com o valor real das métricas, calculado a partir da versão OA. Os resultados obtidos mostraram que a ferramenta proposta foi capaz de produzir estimativas bastante precisas, podendo de fato auxiliar mantenedores no processo de decisão sobre a migração ou não de seus sistemas para a tecnologia de aspectos.

O restante deste artigo está estruturado da seguinte forma. A Seção 2 apresenta a ferramenta ConcernMetrics, dando ênfase aos seus recursos para mapeamento de interesses transversais e para estimativa de métricas de separação de interesses. A Seção 3

[1] http://j-spider.sourceforge.net.

apresenta o estudo de caso com o sistema JSpider. A Seção 4 descreve trabalhos relacionados e a Seção 5 conclui o trabalho.

2 A Ferramenta ConcernMetrics

A ferramenta ConcernMetrics possui duas funcionalidades principais: mapeamento lógico de interesses transversais e estimativa de métricas de separação de interesses. Essas duas funcionalidades são descritas com mais detalhes a seguir.

Mapeamento de Interesses Transversais: Essa funcionalidade do ConcernMetrics é basicamente reutilizada do *plug-in* ConcernMapper, proposto por Robillard *et al.* para mapeamento de interesses transversais de forma lógica [9]. O ConcernMapper é uma ferramenta que permite organizar atributos e métodos correspondentes a interesses transversais em estruturas em forma de árvore, denominadas *Concern Maps*. Essa organização permite uma visão modularizada dos interesses transversais de um sistema sem que ele sofra qualquer tipo de alteração estrutural. Basicamente, usuários devem identificar atributos e métodos que correspondem a interesses transversais e arrastá-los para o mapa de *concerns*, o qual é uma estrutura visual, semelhante por exemplo ao *Package Explorer* da plataforma Eclipse. Em resumo, a ferramenta ConcernMetrics descrita neste trabalho utiliza como interface gráfica o *plug-in* ConcernMapper.

Conforme ilustrado na Figura 1, para iniciar o uso do ConcernMetrics, mantenedores devem criar uma nova entrada no mapa de *concerns* (por exemplo, *logging*). Em seguida, devem arrastar para esse nodo todos os métodos e atributos responsáveis pela sua implementação (por exemplo, os métodos `debug(Object)`, `debug(Object,Throwable)`, `error(Object)`, etc).

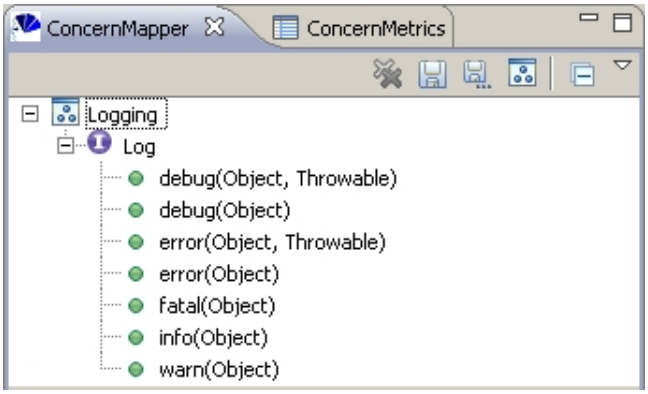

Figura 1. Mapeamento do interesse *logging* usando o ConcernMetrics

Estimativa de Métricas de Separação de Interesses: Essa é a principal contribuição da ferramenta implementada. Basicamente, uma vez mapeados os métodos e atributos que fazem parte de um interesse transversal, a ferramenta ConcernMetrics permite calcular duas métricas clássicas de separação de interesses [2, 6]:

- *Concern Diffusion over Operations* (CDO): Essa métrica conta o número de métodos M_C cujo propósito principal é contribuir para a implementação de um determinado interesse C (por exemplo, os sete métodos mostrados na Figura 1) e o

número de outros métodos – e de adendos, no caso de um sistema OA – que chamam os métodos em M_C.

- *Concern Diffusion over Components* (CDC): Essa métrica conta o número de componentes C_C cujo propósito principal é contribuir para a implementação de um determinado interesse C e o número de outros componentes que acessam os componentes em C_C. Nessa definição, componentes incluem classes (no caso de um sistema OO) e classes e aspectos (no caso de um sistema OA).

