4
Engenharia Empírica e Medição de Software

"A qualidade nunca se obtém por acaso; ela é sempre o resultado do esforço inteligente." (John Ruskin)

Neste capítulo são apresentados conceitos relacionados a Engenharia Empírica e a medição de software (Seção 4.1). Posteriormente, na Seção 4.2 serão apresentadas todas as métricas utilizadas no contexto desse trabalho. Por fim, um breve resumo é apresentado na Seção 4.3.

4.1
Conceitos

A Engenharia de Software Empírica (ESE) é uma sub-área da Engenharia de Software que objetiva melhorar uma série de métodos e técnicas de forma a permitir um melhor processo de desenvolvimento de software. Sendo assim, como forma a melhorar tais métodos e técnicas de desenvolvimento de software, é que as hipóteses de pesquisas são formuladas e comprovadas a partir de estudos experimentais. Segundo Basili et al. (Basili et al. 1986), os estudos experimentais são uma maneira eficaz de fornecer comprovações empíricas que podem melhorar a compreensão das técnicas e métodos da Engenharia de Software. De acordo com (Wohlin et al. 2000), existem três estratégias empíricas diferentes, dependendo das condições investigadas: (i) *Survey*: é um estudo investigativo, por exemplo, de uma ferramenta ou técnica. Neste tipo de estudo não existem controle da execução ou medições; (ii) *Estudos de Casos*: são utilizados para se monitorar projetos e atividades. É visto como um estudo que é feito de forma observacional. O nível de controle empregado nestes tipos de experimentos é baixo em relação ao experimento; e (iii) *Experimento*: é algo mais formal e rigoroso. Um experimento é um estudo mais controlado e com alto grau de controle.

A qualidade de software pode ser avaliada a partir de vários atributos, como: funcionalidade (utilidade), confiabilidade, utilizabilidade, eficiência, manutenibilidade, facilidade de compreensão, dentre outros. No contexto da atividade de manutenibilidade, três atividades são consideradas (Fowler et al. 1999): (i) corretiva: o objetivo desta atividade é a eliminação

de algum defeito; (ii) adaptativa: adaptação do sistema para que ele possa executar em outro ambiente; e (iii) perfectiva: novas modificações são adicionadas ao sistema em detrimento a uma evolução dos requisitos.

O conceito de qualidade de software vêm sendo amplamente utilizado na literatura, porém com diversas definições. (von Staa 2000) define a qualidade de um artefato de software como um conjunto de propriedades a serem satisfeitas em determinado grau, de modo que este satisfaça as necessidades de seus usuários e clientes. Segundo (Sommerville 2001), a qualidade compreende a total adequação com os requisitos.

Dado isso, a medição de software é uma parte importante no gerenciamento dos projetos e é utilizada justamente para prover indícios de como melhorar alguns fatores da qualidade do sistema. No processo de medição de software, algumas entidades importantes são avaliadas com o propósito de quantificar atributos que possibilitem a caracterização e a melhoria da qualidade (Kitchenham et al. 1995). Essas entidades podem ser artefatos, produtos e processos. No contexto deste trabalho, várias características devem estar presentes em uma LP-SMA, como: manuntenibilidade, reusabilidade e modularidade.

4.2 Métricas

Muitas métricas de software têm sido propostas para o paradigma orientado a objetos (Henderson-Sellers 1996, Chidamber and Kemerer 1994, Etzkorn and Delugach 2000). O objetivo das métricas é permitir a melhoria de aspectos da qualidades de software. Dado isso, as métricas de software vêm sendo amplamente utilizadas no desenvolvimento de sistemas OO com o intuito de melhorar a fidedignidade do software e os artefatos gerados no processo. Mais recentemente, algumas dessas métricas foram adaptadas para o desenvolvimento de software orientado a aspectos (DSOA) (Sant'Anna et al. 2003). Sant'Anna et al. (Sant'Anna et al. 2003) propuseram um *framework* de avaliação baseado em um conjunto de métricas para o DSOA. O objetivo deste *framework* é a medição de dois atributos de qualidade no DSOA, que são: reusabilidade e manutenibilidade. O modelo da qualidade proposto é composto por quatro níveis: características externas, fatores, atributos internos e métricas. A Figura 4.1 ilustra o modelo de qualidade proposto em (Sant'Anna et al. 2003). 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, tamanho e interação entre *concerns*. O objetivo dessas métricas é permitir uma melhor modularização dos interesses

e como conseqüência, permitir que a manutenibilidade e a reusabilidade do sistema aumente sobremaneira.

