6
O framework de avaliação quantitativa

"Not everything that counts can be counted; and not everything that can be counted counts." (Albert Einstein)

O desenvolvimento de software orientado a aspectos (DSOA) está recebendo cada vez mais atenção no setor e nos ambientes de pesquisa [110, 149, 156, 230]. O DSOA é um paradigma promissor para promover uma melhor separação de concerns, levando à produção de sistemas de software mais fáceis de manter e reutilizar. Todavia, como o paradigma orientado a aspectos ainda está dando os primeiros passos, é muito difícil determinar o que é uma boa decisão de implementação e projeto para o DSOA. Há pouco consenso de que os crosscutting concerns clássicos e transparentes devem ser modularizados dentro de aspectos, como auditoria [12], rastreamento [12] e tratamento de exceções [161]. Não há um princípio para auxiliar no projeto de outros crosscutting concerns importantes e mais dependentes do domínio, como os concerns do agente. É difícil entender quando usar aspectos como soluções de projeto e arquitetura. Como conseqüência, os aspectos estão atualmente sendo aplicados de uma maneira *ad hoc*.

A utilidade de novos paradigmas de desenvolvimento e das práticas de projetos associadas pode ser avaliada por meio de estudos empíricos. As métricas de software [68, 118] são usadas em estudos empíricos como indicadores dos pontos fortes e fracos da abordagem estudada. Muitas dessas métricas foram propostas [46, 68], usadas e, às vezes, empiricamente validadas [15, 158]. Elas incluem várias linhas de código [68], métricas de Chidamber e Kemerer [46] etc. As métricas de software normalmente estão associadas ao modelo de qualidade que define seu escopo. Muitas empresas constroem seus próprios modelos de qualidade com base nas métricas do produto [23, 168]. Além disso, alguns ambientes de desenvolvimento incorporaram suporte a métricas, como Together [25].

Contudo, as métricas disponíveis na literatura não se dedicam ao DSOA. Como resultado, muitos estudos empíricos que envolvem a aplicação da tecnologia de

aspectos são baseados em uma avaliação qualitativa [82, 115, 138, 141, 226]. Esses estudos não dependem de um modelo de qualidade estruturado e de princípios de engenharia de software amplamente aceitos. Normalmente, para investigar a qualidade das soluções orientadas a aspectos, esses estudos usavam conceitos ainda não bem compreendidos, como *conectividade* e *composabilidade* [115]. Apesar de o objetivo desses conceitos ser capturar várias facetas de manutenção e reutilização de software, suas definições são amplamente vagas e bem fundamentadas na intuição dos proponentes.

Nesse contexto, este capítulo envolve três perguntas da pesquisa:

(i) Qual é a diferença entre a orientação a aspectos e a orientação a objetos em termos de atributos de software bem conhecidos, como acoplamento e coesão?

(ii) Como avaliar a reusabilidade e a manutenibilidade de softwares orientados a aspectos?

(iii) Como comparar quantitativamente a reusabilidade e a manutenibilidade de artefatos orientados a aspectos e artefatos orientados a objetos?

Este capítulo discute o impacto dos aspectos em importantes atributos de software, como acoplamento e coesão. Além disso, propõe um framework de avaliação quantitativa para DSOA que trata das questões supracitadas. O framework é composto por dois componentes: um conjunto de métricas e um modelo de qualidade. Esses componentes baseiam-se em princípios conhecidos e métricas existentes a fim de evitar a reinvenção de soluções bem testadas. Nosso ponto de vista é que os graus de manutenibilidade e reusabilidade dos sistemas orientados a aspectos devem ser avaliados em termos de princípios da engenharia de software conhecidos e usando métricas testadas.

O framework de avaliação foi usado no contexto de um estudo empírico semicontrolado a fim de avaliar a abordagem orientada a aspectos proposta para sistemas multiagentes (Capítulo 8). Contudo, é preciso ressaltar que o framework de avaliação é independente dos concerns do agente (Capítulo 3). Pode ser aplicado a

qualquer domínio de aplicação usando aspectos. De fato, o framework foi usado em dois outros contextos empíricos que envolvem diferentes domínios e concerns (Seção 6.5) [98, 264, 265]. Com base nessa experiência, as vantagens e as desvantagens dos componentes do framework são discutidas neste capítulo.

A descrição do framework de avaliação foi publicada em um capítulo de livro [98] e em um trabalho apresentado em um simpósio [210]. Esse último enfatiza a descrição dos componentes do framework, e o primeiro, o uso do framework no contexto do estudo empírico (Capítulo 8). O restante deste capítulo está organizado da seguinte forma. A Seção 6.1 introduz os requisitos para o conjunto de métricas proposto. A Seção 6.2 apresenta a estrutura do framework. A Seção 6.3 descreve o modelo de qualidade, e a Seção 6.4 introduz o conjunto de métricas proposto. A Seção 6.5 apresenta a avaliação empírica do framework de avaliação e uma análise de sua utilidade. A Seção 6.6 discute o framework proposto e os trabalhos relacionados em termos da reusabilidade. A Seção 6.7 apresenta algumas conclusões e direções para trabalhos futuros.

6.1
Aspectos: requisitos de medida

O propósito de usar aspectos é alcançar uma melhor separação de concerns (SoC). Entretanto, isso normalmente afeta outros atributos de software, como acoplamento, coesão e tamanho. Os aspectos são eficazes na modularização de crosscutting concerns, minimização de código e, como conseqüência, redução do tamanho do sistema. Contudo, o uso inapropriado de aspectos tem um efeito negativo nesses atributos de software e aumenta a complexidade do sistema. As métricas de software são a forma mais eficaz de fornecer comprovações empíricas que podem melhorar a compreensão das diferentes dimensões da complexidade do software [31]. Elas avaliam o uso de abstrações durante o desenvolvimento do software em termos de atributos, como acoplamento e coesão. As métricas são mais eficazes quando estão associadas ao modelo de qualidade [68] de forma que os engenheiros de software possam entender e interpretar os significados dos dados coletados.

