4
O Framework de Avaliação

O propósito central do uso de aspectos é melhorar a separação de *concerns*. Entretanto a orientação a aspectos pode afetar também outros atributos de software, tais como acoplamento, coesão e tamanho. Aspectos podem ser eficazes para modularizar *crosscutting concerns*, minimizar código replicado e, como conseqüência, reduzir o tamanho do sistema. Contudo, o uso inapropriado de aspectos pode afetar negativamente esses atributos de software e aumentar a complexidade do sistema.

Métricas de software são o meio mais eficaz para se obter evidências empíricas que podem melhorar o entendimento das diferentes dimensões de complexidade de software [25]. Elas são mais efetivas quando associadas a algum framework de avaliação, que ajuda os engenheiros de software a entender o significado dos dados coletados. Métricas avaliam o uso de abstrações durante o desenvolvimento de software em termos de atributos de qualidade. Por outro lado, em engenharia de software, medições de atributos internos de produtos são conceitos artificiais e, por si só, não têm significado algum [25]. Por isso, a medição de um atributo interno particular, como acoplamento, é útil se relacionado à avaliação de algum atributo externo (ex. reusabilidade) do objeto de estudo.

Nesse contexto, este trabalho de mestrado propõe um framework de avaliação para capturar o entendimento dos atributos separação de *concerns*, acoplamento, coesão e tamanho em termos de sua utilidade como meios para predizer os atributos externos de manutenibilidade e reusabilidade de software orientado a aspectos. De fato, o objetivo do framework é dar apoio à avaliação do projeto e do código de software orientado a aspectos, em comparação com o projeto e o código de software orientado a objetos, com o foco em manutenibilidade e reusabilidade.

Os elementos do framework ajudam na organização do processo de avaliação e na coleta e interpretação dos dados. Os elementos básicos do

framework são: (i) o conjunto de métricas (Capítulo 5) e (ii) o modelo de qualidade. O conjunto de métricas é formado por métricas de separação de *concerns*, tamanho, acoplamento e coesão. O modelo de qualidade descreve os relacionamentos entre os atributos externos, os atributos internos e as métricas. O usuário do framework reutiliza não só as métricas, como também todas as suposições e pontos de vista que justificam o uso de separação de *concerns*, tamanho, acoplamento e coesão como atributos internos de produto que influenciam manutenibilidade e reusabilidade. Essas suposições são apresentadas na Seção 4.1, onde o modelo de qualidade é descrito.

O framework requer alguns artefatos de entrada para o processo de medição. Primeiro, ele requer os documentos do projeto e o código do sistema, onde as métricas serão aplicadas. Em adição, o framework de avaliação requer uma descrição dos *concerns* do sistema para guiar a identificação deles no projeto e no código durante o uso das métricas de separação de *concerns* (Seção 5.1).

O principal contexto em que o framework pode ser usado, como já foi dito, é o contexto em que se compara o projeto e o código de um sistema de software orientado a aspectos com o projeto e o código de um sistema de software orientado a objetos. Os dois estudos de caso apresentados nesta dissertação usaram o framework nesse tipo de contexto (Capítulo 6). Mas acredita-se que ele possa ser usado também em outros contextos. Por exemplo, durante o desenvolvimento de um software orientado a aspectos, as métricas podem ser usadas para indicar pontos do projeto que pareçam estar mal estruturados ou mal implementados. Esses pontos, depois de uma análise mais detalhada, podem vir a ser reestruturados ou reimplementados. O framework pode ser usado também em avaliações periódicas durante a reestruturação de um software orientado a aspectos. Numa dessas avaliações, pode-se verificar que o valor de determinada métrica melhorou em relação ao valor da avaliação anterior. Isso pode, então, indicar que a estratégia de reestruturação usada, por exemplo, a reescrita de determinados métodos de acordo com um determinado estilo, está surtindo efeito e deve ser mantida.

O restante deste capítulo está organizado da seguinte forma. A Seção 4.1 apresenta o modelo de qualidade do framework de avaliação. A Seção 4.2 descreve o uso da abordagem *GQM* na definição do framework. E a Seção 4.3 define como os princípios de projeto e atributos internos de qualidade usados no

framework são influenciados pelo desenvolvimento de software orientado a aspectos.