As métricas propostas por (Sant'Anna et al. 2003) já foram utilizadas em alguns estudos empíricos recentes (Garcia et al. 2003a, Garcia et al. 2005, Kulesza et al. 2006b, Figueiredo et al. 2008, Greenwood et al. 2007). Mais recentemente, (Eaddy et al. 2008) utilizou duas métricas de separação de *concerns*, CDC e CDO (Seção 4.2.1), em um estudo empírico controlado com o intuito de validá-las. Eaddy et al. (Eaddy et al. 2008) considera que a possibilidade da causa de um defeito é a pobre modularização dos interesses do programa. Nesse trabalho eles argumentam que os interesses transversais impactam em pelo menos um atributo de qualidade externa, que são os defeitos no software.

No contexto deste trabalho de mestrado, o objetivo da utilização dessas métricas é fornecer evidências empíricas na modularização de *features*. Isto é, quais mecanismos de variabilidade são mais adequados na modularização de *features* de agentes no contexto evolutivo de LP-SMA (Capítulo 2). Abaixo são descritas as métricas utilizadas nesta dissertação.

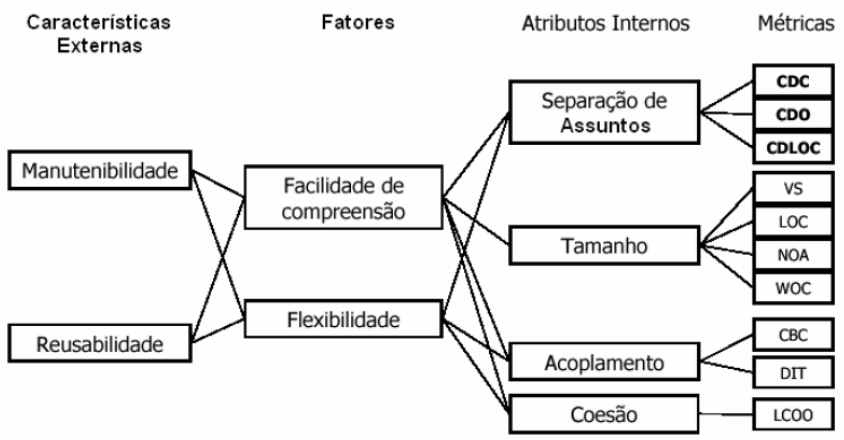

Figura 4.1: Modelo de Qualidade.

4.2.1 Métricas de Separação de Concerns

As métricas de separação de *concerns* se referem a capacidade de conseguir encapsular as principais partes de um interesse. O conjunto de métricas de separação de *concerns* compreende as seguintes métricas: Difusão do *Concern* por Componentes (CDC), Difusão do *Concern* por Operações (CDO) e Difusão do *Concern* por Linhas de Código (CDLOC).

Difusão do Concern por Componentes (CDC)

A métrica CDC conta o número de componentes que são responsáveis pela implementação de um determinado *concern*. O número de componentes inclui classes, interfaces e aspectos. Além disso, conta o número de componentes que fazem referência a esses componentes principais, como: declarações de atributos, parâmetros dos métodos, tipos de retorno, variáveis locais ou chamando seus métodos. A métrica CDC é importante para medir o quanto um determinado *concern* está espalhado por entre os componentes do software.

Difusão do Concern por Operações (CDO)

A métrica CDO conta o número de operações (métodos e *advices*) principais que contribuem para a implementação de um *concern*. Ela também conta o número de operações que fazem referência a essas operações principais, chamando seus métodos ou utilizando-os em parâmetros, tipos de retorno, declarações e variáveis locais. Os construtores também são contados como operações. Essa métrica mede o espalhamento do *concern* por entre as operações, quanto mais operações fizerem referência a um determinado *concern*, mais difícil é para entender e realizar manutenção no software.