A literatura não contém muitos conjuntos de métricas tradicionais e outros sistemas OO. Muitas métricas existentes não podem ser aplicadas diretamente em softwares orientados a aspectos [252, 254], uma vez que o DSOA introduz novas abstrações na engenharia de software (Seção 2.2.2). O DSOA não tem efeitos só na separação dos concerns do sistema. Os componentes do sistema (classes e aspectos) são compostos de forma diferente e, como conseqüência, as abstrações orientadas a aspectos incorporam diferentes dimensões de acoplamento e coesão [252, 254]. Por exemplo, a Figura 76 ilustra maneiras de combinar classes e aspectos, que são as fontes potenciais de acoplamento em um sistema orientado a aspectos. Ademais, o DSOA tem um impacto direto no tamanho do sistema, porque o uso apropriado de aspectos pode minimizar a replicação de código.

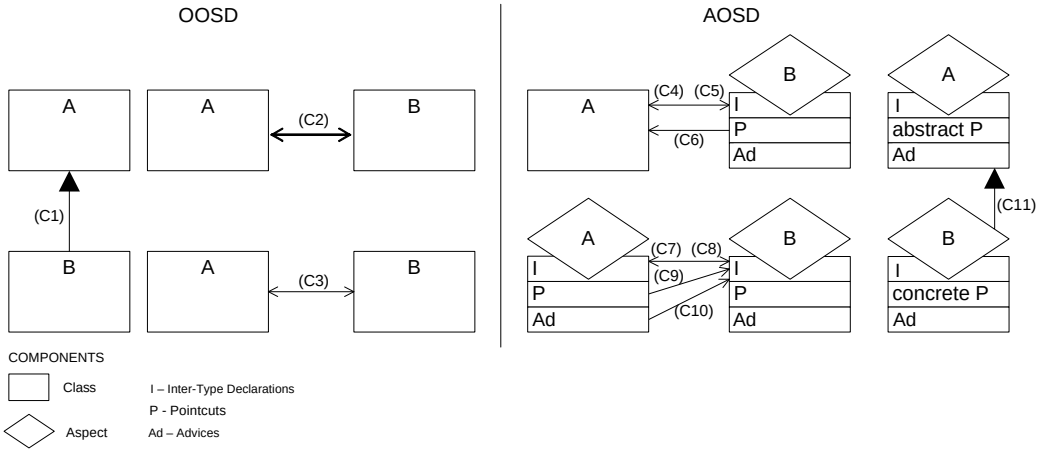

Figura 76. Dimensões de acoplamento em um DSOA.

Dessa forma, a definição da métrica adequada para o software orientado a aspectos deve satisfazer a estes requisitos:

Requisito 1 – medir atributos de software conhecidos, como a separação de concerns, acoplamento, coesão e tamanho.

Requisito 2 – depender o máximo possível das métricas tradicionais e da extensão das métricas OO para o DSOA, uma vez que as abstrações orientadas a aspectos estendem o conjunto de abstrações OO.

Requisito 3 – capturar diferentes dimensões da separação de concerns, tamanho, acoplamento e coesão no desenvolvimento de software orientado a aspectos.

Requisito 4 – oferecer suporte à identificação das vantagens e desvantagens do uso de aspectos em um projeto de software em comparação a uma solução orientada a objetos no mesmo problema.

6.2
A estrutura do framework

A medida de um atributo interno particular, como o acoplamento, é útil se estiver relacionada a uma medida de algum atributo externo do objeto de estudo (por exemplo, reusabilidade). Na engenharia de software, as medidas de atributos de produtos internos são conceitos artificiais e, em si, não têm qualquer significado [31]. Nesse contexto, desenvolvemos um framework para capturar a compreensão de atributos como a separação de concerns, acoplamento, coesão e tamanho em termos de sua utilidade como indicadores das qualidades de manutenibilidade e reusabilidade. De fato, o objetivo do framework de avaliação é oferecer suporte à avaliação da reusabilidade e manutenibilidade de sistemas orientados a aspectos.

Os componentes do framework ajudam os engenheiros de software a organizar o processo de avaliação e ajudam na coleta e interpretação de dados (Figura 77). Os componentes básicos do framework são: (i) um modelo de qualidade (Seção 6.3) e (ii) um conjunto de métricas (Seção 6.4). O modelo de qualidade estabelece os relacionamentos entre os atributos externos, os atributos internos e as métricas. O framework requer alguns artefatos como entradas para o processo de medição. Primeiro, os documentos do projeto e o código do sistema para o uso das métricas. Além disso, o framework de avaliação requer uma descrição dos concerns do sistema para orientar sua identificação ao usar as métricas de separação de concerns.

6.3
O modelo de qualidade

O modelo de qualidade define uma terminologia e esclarece os relacionamentos entre a reusabilidade e a manutenibilidade e o conjunto das métricas. É uma ferramenta útil para orientar os engenheiros de software no processo de avaliação e medição. A definição do modelo de qualidade baseia-se em: (i) uma análise extensiva

de um conjunto de modelos de qualidade existentes [68, 83], (ii) definições clássicas de atributos de qualidade [123, 170, 228, 267] e (iii) teorias de projeto tradicionais, como a teoria de Parnas [193] e a teoria de Dijkstra [60], normalmente aceitas entre os pesquisadores e os profissionais e (iv) os atributos de software impactados pelas abstrações orientadas a aspectos (Seção 6.1) [252, 254].