4.1. O Modelo de Qualidade

O modelo de qualidade proposto neste trabalho define uma terminologia e torna mais claros os relacionamentos entre manutenibilidade e reusabilidade e o conjunto de métricas. Ele foi definido com base em um conjunto de suposições sobre os atributos internos de produto de software e métricas que influenciam manutenibilidade e reusabilidade. Esse conjunto de suposições e, conseqüentemente, a definição do modelo de qualidade foram baseadas em: (i) uma revisão de uma série de modelos de qualidades existentes [19, 21, 22], (ii) definições clássicas de atributos de qualidade [20, 26, 27, 28], (iii) teorias tradicionais de projeto de software, como a teoria de Parnas [29], que é comumente aceita entre pesquisadores e profissionais da indústria, (iv) atributos de qualidade de software que são afetados pelas abstrações e pelos mecanismos de composição e decomposição orientados a aspectos [30, 31]. Só para exemplificar o que dizem alguns dos pesquisadores estudados, Lamb afirma em [28] que anos de experiência mostraram que módulos com alta coesão e baixo acoplamento resultam em sistemas mais fáceis de compreender. Em complementação, Sommerville [27], ao falar sobre qualidade de projeto de software, diz que a facilidade de compreensão de um projeto é importante, pois qualquer pessoa, para realizar mudanças em um projeto, precisa entendê-lo primeiro.

Como já foi dito na Seção 3.1, modelos de qualidade são construídos em forma de árvore, uma vez que um atributo de qualidade pode ser considerado como uma composição de muitos outros atributos de qualidade [19]. O modelo de qualidade proposto neste trabalho é composto de quatro elementos: (i) qualidades, (ii) fatores, (iii) atributos internos e (iv) métricas. A Figura 5 apresenta os elementos do modelo de qualidade. As qualidades são os atributos externos que são o foco do framework de avaliação (manutenibilidade e reusabilidade). Os fatores são atributos de qualidades secundários que influenciam as qualidades, mas que ainda não podem ser medidos diretamente nos artefatos de software. Os atributos internos se referem a propriedades internas de sistemas de software que

influenciam os fatores e, conseqüentemente, as qualidades que se deseja avaliar. Os atributos internos são mais fáceis de medir que as qualidades e fatores [19], por isso as métricas são ligadas a esses atributos. As próximas subseções descrevem os elementos do modelo de qualidade proposto, ao mesmo tempo em que explicitam as suposições que levaram a sua definição e apresentam a definição de cada atributo usado. A Figura 5 mostra apenas as siglas das métricas, que serão descritas mais adiante no Capítulo 5.