A ferramenta ConcernMetrics é capaz de calcular os valores de CDO e CDC para a versão existente do sistema (isto é, integralmente orientada por objetos). Mais importante, a partir do mapeamento lógico dos interesses transversais, o ConcernMetrics estima os novos valores de CDO e CDC, caso aspectos venham a ser usados para modularizar a implementação de tais interesses.

Implementação: Como afirmado, a interface gráfica da ferramenta ConcernMetrics consiste no *plug-in* ConcernMapper. Para calcular os valores da métricas CDO e CDC, o ConcernMetrics se baseia em um *framework* para manipulação, análise e reengenharia de bytecodes, chamado ASM[2]. Por meio dessa ferramenta, o *bytecode* do sistema analisado é inspecionado, instrução por instrução, procurando por chamadas dos métodos incluídos no mapa de interesses transversais (provido pelo ConcernMapper). O cálculo do valor de CDO e CDC para versão OO é direto, visto que as chamadas que devem ser consideradas estão fisicamente presentes no *bytecode* e, portanto, serão encontradas por essa rotina de inspeção.

Para estimativa dos valores de CDO e CDC para uma eventual versão OA, a decisão crucial que deve ser tomada pela ferramenta consiste em identificar, dentre as chamadas de métodos incluídas no mapa de interesses transversais, aquelas que podem ser implementadas no mesmo adendo. Para isso, o seguinte algoritmo de decisão é adotado:

1. Suponha as seguintes chamadas para um método m, incluído no mapa de interesses transversais: $t_1.m(arg_1)$ and $t_2.m(arg_2)$, onde t_i denota o alvo da chamada e arg_i denota seu argumento ($i = 1$ ou $i = 2$).

2. Essas duas chamadas podem ser agrupadas em um mesmo adendo, usando os recursos de quantificação de AspectJ, quando as seguintes condições são simultaneamente válidas:
   (a) t_1 e t_2 são variáveis da mesma classe (ou a mesma classe, no caso de métodos estáticos).
   (b) arg_1 e arg_2 representam o mesmo valor constante (seja ele, um inteiro, real, string, etc).

Exemplo: Suponha as chamadas a `start` e `log` presentes nas classes mostradas na Figura 2. As duas chamadas a `start` podem ser agrupadas em um mesmo adendo, caso aspectos venham a ser usados para modularizar a implementação de controle de transações nesse exemplo. O motivo é que os alvos de tais chamadas são campos do

[2]http://asm.objectweb.org.

mesmo tipo (`Transaction`) e seus argumentos incluem a mesma constante inteira (no caso, `1`). Por outro lado, as chamadas de `log` não poderão ser agrupadas em um mesmo adendo em uma eventual "aspectização" desse interesse, visto que seus argumentos são strings diferentes (no caso, `finished` e `panic`).

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

Figura 2. Exemplo de agrupamento de chamadas de métodos em adendos

3 Estudo de Caso: JSpider

JSpider é um robô que permite recuperar e validar páginas Web. O sistema possui uma versão orientada por objetos e uma versão orientada por aspectos, usada por Binkley *et al.* para descrever refatorações de interesses transversais [1]. Na versão OA, implementada em AspectJ, o código de *logging* foi devidamente modularizado em um aspecto.

A fim de validar a implementação da ferramenta ConcernMetrics, inicialmente os métodos responsáveis pela implementação de *logging* na versão OO do JSpider foram inseridos no mapa de interesses transversais (uma visão desse mapa foi mostrada na Figura 1). Em seguida, a ferramenta foi usada para calcular o valor das métricas CDO e CDC para a versão original do sistema (orientada por objetos, integralmente implementada em Java) 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 da versão OA do JSpider, cuja implementação foi independentemente realizada por Binkley et. al [1]. Os resultados obtidos são mostrados nas Tabelas 1 e 2.