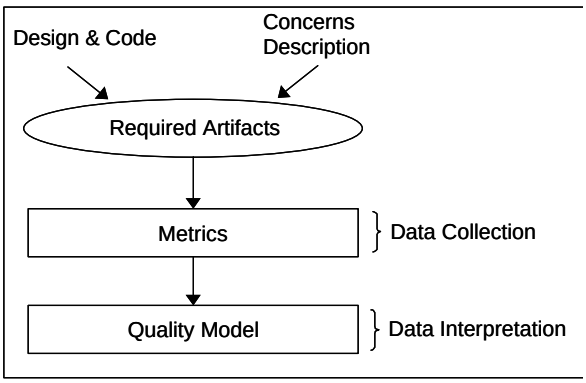

Figura 77. O framework de avaliação.

Os modelos de qualidade são construídos usando um método do tipo árvore uma vez que a qualidade é um conjunto de muitas outras qualidades [68]. A noção da qualidade de software é normalmente capturada em um modelo que descreve outras qualidades intermediárias, que chamamos de fatores. O modelo de qualidade é composto por três elementos diferentes: (i) qualidades, (ii) fatores e (iii) atributos internos. Além disso, o modelo de qualidade conecta os atributos internos às métricas. As *qualidades* são os atributos que desejamos observar primariamente no sistema do software (reusabilidade e manutenibilidade).

Os *fatores* são os atributos de qualidade secundários que influenciam as qualidades primárias. Os *atributos* estão relacionados às propriedades internas dos sistemas de software. Esses atributos estão relacionados a princípios da engenharia de software bem definidos, que, por sua vez, são essenciais para alcançar as qualidades e seus respectivos fatores [68]. A Figura 78 apresenta os elementos do modelo de qualidade. A parte superior (lado esquerdo) contém os fatores e as qualidades de alto nível que desejamos quantificar. Os atributos internos são mais fáceis de medir do que os fatores e as qualidades e, assim, as métricas reais estão conectadas a esses

atributos [68]. Há diferentes métricas definidas para oferecer suporte à medição dos atributos internos (Figura 78). As métricas são definidas na Seção 6.4. As subseções a seguir descrevem os elementos do modelo de qualidade proposto.

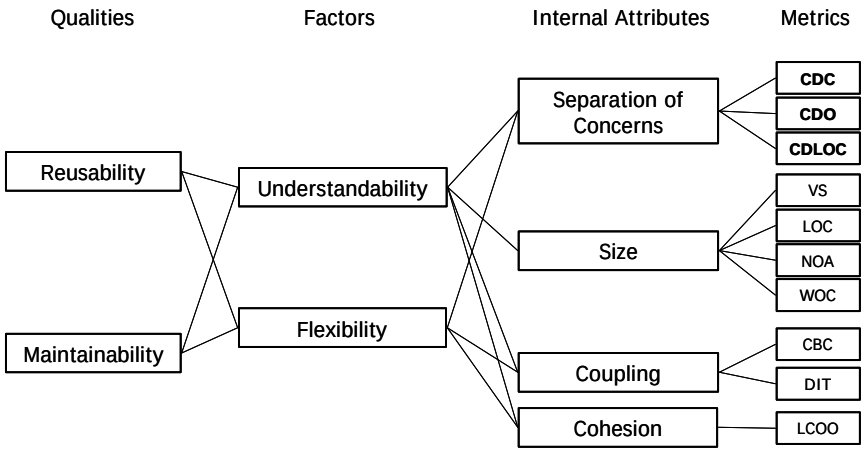

Figura 78. O modelo de qualidade.

6.3.1 Qualidades e fatores

Conforme mencionado anteriormente, a manutenibilidade e a reusabilidade são o foco do framework de avaliação. A reusabilidade é a capacidade dos elementos do software de oferecer suporte à construção de diferentes elementos no mesmo sistema do software ou através de sistemas diferentes [170]. O framework proposto está relacionado à avaliação da reusabilidade do código e do projeto orientado a aspectos. A manutenção é a atividade que modifica o sistema de um software depois de uma transferência inicial. A manutenibilidade do software pode ser definida como a facilidade com que os seus componentes podem ser modificados. As atividades de manutenção são divididas em quatro categorias [68, 228]: manutenção corretiva, manutenção aperfeiçoadora, manutenção adaptativa e evolução. Como o objetivo principal do DSOA é melhorar a evolução dos sistemas OO (Seção 2.2.2), o framework se concentra nos aspectos da evolução de sistemas orientados a aspectos.

O modelo de qualidade enfatiza que os fatores similares são úteis para a promoção da manutenibilidade e da reusabilidade. Essa similaridade está relacionada

ao fato de a manutenção e a reutilização incorporarem tarefas cognitivas comuns. A flexibilidade e a compreensibilidade são fatores centrais para a promoção da reusabilidade e manutenibilidade [68, 170, 193, 228]. Os dois tipos de atividades requerem abstrações de software para oferecer suporte à manutenibilidade e flexibilidade. A compreensibilidade indica o nível de dificuldade e compreensão do código e do projeto de um sistema [193]. A flexibilidade indica o nível de dificuldade ao efetuar alterações drásticas em um componente de um sistema sem a necessidade de alterar outros [193]. Um sistema compreensível aumenta sua própria manutenibilidade e reusabilidade, uma vez que muitas atividades de manutenção e reutilização requerem que os engenheiros de software tentem primeiro entender os componentes do software antes de fazer outras modificações ou extensões no sistema. Ademais, um sistema de software precisa ser flexível o suficiente para minimizar o impacto das alterações em diferentes componentes do sistema.