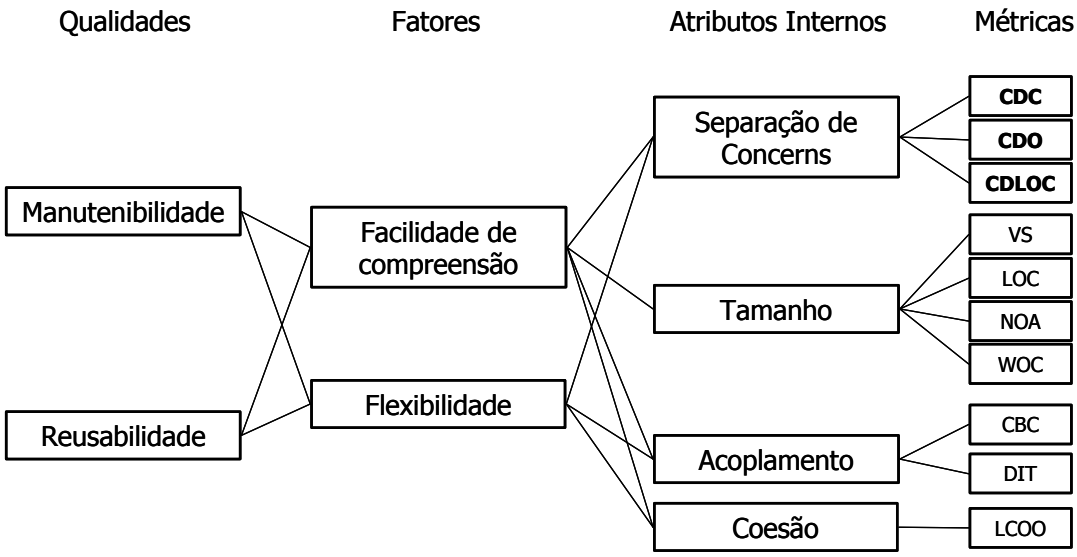

Figura 5 – O Modelo de Qualidade

4.1.1. Qualidades

Como já foi dito anteriormente, manutenibilidade e reusabilidade são o foco do framework de avaliação proposto neste trabalho. **Reusabilidade** é a habilidade de elementos de software servirem à construção de diferentes elementos do mesmo sistema ou de outros sistemas de software [26]. O framework de avaliação tem o objetivo de apoiar a avaliação da reusabilidade de elementos do projeto e do código de sistemas orientados a aspectos de forma a poder compará-la com a reusabilidade de elementos do projeto e do código de sistemas orientados a objetos.

Manutenção é a atividade de modificar um sistema de software depois dele ser entregue ao cliente. **Manutenibilidade** de software é a facilidade com a qual componentes de software podem ser modificados. As atividades de manutenção

podem ser classificadas em quatro categorias [19, 27]: manutenção corretiva, manutenção perfectiva, manutenção adaptativa e manutenção evolutiva (ou simplesmente evolução). Uma vez que o principal objetivo do DSOA é melhorar a evolução de sistemas de software (Capítulo 2), este trabalho está preocupado com a facilidade de evolução de sistemas de software orientados a aspectos, ou seja, com facilidade com que se adiciona, remove ou altera funcionalidades de sistemas de software orientados a aspectos.

Manutenibilidade é um atributo externo de produto, pois não é dependente exclusivamente do produto, mas depende também de pessoas realizando a manutenção [19]. O mesmo pode ser dito para reusabilidade, que depende de pessoas reutilizando artefatos de software. Para não confundir, é bom frisar que reusabilidade é diferente de reutilização. Reusabilidade, como já foi dito, se refere à capacidade de determinado artefato poder ser usado na construção de outros. Enquanto reutilização se refere ao uso já consumado de um artefato de software para a construção de outros artefatos. Reutilização ou grau de reutilização é um atributo interno.

Como manutenibilidade e reusabilidade são atributos externos, a abordagem mais direta para se avaliá-los é medir o processo de manutenção ou o processo de reutilização [19]. Assume-se que quanto mais eficaz for o processo, maior a manutenibilidade ou a reusabilidade o produto. Mas esse tipo de abordagem só pode ser usado no momento em que a manutenção ou reutilização estiver sendo realizada. Por outro lado, como já foi dito anteriormente, atributos internos são mais fáceis de medir e podem ser medidos antecipadamente dentro do ciclo de vida do sistema de software. Portanto, uma abordagem alternativa é identificar atributos internos de produto que podem ser usados para predizer manutenibilidade e reusabilidade, ou seja, antever a eficácia do processo de manutenção ou reutilização. Sendo assim, a contribuição do modelo de qualidade é identificar quais atributos internos exercem influência sobre manutenibilidade e reusabilidade dentro do contexto de uso do framework, que é a avaliação de software orientado a aspectos, em comparação com software orientado a objetos.

4.1.2. Fatores

Antes de chegar aos atributos internos e as métricas, o modelo de qualidade propõe que fatores similares exercem influência tanto em manutenibilidade quanto em reusabilidade. Essa similaridade está relacionada ao fato de que as atividades de manutenção e reutilização compreendem tarefas cognitivas comuns. No modelo proposto, flexibilidade e facilidade de compreensão são os fatores centrais que influenciam manutenibilidade e reusabilidade. [19, 26, 27, 29]. Atividades de manutenção e reutilização requerem abstrações de software que proporcionem boa flexibilidade e facilidade de compreensão.