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

Tabela 1. Métrica CDO para o interesse *logging* nas versões OO e OA do sistema JSpider, bem como estimativa para essas métricas produzida pela ferramenta ConcernMetrics (CM)

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

Tabela 2. Métrica CDC para o interesse *logging* nas versões OO e OA do sistema JSpider, bem como estimativa para essas métricas produzida pela ferramenta ConcernMetrics (CM)

Análise dos Resultados: Como pode ser observado na Tabela 1, a estimativa para a métrica CDO produzida pela ferramenta ConcernMetrics coincidiu em três dos sete métodos avaliados: métodos de número 1, 5 e 7 da Tabela 1. Para os métodos onde houve uma diferença entre a estimativa do ConcernMetrics e o valor medido da métrica CDO na versão OA, realizou-se uma inspeção manual no código da versão OA para tentar explicar a diferença. Os resultados dessa inspeção são reportados abaixo:

- O valor de CDO do método `debug(Object,Throwable)` calculado pelo ConcernMetrics é menor em uma unidade do que o valor medido na versão OA. Essa diferença aconteceu pelo fato de uma chamada extra a esse método ter sido adicionada ao código da versão OA (chamada essa que não existe na versão OO 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 um chamada ao método `error(Object)` e três chamadas ao método `error(Object,Throwable)` terem sido eliminadas do código da versão OA.

- O valor de CDO do método `info(Object)` calculado pelo ConcernMetrics é maior em uma unidade do que o valor medido na versão OA. Essa diferença também é explicada pelo fato de uma chamada a esse método ter sido removida do código OA.

Ou seja, a estimativa produzida pela ferramenta ConcernMetrics foi sempre aquela esperada em uma ferramenta que trabalha analisando apenas o código OO do sistema. Na verdade, constatou-se que as refatorações realizadas no JSpider para "aspectização" do código de *logging* 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 mantenedores que realizaram essa tarefa.

Conforme mostrado na Tabela 2, em relação à métrica CDC, pode-se observar que não houve nenhuma diferença entre a estimativa calculada pelo ConcernMetrics e o

valor efetivamente medido na versão OA do JSpider. Para todos os métodos avaliados, CDC=2. O motivo é que essa métrica conta o número de componentes que implementam a funcionalidade de *logging* – no caso do JSpider, apenas a classe `Log` – e o número de componentes que chamam métodos dessa classe – no caso do JSpiderAO, apenas um aspecto, o qual passa a confinar todo o código de *logging* do sistema.

Assim, conclui-se que a ferramenta ConcernMetrics estimou corretamente os novos valores das métricas CDC e CDO caso refatorações para extração de aspecto venham a ser aplicadas para modularizar o interesse de *logging* no sistema JSpider.

4 Trabalhos Relacionados

Proposto por Eaddy *et al.*, o ConcernTagger é uma outra extensão do *plug-in* ConcernMapper que permite um mapeamento lógico entre interesses (possivelmente, transversais) e elementos sintáticos de um programa Java [3]. A partir desse mapeamento, o sistema calcula algumas métricas, incluindo CDC e CDO. Portanto, a ferramenta possui alguma similaridade com o ConcernMetrics. No entanto, existem duas diferenças fundamentais entre as abordagens adotadas pelos dois sistemas:

- Em sua atual versão, o ConcernTagger requer que usuários identifiquem manualmente cada elemento do programa base onde existe implementação de um determinado interesse transversal. Por exemplo, para calcular a métrica CDO tal como realizado no Estudo de Caso descrito na Seção 3, usuários devem manualmente inserir no mapa de interesses transversais não apenas os métodos da classe `Log`, mas também todos os demais métodos que fazem chamadas a métodos dessa classe. Ou seja, pode-se afirmar que o cálculo da métrica ocorre de forma semi-automática, pois cabe aos usuários manualmente indicar os métodos que usam `Log`. Já no ConcernMetrics, a localização dos métodos usuários da classe `Log` ocorre de forma totalmente automática.

- A partir da identificação manual de métodos que fazem uso de um interesse transversal, o ConcernTagger apenas realiza uma contagem desses métodos, obtendo assim o valor da métrica CDO para o código OO. Ou seja, a ferramenta não implementa o processo de decisão descrito na Seção 2, o qual é fundamental para estimar o número de adendos necessários para implementar os interesses manualmente identificados. Em resumo, diferentemente do ConcernMetrics, o ConcernTagger não realiza nenhuma estimativa dos novos valores de CDC e CDO, caso aspectos venham a ser utilizados para modularizar a implementação dos interesses mapeados.