No modelo proposto, o fator de compreensibilidade está relacionado aos seguintes atributos internos: (i) tamanho, (ii) acoplamento, (iii) coesão e (iv) separação de concerns. O acoplamento e a coesão afetam a compreensibilidade porque um componente do sistema não pode ser entendido sem referências a outros componentes com os quais está relacionado. O tamanho do projeto e o código podem indicar a quantidade de esforço necessário para entender os componentes do software. O critério de separação de concerns é um indicador de compreensibilidade porque quanto mais fácil for localizar os concerns do sistema, mais fácil será compreendê-los. O fator de flexibilidade é influenciado pelos seguintes atributos internos: (i) acoplamento, (ii) coesão e (iii) separação de concerns. Alta coesão, baixo acoplamento e alta separação de concerns são características desejáveis, porque significam que um componente representa uma única parte do sistema e que os componentes do sistema são independentes ou quase independentes. Além disso, os concerns do sistema não estão entrelaçados e espalhados. Se for necessário adicionar, remover ou reutilizar funcionalidades, elas estão localizadas em um único componente, e as atividades de manutenção e reutilização são flexivelmente restritas a esse componente isolado. Observe que o fator de flexibilidade não é influenciado pelo atributo de tamanho uma vez que o número de componentes ou linhas de código

não tem qualquer efeito na propagação de alterações nos diferentes componentes do sistema.

6.3.2 Atributos Internos

Cada atributo interno se conecta a um conjunto de métricas. Nos parágrafos a seguir, declaramos os relacionamentos entre os atributos internos, as métricas e as qualidades e fatores em termos de cada atributo interno.

Separação de Concerns (SoC). A separação de concerns se refere à capacidade de encapsular essas partes do software que são relevantes para um determinado concern [234]. Como conseqüência, quanto mais diretamente o concern mapear para o projeto e os elementos de código, mais fácil será para os engenheiros de software entendê-lo. Quanto mais diretamente um concern mapear para o projeto e os elementos do código, menos componentes serão alterados durante as atividades de manutenção ou menos componentes são necessários para entender e estender o concern durante as atividades de reutilização. As métricas Difusão de Concerns em Componentes, Difusão de Concerns em Operações e Difusão de Concerns em LOC (Seção 6.4) medem o grau com que um único concern no sistema mapeia os elementos do software no código e projeto do software.

Acoplamento. O acoplamento entre componentes do sistema foi durante muito tempo considerado o maior causador da complexidade do sistema. Ele é uma indicação da força das interconexões entre os componentes em um sistema. Os sistemas altamente acoplados têm fortes interconexões, com as unidades do programa dependentes entre si [228]. A compreensão de um componente envolve a compreensão dos componentes aos quais ele está acoplado. Assim, quanto maior o número de acoplamentos de um componente, mais difícil é entender o sistema. Quanto maior o número de acoplamentos, maior a sensibilidade às mudanças em outras partes do projeto e, portanto, mais difícil é a manutenção. O acoplamento excessivo é prejudicial ao projeto modular e impede a reutilização. Quanto mais independente for um componente, mais fácil é reutilizá-lo em outra aplicação. As

métricas Acoplamento entre Componentes e Profundidade da Árvore de Herança (Seção 6.4) medem o acoplamento a partir de diferentes pontos de vistas.

Coesão. A coesão de um componente é uma medida da proximidade de relacionamento entre seus componentes internos [228]. Ela caracteriza o *acoplamento interno* de um componente, como uma distinção do acoplamento *entre* os componentes. Quanto mais alto for o grau com o qual as diferentes ações realizadas por um componente contribuem para funções distintas, mais difícil é reutilizar e manter o componente ou uma de suas funcionalidades. A coesão é uma qualidade desejável devido à localização dos atributos comportamentais de um subsistema. A compreensão de um componente coeso pode ser amplamente limitada à compreensão do próprio componente. A métrica Falta de Coesão em Operações (Seção 6.4) detecta o grau com o qual um componente implementa uma única função lógica.

Tamanho. As métricas de tamanho estão relacionadas a diferentes aspectos do tamanho do sistema. Em geral, quanto maior o tamanho, mais difícil é entender o sistema. Há diferentes formas de medir o tamanho do sistema. Por exemplo, o tamanho do sistema pode ser medido em termos das linhas de código. Quanto maior o número de linhas de código, mais difícil é entender o sistema. Quanto mais linhas de código, mais difícil é encontrar as linhas que precisam ser alteradas durante as atividades de evolução ou de entender a implementação das funcionalidades necessárias durante as atividades de reutilização. As métricas Tamanho do Vocabulário, Linhas de Código, Número de Atributos, Operações Consideradas por Componente (Seção 6.4) medem o tamanho do sistema a partir de diferentes pontos de vista.

6.4
O conjunto de métricas

O conjunto de métricas proposto captura as informações relativas ao projeto e ao código em termos dos atributos fundamentais do software, como separação de concerns, acoplamento, coesão e tamanho. Esse conjunto reutiliza e refina as métricas existentes para oferecer suporte à medição dos atributos internos [46, 68]. Inclui

métricas clássicas, como a métrica LOC e refina métricas orientadas a objetos, como as métricas de Chidamber e Kemerer (CK) [46]. A definição dessas métricas foram retrabalhadas para refletir as novas abstrações introduzidas pelos aspectos. Os critérios para a seleção dessas métricas baseavam-se em demandas práticas e teóricas. Por exemplo, as métricas de CK baseiam-se em uma teoria de medição confiável e foram amplamente usadas e empiricamente validadas no contexto da orientação a objetos [15].