Facilidade de compreensão indica o nível de dificuldade para estudar e entender o projeto e o código de um sistema [29]. Um projeto e um código fáceis de entender contribuem para melhorar a manutenibilidade e a reusabilidade do sistema, pois a maioria das atividades de manutenção e reutilização requer que os engenheiros de software entendam primeiramente os componentes do sistema antes de modificá-los, estendê-los ou reutilizá-los.

Flexibilidade indica o nível de dificuldade para fazer mudanças em um componente de um sistema sem a necessidade de mudar outros [29]. Ou seja, flexibilidade se refere à questão da propagação, para outros componentes, do efeito de uma mudança em um determinado componente. Para adicionar ou remover alguma funcionalidade do sistema durante as atividades de evolução ou para reutilizar algum componente, é necessário realizar modificações no sistema, portanto quanto mais flexível for o projeto ou código de um sistema, melhor será sua manutenibilidade e reusabilidade, pois o efeito das modificações se propagará por menos componentes do sistema.

4.1.3. Atributos Internos

Os atributos internos que compõem o modelo de qualidade proposto são separação de *concerns*, tamanho, acoplamento e coesão. **Separação de *concerns*** se refere a habilidade para identificar, encapsular e manipular as partes do software que são relevantes para determinado *concern* [3]. O atributo **tamanho** mede fisicamente quão extenso é o projeto e o código do sistema de software [19].

Acoplamento é uma indicação da força das interconexões entre os componentes de um sistema [27]. Sistemas com alto acoplamento têm interconexões fortes, com unidades do programa dependentes entre si [27]. A **coesão** de um componente é a medida da proximidade do relacionamento entre seus componentes internos [27].

No modelo proposto, o fator facilidade de compreensão está relacionado aos seguintes atributos internos: (i) separação de *concerns*, (ii) acoplamento, (iii) coesão e (iv) tamanho. Acoplamento e coesão afetam a facilidade de compreensão, porque um componente do sistema não pode ser entendido sem a referência para os outros componentes com os quais ele está relacionado. O tamanho do projeto ou código pode indicar a quantidade de esforço necessária para entender os componentes do software. O critério de separação de *concerns* é um meio para predizer facilidade de compreensão, pois quanto mais bem localizados estão os *concerns* do sistema, mais fácil é para entendê-los.

O fator flexibilidade é influenciado pelos seguintes atributos internos: (i) separação de *concerns*, (ii) acoplamento e (iii) coesão. Alta coesão, baixo acoplamento e separação de *concerns* são características desejadas, porque elas significam que um componente representa uma parte única do sistema e os componentes do sistema são independentes ou quase independentes. Além disso, os *concerns* do sistema não estão espalhados por vários componentes e misturados entre si. Se for necessário adicionar, remover ou reutilizar alguma funcionalidade, ela estará localizada em um único componente ou em uma única parte do sistema. Conseqüentemente, as atividades de manutenção e reutilização estarão restritas a esse componente ou parte isolada do sistema, o que dá maior flexibilidade para a realização das modificações necessárias durante essas atividades. Note que o fator de flexibilidade não é influenciado pelo atributo tamanho, pois a extensão do projeto ou código de um software não contribui para a propagação para outras partes do sistema dos efeitos de uma modificação de determinado componente.

4.2. Abordagem *GQM*: Objetivo e Perguntas

No sentido de organizar melhor o processo de medição, acredita-se que ele deva ser realizado de acordo com um objetivo específico e um conjunto de perguntas que represente a definição operacional do objetivo. Por isso a

abordagem *Goal-Question-Metric* [23] (Seção 3.2) foi usada para definir o objetivo do framework de avaliação e para derivar dele as perguntas que devem ser respondidas no intuito de determinar se o objetivo foi atingido. A definição das perguntas ajudou na definição do modelo de qualidade (Seção 4.1) e no conjunto de métricas proposto, que será apresentado no Capítulo 5. As perguntas também associam o modelo de qualidade a uma semântica mais precisa para as qualidades, fatores e atributos internos. Além disso, as perguntas podem ajudar os engenheiros de software na interpretação dos dados obtidos durante o processo de medição.