Um dos primeiros trabalhos sobre avaliação empírica de sistemas orientados por aspectos foi realizado por Garcia *et al.*, inicialmente no contexto de padrões de projeto [6]. Basicamente, eles comparam nesse primeiro trabalho implementações tradicionais de diversos padrões de projeto com implementações em AspectJ dos mesmos padrões. Para realizar essas comparações, eles se baseiam em métricas de separação de interesses (incluindo CDO e CDC), acoplamento, coesão e tamanho. Posteriormente, eles reproduziram esse primeiro estudo em outros contextos, incluindo tratamento de exceções [5] e linhas de produtos de software [4]. No entanto, nesses trabalhos, assume-se sempre a existência prévia de uma versão OA, a qual é comparada com a versão original do sistema.

5 Conclusões

Neste trabalho, foi apresentada uma ferramenta que permite, a partir do próprio código orientado por objetos, estimar os valores das métricas CDO e CDC caso aspectos venham a ser utilizados para modularizar interesses transversais de um sistema existente. O estudo de caso apresentado mostrou que a ferramenta estimou corretamente os valores dessas métricas e que, portanto, ela possui potencial para auxiliar mantenedores de software no processo de decisão sobre o eventual uso de aspectos em seus sistemas. Como trabalho futuro, pretende-se incorporar novas métricas na ferramenta ConcernMetrics, incluindo métricas de acoplamento, coesão e quantificação. Pretende-se também realizar novos estudos de caso.

Agradecimentos: Este trabalho foi apoiado pela FAPEMIG e CNPq.

Referências

[1] David Binkley, Mariano Ceccato, Mark Harman, Filippo Ricca, and Paolo Tonella. Tool-supported refactoring of existing object-oriented code into aspects. *IEEE Transactions Software Engineering*, 32(9):698–717, 2006.

[2] Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[3] Marc Eaddy, Thomas Zimmermann, Kaitin D. Sherwood, Vibhav Garg, Gail C. Murphy, Nachiappan Nagappan, and Alfred V. Aho. Do crosscutting concerns cause defects? *IEEE Transaction Software Engineering*, 34(4):497–515, 2008.

[4] Eduardo Figueiredo, Nélio Cacho, Cláudio Sant'Anna, Mario Monteiro, Uirá Kulesza, Alessandro Garcia, Sérgio Soares, Fabiano Cutigi Ferrari, Safoora Shakil Khan, Fernando Castor Filho, and Francisco Dantas. Evolving software product lines with aspects: an empirical study on design stability. In *30th International Conference on Software Engineering (ICSE)*, pages 261–270, 2008.

[5] Fernando Castor Filho, Nelio Cacho, Eduardo Figueiredo, Raquel Maranhao, Alessandro Garcia, and Cecilia Rubira. Exceptions and aspects: the devil is in the details. In *14th International Symposium on Foundations of Software Engineering (FSE)*, pages 152–162, 2006.

[6] Alessandro Garcia, Cláudio Sant'Anna, Eduardo Figueiredo, Uirá Kulesza, Carlos Lucena, and Arndt von Staa. Modularizing design patterns with aspects: a quantitative study. In *4th International Conference on Aspect-Oriented Software Development (AOSD)*, pages 3–14, 2005.

[7] Roberto Lopez-Herrejon and Sven Apel. Measuring and characterizing crosscutting in aspect-based programs: Basic metrics and case studies. In *International Conference on Fundamental Approaches to Software Engineering (FASE)*, March 2007.

[8] Miguel P. Monteiro and João M. Fernandes. Towards a catalogue of refactorings and code smells for AspectJ. *Transactions on Aspect-Oriented Software Development*, 3880:214–258, 2006.

[9] Martin P. Robillard and Frederic Weigand-Warr. ConcernMapper: Simple view-based separation of scattered concerns. In *OOPSLA Eclipse Technology Exchange Workshop (ETX)*, pages 65–69, 2005.