Além disso, o conjunto inclui métricas originais para a separação de concerns. Essas métricas capturam diferentes dimensões de separação de concerns. Elas detectam o grau de entrelaçamento do código e do espalhamento de concern (Seção 2.2.1). O conjunto é composto por cinco métricas de projeto e cinco métricas de código. Nas subseções a seguir, essas métricas são agrupadas de acordo com os atributos que medem: (i) separação de concerns (SoC) (ii) acoplamento, (iii) coesão e (iv) tamanho. A descrição de cada métrica enfatiza como ela satisfaz os requisitos de medição (Seção 6.1). Também é descrita a relevância da métrica para a avaliação da reusabilidade e manutenibilidade.

As métricas baseiam-se em uma série de premissas a fim de oferecer suporte à comparação entre o projeto orientado a objetos e o projeto orientado a aspectos. Primeiro, os aspectos e as classes são contados como unidades similares nas métricas de tamanho, uma vez que os dois são unidades de modularidade. Segundo, os advices são considerados similares aos métodos, uma vez que os dois são operações. A única diferença é que os advices são definidos dentro dos aspectos. Terceiro, os parâmetros de advices são considerados exatamente como os parâmetros de métodos.

6.4.1
Métricas de separação de concerns

O conjunto é composto por três métricas para a separação de concerns: (i) Difusão de Concerns em Componentes (CDC), (ii) Difusão de Concerns em Operações (CDO) e (iii) Difusão de Concerns em Linhas de Código (CDLOC). O termos “difusão” representa os problemas de entrelaçamento do código e espalhamento de concern (Seção 2.2.2).

Difusão de Concerns em Componentes (CDC)

Definição: A DDC é uma métrica de projeto que conta o número de "componentes primários" cujo propósito principal é contribuir para a implementação de um concern. Ademais, ela conta o número de componentes que acessam os componentes primários ao se referirem a eles em declarações de atributos, parâmetros formais, tipos de retorno, declarações levantadas e variáveis locais ou chamando seus métodos.

Relevância: Essa métrica mede o grau com o qual um concern de um único sistema mapeia os componentes no projeto do software. Quanto mais diretamente um concern mapear para os componentes, mais fácil será entender o projeto. Quanto mais diretamente um concern mapear para componentes, menos componentes serão alterados durante as atividades de manutenibilidade ou menos componentes serão entendidos e estendidos durante as atividades de reutilização.

Difusão de Concerns em Operações (CDO)

Definição: A CDO conta o número de "operações primárias", ou seja, as operações cujo objetivo principal é contribuir para a implementação de um concern. Ela também conta o número de métodos e advices que acessam as operações primárias, chamando seus métodos ou usando-os em parâmetros formais, tipos de retorno, declarações levantadas e variáveis locais. Os construtores também são contados como operações.
Relevância: A CDO captura o grau de entrelaçamento e espalhamento no sistema, uma vez que conta o número de operações afetadas por um determinado concern. Quanto mais operações forem afetadas pelo concern, mais difícil é entender o projeto. Quanto mais operações forem afetadas pelo concern, mais afetado será o concern. Portanto, é mais difícil reutilizar e mais alterações serão feitas durante as atividades de manutenção.

Difusão de Concerns em LOC (CDLOC)

Definição: A CDLOC conta o número de pontos de transição para cada concern pelas linhas de código. O uso dessa métrica requer um processo de sombreamento que divida o código em áreas sombreadas e áreas não-sombreadas [83]. As áreas sombreadas são linhas de códigos que implementam um dado concern. A Figura 79

apresenta um exemplo de sombreamento de código para o concern de Interação na classe PAgent do sistema Portalware (Seção 4.2). Os pontos de transição são pontos no código em que há uma transição de uma área não-sombreada para uma área sombreada e vice-versa. A intuição por trás disso é que são pontos no texto do programa onde há uma “chave de concern”. Para cada concern, o texto do programa é analisado linha por linha a fim de contar os pontos de transição.

```
public class PAgent {

    private String agentName;
   ...
   ...
    protected Interaction theInteraction;
    protected Autonomy    theAutonomy;
    protected Adaptation  theAdaptation;

    public PAgent(String aName, Vector pl) {
        init();
         agentName = aName;
        theInteraction = new Interaction(this);
        theAutonomy = new Autonomy(this);
        theAdaptation = new Adaptation(this);
        planList = pl;
        System.out.println(" Name == " + agentName);
   }
   ...
   ...
/* Interface  for  Interaction */
   public void receiveMsg(Message msg)
   {
       theAutonomy.makeDecision(msg);
       theAdaptation.adaptBeliefs(msg);
   }

   public void outcomingMsg(Message msg)
   {
       theInteraction.outcomingMsg(msg);
   }
}
```

Figura 79. Um exemplo de sombreamento de código.

O processo de sombreamento segue estas diretrizes:

a) Classes cujo único propósito é auxiliar a implementação de cada concern são tratadas de forma especial: a declaração e seus métodos são sombreados como um bloco único. Chamadas de método a instâncias dessas classes são sombreadas.

b) Aspectos cujo único propósito é auxiliar a implementação de cada concern são tratados de forma especial: a declaração e seus métodos e advices são sombreados como um bloco único. Chamadas de método a instâncias desses aspectos são sombreadas.

c) Métodos cujo único objetivo é a implementação de um concern são sombreados; chamadas àqueles métodos também são sombreadas. Observe que esses métodos não fazem parte das classes ou aspectos cujo único propósito é implementar um concern.
d) As declarações das variáveis usadas para manter os elementos específicos aos concern também são sombreadas, como um bloco; o uso dessas variáveis também é sombreado.
e) As assinaturas cujos parâmetros contêm referências a objetos que implementam um concern são sombreadas. Observe que esses métodos não fazem parte da definição desse concern.
f) As declarações de aspectos que incluem referências a outros aspectos (encapsulando outros concerns) são sombreadas. Esse é o caso da declaração “declare precedence” [12] (Apêndice I).
g) A aplicação das diretrizes a-f pode gerar dois ou mais blocos sombreados para o mesmo concern, que estão em uma seqüência; em seguida, esse conjunto de blocos deve ser unificado como um único bloco.
h) Se dois blocos do mesmo concern não estiverem em seqüência, mas poderiam estar, devemos contá-los como um bloco único.