A Figura 6 apresenta o objetivo e as perguntas geradas. As perguntas 1 e 2 são derivadas diretamente do objetivo estabelecido, se referem, portanto, à facilidade de evolução e reusabilidade. Essas duas perguntas são refinadas por perguntas sobre facilidade de compreensão e flexibilidade, que são, por sua vez, refinadas por perguntas sobre separação de *concerns*, acoplamento e coesão. A pergunta sobre facilidade de compreensão é também refinada por perguntas relativas ao tamanho do sistema. As perguntas 1.1.2, 1.2.1, 2.1.2, 2.2.1, sobre separação de *concerns*, podem ser refinadas por uma ou mais perguntas, dependendo do número de *concerns* (representado por N na Figura 6) identificados no sistema a ser avaliado.

4.3. Princípios de Projeto no DSOA

O desenvolvimento de software orientado a aspectos introduz novas abstrações e novas formas de composição e decomposição dos componentes de um sistema (classes e aspectos), conseqüentemente, as abstrações orientadas a aspectos compreendem diferentes dimensões de princípios de projeto e atributos de qualidade, como acoplamento e coesão [30, 31]. Além disso, o DSOA tem impacto direto no tamanho do sistema e na separação dos *concerns* do sistema. As seções a seguir descrevem as diferentes dimensões de acoplamento, coesão, separação de *concerns* e tamanho que são introduzidos pelo DSOA e que motivam a definição de novas métricas ou adaptação de métricas existentes para que esses atributos possam ser medidos em software orientado a aspectos.

Objetivo
Avaliar sistemas orientados a aspectos com o propósito de predição da manutenibilidade e da reusabilidade do ponto de vista do engenheiro de software.

Perguntas

1. Quão fácil é para evoluir o sistema?

    1.1. Quão fácil é para compreender o sistema?
        1.1.1. Quão conciso é o sistema?
            1.1.1.1. Quantos componentes existem no sistema?
            1.1.1.2. Quantas linhas de código existem no sistema?
            1.1.1.3. Quantos atributos existem no sistema?
            1.1.1.4. Quantos métodos e *advices* existem no sistema?
        1.1.2. Quão bem localizados estão os *concerns* do sistema?
            1.1.2.1. Quão espalhada e misturada está a definição de <nome *concern*1>?
            1.1.2.N.Quão espalhada e misturada está a definição de <nome *concern*N>?
        1.1.3. Quão elevado é o acoplamento do sistema?
            1.1.3.1. Quão elevado é o acoplamento entre os componentes?
        1.1.4. Quão elevada é a coesão do sistema?
            1.1.4.1. Quão elevada é a coesão dos componentes do sistema?

    1.2. Quão flexível é o sistema?
        1.2.1. Quão bem localizados estão os *concerns* do sistema?
            1.2.1.1. Quão espalhada e misturada está a definição de <nome *concern*1>?
            1.2.1.N.Quão espalhada e misturada está a definição de <nome *concern*N>?
        1.2.2. Quão elevado é o acoplamento do sistema?
            1.2.2.1. Quão elevado é o acoplamento entre os componentes?
        1.2.3. Quão elevada é a coesão do sistema?
            1.2.3.1. Quão elevada é a coesão dos componentes do sistema?

2. Quão fácil é para reutilizar elementos do sistema?

    2.1. Quão fácil é para compreender o sistema?
        2.1.1. Quão conciso é o sistema?
            2.1.1.1. Quantos componentes existem no sistema?
            2.1.1.2. Quantas linhas de código existem no sistema?
            2.1.1.3. Quantos atributos existem no sistema?
            2.1.1.4. Quantos métodos e *advices* existem no sistema?
        2.1.2. Quão bem localizados estão os *concerns* do sistema?
            2.1.2.1. Quão espalhada e misturada está a definição de <nome *concern*1>?
            2.1.2.N.Quão espalhada e misturada está a definição de <nome *concern*N>?
        2.1.3. Quão elevado é o acoplamento do sistema?
            2.1.3.1. Quão elevado é o acoplamento entre os componentes?
        2.1.4. Quão elevada é a coesão do sistema?
            2.1.4.1. Quão elevada é a coesão dos componentes do sistema?