Relevância: Quando mais alta a CDLOC, mais intermisturado estará o código do concern com a implementação de outros concerns nos componentes; quanto mais baixa a CDLOC, mais localizável será o código do concern. Quanto mais entrelaçado e espalhado for um concern: (i) mais difícil será entender o código, (ii) mais linhas de código serão afetadas durante as atividades de evolução e (iii) mais difícil será reutilizar o código. As medições da CDLOC são relativas aos concerns que foram buscados e o método para contar os pontos de transição. Qualquer pequena variação desses dois fatores resulta em alterações drásticas das medidas.

6.4.2
Métricas de acoplamento

O conjunto é composto de duas métricas para o acoplamento: (i) Acoplamento entre Componentes (CBC) e (ii) Profundidade de Árvore de Herança (DIT).

Acoplamento entre Componentes (CBC)

Definição: O CBC é definido para um componente (classe ou aspecto) como um registro dos outros componentes aos quais está acoplado. Ele conta o número de classes usadas em declarações de atributos, ou seja, captura os acoplamentos C2 e C3 descritos na Figura 76. Também conta o número de componentes declarados em parâmetros formais, tipos de retorno, declarações levantadas e variáveis locais e classes e aspectos a partir dos quais as seleções de métodos e atributos são feitas. Se um componente A for acoplado a um componente B em um número arbitrário de formas, o CBC conta somente uma vez. Essa métrica é uma extensão da métrica de CK para o acoplamento entre objetos (CBO). A fim de definir o CBC, alteramos a definição de CBO para lidar com as novas dimensões em DSOA: acessos a atributos e métodos de aspectos definidos por inter-type declarations (acoplamentos C4, C5, C7, C8, C10), e os relacionamentos entre aspectos e classes e outros aspectos definidos em pointcuts (acoplamentos C6, C9). Essa métrica incorpora nove dimensões de acoplamento descritas na Figura 76 (de C2 a C10).

Relevância: A compreensão de um componente envolve a compreensão dos componentes aos quais ele está acoplado. Quanto maior o acoplamento do componente, mais difícil é entender o sistema. A fim de melhorar a modularidade e promover a encapsulação, o acoplamento do sistema deve ser mantido no mínimo. Quanto maior o acoplamento, maior a sensibilidade a mudanças em outras partes do projeto e, portanto, mais difícil é a manutenção. O acoplamento excessivo entre componentes é prejudicial ao projeto modular e impede a reutilização. Quanto mais independente for um componente, mais fácil é reutilizá-lo em outra aplicação.

Profundidade de Árvore de Herança (DIT)

Definição: A DIT é definida como o comprimento máximo de um nó até a raiz da árvore. Ela conta a distância inferior em que uma classe ou aspecto é declarado na hierarquia de herança. É uma extensão de uma métrica de CK com o mesmo nome que considera a herança entre aspectos. Essa métrica incorpora as dimensões de acoplamento C1 e C11 ilustradas na Figura 76.

Relevância: Quanto mais profundo for um componente (classe ou aspecto) na hierarquia, maior o número de métodos, advices e atributos que o componente deve herdar, o que dificulta seu entendimento. Os componentes que herdam os atributos e as operações são acoplados aos supercomponentes (superclasse ou superaspecto). As alterações a um supercomponente devem ser feitas com cuidado porque elas se propagam a todos os componentes que herdam características. Se um componente herda atributos e operações de um supercomponente, a coesão desse componente é reduzida. Quanto mais profundo está o componente na árvore de herança, mais difícil é reutilizá-la porque todos os seus supercomponentes devem ser entendidos.

6.4.3
Métrica de coesão

A coesão de um componente é uma medida da proximidade de relacionamento entre seus componentes internos [228]. O conjunto define uma métrica para a coesão: Falta de Coesão nas Operações (LCOO).

Falta de Coesão nas Operações (LCOO)

Definição: Essa métrica mede a falta de coesão de um componente. Se um componente C_1 possui n operações (métodos e advices) $O_1,\ldots, O_n$ então $\{I_j\}$ é o conjunto de variáveis de instâncias usadas pela operação O_j. Considere |P| o número de interseções nulas entre os conjuntos de variáveis de instâncias. Considere |Q| o número de interseções não-vazias entre os conjuntos de variáveis de instâncias. Então: LCOO = |P| – |Q|, se |P| > |Q|, LCOO = 0 caso contrário. LCOO mede a quantidade de pares de método/advice que não acessam a mesma variável de instância. Dessa forma, é uma medida da falta de coesão. Essa métrica estende a

métrica de LCOO de CK. Consideramos advices e métodos de aspectos da mesma forma que CK considera os métodos de classes.

Relevância: Quanto maior a coesão do componente, mais fácil é entender, reutilizar e alterar o componente.

6.4.4 Métricas de tamanho

O tamanho do software mede fisicamente o tamanho do código e do projeto de um sistema de software [68]. O conjunto de métricas incorpora quatro métricas de tamanho: (i) Tamanho do Vocabulário (VS), (ii) Linhas de Código (LOC), (iii) Número de Atributos (NOA) e (iv) Operações Consideradas por Componente (WOC).

Tamanho do vocabulário (VS)

Definição: A VS conta o número de componentes do sistema, ou seja, o número de classes e aspectos. Essa métrica mede o tamanho do vocabulário do sistema [191]. O nome de cada componente é contado como parte do vocabulário do sistema. As instâncias do componente não são contadas.

Relevância: Quanto maior o tamanho do vocabulário, mais difícil é entender o sistema. Quanto mais difícil for entender o sistema, mais complicado é encontrar os componentes que precisam ser alterados durante as atividades de evolução ou de reutilização.

Linhas de Código (LOC)

Definição: Essa métrica conta o número de linhas de código. Essa é a medida tradicional do tamanho. Os comentários de implementação e documentação, além das linhas em branco, não são interpretados como código. Os diferentes estilos de programação normalmente influenciam os resultados da aplicação dessa métrica. Estudos empíricos (Capítulo 8) resolveram esse problema garantindo que o mesmo estilo de programação fosse usado nos dois projetos.

Relevância: Quanto maior o número de linhas de código, mais difícil é entender o sistema. Quanto maior o número de linhas de código, mais difícil é entender as linhas que precisam ser alteradas durante as atividades de evolução e reutilização.

Número de Atributos (NOA)

Definição: Essa métrica conta o vocabulário interno de cada componente, ou seja, o número de atributos de cada classe ou aspecto. Os atributos herdados não são incluídos no cálculo.

Relevância: Quanto maior o número de atributos por componente, mais difícil é entender o sistema. Quanto maior o número de atributos, mais difícil é encontrar os locais do programa que precisam ser alterados durante as atividades de evolução e reutilização.

Operações Consideradas por Componente (WOC)

Definição: Essa métrica mede a complexidade de um componente em termos de suas operações. Considere um componente C_1 com operações (métodos ou advices) O_1 ,..., O_n. Considere c_1 ,…, c_n a complexidade das operações. Então: WOC = c_1 + ... + c_n. Essa métrica estende a métrica de WOC de CK [46]. Essa métrica não especificou originalmente a complexidade de cada operação [46]. No entanto, neste trabalho, a medida da complexidade da operação é obtida contando o número de parâmetros da mesma, supondo que uma operação com mais parâmetros será provavelmente mais complexa. A métrica trata os advices e métodos de aspectos da mesma forma que CK trata os métodos de classes.

Relevância: Quanto maior o número e a complexidade das operações por componente, mais difícil é entender o sistema. Quanto maior o número e a complexidade das operações por componente, mais difícil é encontrar os locais que precisam ser alterados durante as atividades de evolução e reutilização.

6.5
Avaliação empírica

As métricas e o modelo foram avaliados no contexto de três estudos empíricos com diferentes características, diversos domínios, diferentes níveis de controle e graus de complexidade. O primeiro estudo foi um experimento semicontrolado (Capítulo 8) para comparar o uso de uma abordagem orientada a objetos (com base nos padrões de projeto [79]) e uma abordagem orientada a aspectos para projetar e implementar o Portalware, o sistema multiagentes apresentado na Seção 4.1. O segundo estudo [264] envolveu a aplicação do framework proposto para avaliar as implementações em Java de Hannemann e as implementações de AspectJ dos padrões de projeto GoF [115]. O terceiro estudo [265] foi um estudo comparativo entre uma implementação orientada a aspectos e uma implementação orientada a objetos de sistemas de informação baseados na Web [226], envolvendo os concerns tradicionais, como distribuição, persistência, concorrência e tratamento de exceções.

O Capítulo 8 relata os resultados do primeiro estudo; o objetivo é apresentar uma avaliação substantiva do framework proposto. Esse experimento foi desenvolvido para demonstrar a utilidade do conjunto de métricas e o modelo de qualidade a fim de predizer a manutenibilidade e a reusabilidade de sistemas de software. Reunimos dados sobre o desenvolvimento de duas versões do sistema Portalware: uma versão orientada a aspectos e uma versão orientada a objetos. A seleção desse estudo e a escolha do domínio do agente basearam-se no fato de que não é óbvio quais entidades do problema devem ser desenvolvidas como classes e quais devem ser desenvolvidas como aspectos. Além disso, esse estudo de caso foi escolhido por várias razões: (i) envolve concerns específicos ao domínio e dependentes da aplicação; (ii) seu foco não está nos crosscutting concerns triviais e tradicionais (como auditoria e rastreamento) e (iii) também incorpora concerns que não foram investigados na comunidade de DSOA. Usamos o framework de avaliação para determinar se os projetistas de SMAs fizeram boas escolhas no projeto.

6.6
Discussão e trabalhos relacionados

Até agora, muitos estudos empíricos no contexto do DSOA baseiam-se em critérios subjetivos e em uma investigação qualitativa [62, 82, 115, 138, 141, 226]. Por exemplo, Hannemann e Kiczales comparam as implementações de Java e as implementações de AspectJ dos padrões de projeto GoF [115] em termos de critérios de medição fragilmente definidos, como composabilidade e *plugability*. Apenas alguns trabalhos propuseram métricas de software para o DSOA, como o trabalho de Lopes [162]. Ela definiu um conjunto de diferentes métricas para a separação de concerns. De fato, as métricas CDC, CDO e CDLOC (Seção 6.4.1) inspiram-se, de certa forma, em seu conjunto de métricas. Contudo, as métricas de Lopes só capturam diferentes dimensões da separação de concerns. Além disso, a definição dessas métricas é bastante acoplada a seu estudo empírico e ajustada à distribuição de concerns no código Java. O conjunto de métricas apresentado aqui generaliza suas métricas e é bastante útil para diferentes concerns. Ademais, as métricas propostas são mecanismos de predição para a reusabilidade e manutenibilidade em termos de princípios precisos adicionais, como acoplamento e coesão.