    2.2. Quão flexível é o sistema?
        2.2.1. Quão bem localizados estão os *concerns* do sistema?
            2.2.1.1. Quão espalhada e misturada está a definição de <nome *concern*1>?
            2.2.1.N.Quão espalhada e misturada está a definição de <nome *concern*N>?
        2.2.2. Quão elevado é o acoplamento do sistema?
            2.2.2.1. Quão elevado é o acoplamento entre os componentes?
        2.2.3. Quão elevada é a coesão do sistema?
            2.2.3.1. Quão elevada é a coesão dos componentes do sistema?

Figura 6 – Objetivo e perguntas geradas com o uso da abordagem *GQM*

4.3.1.
Dimensões de Acoplamento

A Figura 7 ilustra as diferentes formas de combinação entre classes e aspectos, que são fontes potenciais de acoplamento num sistema orientado a aspectos. À esquerda, são apresentadas as formas de acoplamento entre classes existentes no desenvolvimento de software orientado a objetos (DSOO), e, portanto, existentes também no desenvolvimento de software orientado a aspectos (DSOA), já que a orientação a aspectos estende a orientação a objetos. C1 representa o acoplamento ocorrido por causa do relacionamento de especialização/generalização entre duas classes. C2 e C3 representam o acoplamento relativo aos relacionamentos de dependência e associação entre classes respectivamente.

O lado direito da Figura 7 mostra as novas dimensões de acoplamento introduzidas pelo DSOA. São sete formas de acoplamento entre classe e aspecto e entre aspecto e aspecto. O acoplamento do tipo C4 ocorre quando um aspecto Y introduz um atributo, um método, uma declaração de extensão de uma classe ou uma declaração de implementação de interface em uma classe X, por meio do mecanismo de *inter-type declaration*. Neste caso, o aspecto está acoplado a classe, mas a classe não está acoplada ao aspecto, pois não o conhece. O acoplamento C5 ocorre quando uma classe acessa um atributo ou método introduzido por um aspecto. C6 é a forma de acoplamento entre aspectos e classes que existe com mais freqüência. Ele acontece quando um *pointcut* de um aspecto Y define um ponto da execução da classe X que é interceptado pelo aspecto. Neste caso, mais uma vez, o aspecto está acoplado à classe, mas a classe não está acoplada ao aspecto, pois não o conhece. C7 acontece quando um método de um aspecto X acessa um atributo ou método introduzido por um aspecto Y. De maneira semelhante, C9 acontece quando um *advice* de um aspecto X acessa um atributo ou método introduzido por um aspecto Y. A forma de acoplamento C8 ocorre quanto um *pointcut* define que um aspecto X intercepta a chamada de algum método ou o acesso a um atributo introduzido por um aspecto Y. Finalmente, C10 representa o acoplamento ocorrido por causa do relacionamento de especialização/generalização entre dois aspectos.

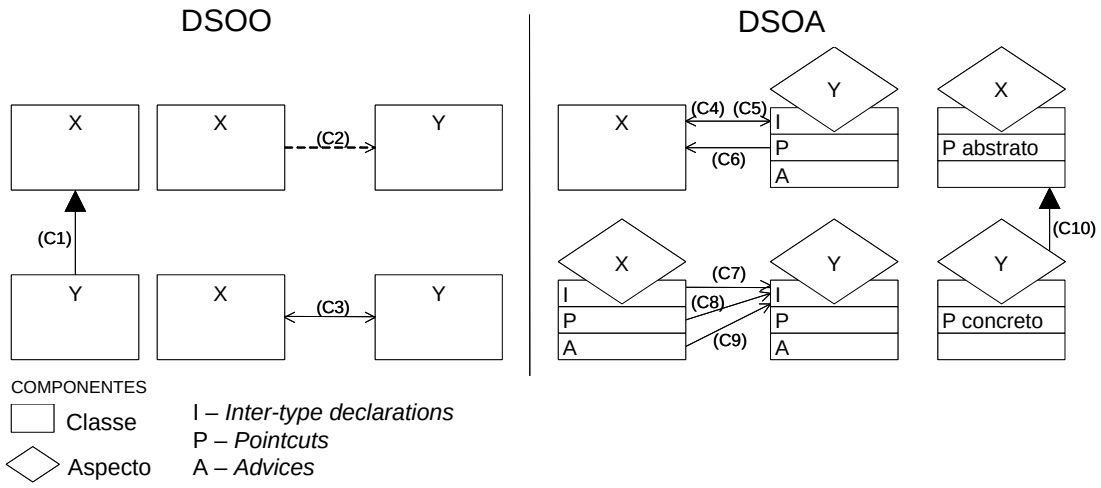

Figura 7 – Dimensões de acoplamento no DSOA

4.3.2. Dimensões de Coesão

Existem algumas métricas na literatura para medir a coesão interna das classes no desenvolvimento de software orientado a objetos, tais como as propostas por Chidamber & Kemerer [6], Henderson-Sellers [7] e Hitz & Montazeri [32]. A maioria delas se baseia no relacionamento entre os métodos da classe, como, por exemplo, a métrica proposta por Chidamber & Kemerer que mede a quantidade de pares de métodos que não acessa um mesmo atributo.

Como já foi dito no Capítulo 2, na linguagem AspectJ, existe um tipo de construção parecida com um método, chamada de *advice*, que é usada para se definir um trecho de código que deve ser executado quando um *pointcut* é atingido. Portanto, além dos métodos, os *advices* devem ser levados em consideração quando se for medir a coesão de um aspecto.

4.3.3. Dimensões de Tamanho

O tamanho do projeto ou do código de um sistema de software orientado a objetos pode ser avaliado de acordo com vários pontos de vistas. Por exemplo, pode-se contar o número de componentes do sistema ou de parte do sistema, no caso da linguagem Java, o número de classes e interfaces. Uma classe pode ter seu

tamanho avaliado em termos da quantidade de atributos, da quantidade de métodos, da extensão dos métodos, entre outras maneiras.

O desenvolvimento de software orientado a aspectos introduz novas abstrações, mecanismos e construções, que devem ser considerados ao se avaliar o projeto e o código de software orientado a aspectos. Portanto, na linguagem AspectJ, que é uma extensão da linguagem Java, assim como as classes e as interfaces, os aspectos também devem ser levados em consideração ao se contar o número de componentes que formam o sistema. Além disso, a avaliação do tamanho de um aspecto deve atentar para *pointcuts*, *advices* e *inter-type declarations*, que são construções exclusivas de aspectos, além de atributos e métodos.

4.3.4. Dimensões de Separação de *Concerns*

A separação de um determinado *concern* de um sistema pode ser avaliada em termos de duas perspectivas complementares: (i) quão espalhado pelos componentes do sistema o *concern* está e (ii) quão misturado (ou emaranhado) com os outros *concerns* do sistema o *concern* está. Num sistema orientado a objetos, para se saber o nível de espalhamento de um *concern*, pode-se verificar quais as classes e métodos que o implementam. Num sistema orientado a aspectos, deve-se verificar quais as classes, aspectos, métodos e *advices* que implementam tal *concern*. Além disso, para se ter a noção de quão misturados entre si estão os *concerns*, deve-se verificar a existência de vários *concerns* sendo implementados pelas mesmas classes e aspectos e pelos mesmos métodos e *advices*.

Vale destacar que essas duas perspectivas são complementares para avaliar se determinado *concern* está bem modularizado. Um *concern* pode ser implementado por várias classes e aspectos, mas essas classes e aspectos implementarem unicamente esse *concern*. Nesse caso, o *concern* está bem separado, pois apesar de estar espalhado por vários componentes, não está misturado com outros *concerns*. Por outro lado, um *concern* pode ser implementado por poucas classes ou aspectos, mas essas classes ou aspectos implementarem vários outros *concerns* ao mesmo tempo. Já nesse caso, o *concern*

não está bem separado, pois, apesar de não estar muito espalhado, está misturado com outros *concerns*.