Zhao propôs um conjunto de métricas para o software orientado a aspectos, desenvolvido especificamente para quantificar os fluxos de informações em um programa orientado a aspectos [254]. Sua métrica baseia-se em um modelo de dependência para softwares orientados a aspectos que consiste em um grupo de gráficos de dependência; cada um pode ser usado para representar de forma explícita as diversas relações de dependência em diferentes níveis de um programa orientado a aspectos. Apesar de a métrica de Zhao poder ser vista como complementar a nossas métricas, sua aplicação é incômoda e consome muito tempo, por várias razões. O uso dessas métricas requer que os engenheiros de software construam alguns gráficos de dependência para diferentes níveis de modularidade, como o gráfico de dependência do método (MDG), o gráfico de dependência do advice (ADG), o gráfico de dependência da introdução (IDG) etc. Como conseqüência, essas métricas são muito complexas para entender e usar e requer a implementação de uma ferramenta de análise da dependência que provavelmente diferirá de uma linguagem para a outra.

Ademais, as métricas de Zhao não são derivadas de métricas bem testadas, e o modelo de dependência associado não se baseia em qualquer modelo conhecido da engenharia de software.

Em relação à aplicação do framework de avaliação em diferentes contextos e estudos de casos, há uma crítica geral que pode ser feita às métricas de software propostas neste capítulo. Refere-se a argumentos teóricos no uso de métricas de tamanho convencionais, como LOC e VS (Seção 6.4.4), uma vez que são aplicadas ao desenvolvimento e aos projetos tradicionais (softwares não orientados a aspectos). Entretanto, apesar das limitações conhecidas dessas métricas, aprendemos que sua aplicação não pode ser analisada de forma isolada; e elas se mostraram extremamente úteis em conjunto com as demais métricas do conjunto proposto (Seção 6.4). Além disso, o modelo de qualidade proposto oferece orientações para interpretar os dados gerados por essas métricas no contexto da reusabilidade e manutenibilidade. Ademais, alguns pesquisadores (como Henderson-Sellers [118]) criticaram a métrica LCOO afirmando que ela não tem bases teóricas sólidas e que ainda precisa de validação empírica [15]. Contudo, o autor vê esse problema como um problema geral da pesquisa, em relação às métricas de coesão. No futuro, o autor pretende investigar outras métricas de coesão emergentes com base na dinâmica do programa para incluí-las no framework de avaliação.

Até agora, ainda não foi implementada uma ferramenta para oferecer suporte às métricas propostas. Entretanto, as métricas são fáceis de usar porque a maioria delas se baseia em métricas já conhecidas da comunidade da engenharia de software. Como resultado, alguns ADSs oferecem suporte a elas. Por exemplo, Together [25] foi usado em estudos empíricos (Capítulo 8) para o processo de coleta de dados das métricas de coesão, acoplamento e tamanho. Como as métricas propostas para a SoC são inovadoras, não há qualquer ferramenta que ofereça diretamente suporte à essa aplicação. Como conseqüência, a identificação dos concerns é atualmente uma imposição aos usuários das métricas de SoC, uma vez que muitos concerns são claramente dependentes da aplicação. No entanto, há algumas ferramentas, propostas há pouco tempo na comunidade de DSOA, que podem ajudar os engenheiros de software na identificação de concerns. Por exemplo, a ferramentas FEAT [206]

oferece suporte à localização, descrição e análise do código, implementando um ou mais concerns em um sistema Java.

6.7 Resumo

A construção de sistemas de qualidade é um importante objetivo de todos os esforços da engenharia de software ao longo das duas últimas décadas. A falta de orientação em relação à implementação e projeto pode levar ao uso impróprio das abstrações orientadas a aspectos, piorando a qualidade geral do sistema. Alguns requisitos de qualidade importantes, como a reusabilidade e a manutenibilidade, podem ser afetados negativamente devido ao uso inadequado das linguagens orientadas a aspectos e às respectivas abstrações. Dessa forma, à medida que o DSOA evolui, torna-se necessário um significativo esforço de pesquisa para definir as medidas de qualidade. A medição das propriedades estruturais do projeto de artefatos de software, como acoplamento, coesão e separação de concerns, é uma abordagem promissora em relação às primeiras avaliações de qualidade. Para usar essas métricas de forma eficaz, são necessários modelos de qualidade para a descrição quantitativa de como essas propriedades internas estão relacionadas às qualidades externas relevantes. Entretanto, a pesquisa de DSOA enfatizou basicamente os construtos da linguagem do projeto e implementação. Alguns estudos empíricos foram realizados no contexto do DSOA. Contudo, a avaliação nesses estudos é qualitativa.

Este capítulo apresenta um framework com base em um conjunto de métricas e um modelo de qualidade, para ajudar na avaliação de softwares orientados a aspectos em termos de reusabilidade e manutenibilidade. As métricas propostas satisfazem importantes requisitos a fim de alcançar medições bem-sucedidas no contexto do DSOA (Seção 2.2). O conjunto de métricas baseia-se em atributos conhecidos da implementação e do projeto de software. As expectativas relacionadas às novas e estendidas métricas estão ligadas à necessidade de lidar com novas abstrações e novas dimensões de acoplamento e coesão introduzidas pelo DSOA. Este capítulo também discutiu a utilidade e a reusabilidade do framework de avaliação e métricas associadas, facilitando, assim, o desenvolvimento de estudos empíricos futuros, assim como o refinamento do framework.