Difusão do Concern por Linhas de Código (CDLOC)

A métrica CDLOC verifica o quanto um determinado *concern* está misturado com outros *concerns*. CDLOC conta o número de pontos de transição para cada *concern* por linhas de código. Para a contagem desta métrica é necessário sombrear as partes do código que implementam um determinado *concern*. Após esse processo, são contados os pontos de transição, que são os pontos no código onde existe transição entre uma área não-sombreada para uma área sombreada e vice-versa. Quanto maior o CDLOC, mais entrelaçado está o código de um determinado *concern* com outros *concerns*.

4.2.2 Métricas de Acoplamento

O acoplamento é uma métrica para medir o quanto os componentes do software estão conectados. O acoplamento indica o quanto um determinado componente está relacionado com outros. Assim, o entendimento de um determinado componente depende do entendimento dos outros componentes ao qual ele está acoplado. Portanto, quanto maior o acoplamento, mais difícil é o entendimento e a realização de manutenção do software. Quanto menos acoplado é um componente, mais fácil é para reutilizá-lo em outros

sistemas. A métrica de acoplamento utilizada neste trabalho é a métrica CBC (Acoplamento entre Componentes).

Acoplamento entre Componentes (CBC)

A métrica CBC é definida para um componente (classe ou aspecto) como sendo um número de outros componentes com os quais ele está acoplado. Para cada componente, conta-se o número de outras classes que são usadas em declarações de atributos, tipos de parâmetros, de operações, tipos de retorno ou variáveis locais. Esta métrica é relevante, pois o entendimento do sistema pode ser bastante prejudicado devido ao alto acoplamento entre os componentes. Portanto, esta métrica dá indícios do quanto a modularidade do sistema pode ser melhorada de forma a permitir um baixo acoplamento.

4.2.3 Métricas de Coesão

A coesão identifica as relações internas de um determinado componente (classe ou aspecto). De acordo com a métrica de coesão, quanto maior for o grau das diferentes ações realizadas pelo componente, mais difícil é a manutenção e a reutilização do componente ou de alguma da suas funcionalidades. A coesão é um atributo importante de qualidade interna, pois permite um melhor entendimento do componente. Abaixo é detalhada a métrica LCOO (Falta de Coesão em Operações).

LCOO

A métrica LCOO mede as operações que compartilham atributos menos o número de pares de operações que não compartilham nenhum atributo. Diante disso, quanto maior é a coesão do componente, mais fácil é entender, reutilizar e alterar o componente.

4.2.4 Métricas de Tamanho

As métricas de tamanho, como o próprio nome diz, se refere ao tamanho de um determinado sistema. As métricas de tamanho estão muito relacionadas com o entendimento de um determinado sistema, pois quanto maior for o tamanho do sistema, mais difícil para entender e identificar as partes que precisam ser alteradas durante uma atividade de manutenção. As métricas de tamanho utilizadas neste trabalho são: número de linhas de código (LOC), número de operações (NOO), número de componentes (NOC).

Número de Operações (NOO)

A métrica NOO conta o número de operações do sistema. Esta métrica está relacionada com o entendimento do sistema, pois quanto maior o número de operações, mais difícil é encontrar os locais do programa que precisam ser modificados durante as atividades de evolução e reutilização.

Número de Componentes (NOC)

A métrica NOC conta o número de componentes (classes e aspectos) do sistema. Quanto maior o número de componentes, mais difícil é entender o sistema. Sendo difícil o entendimento do sistema, mais complicado é encontrar os componentes que precisam ser alterados durante as atividades de evolução e reutilização.

Número de Linhas de Código (LOC)

Número de Linhas de Código (LOC) é a medida mais simples do software. Os diferentes estilos de programação normalmente influenciam os resultados da aplicação dessa métrica.

4.2.5 Interação entre Concerns

As métricas de interação entre *concerns* avaliam as dependências causadas por *concerns* que não estão bem modularizados. Alguns *concerns* não são inteiramente encapsulados por componentes e, portanto, a manutenibilidade pode ser prejudicada devido a mudança em um componente prejudicar os outros *concerns*. Sendo assim, a métrica de interação entre concerns (CIBC) utilizada foi proposta por (Sant'Anna et al. 2007).

Interação entre Concerns em Nível de Componente (CIBC)

A métrica CIBC conta o número de outros *concerns* com o qual um dado *concern* compartilha pelo menos um componente. Um *concern* muito dependente de outros *concerns* em nível de componente pode prejudicar a modularidade e a manutenibilidade do mesmo.

4.3 Resumo

A construção de sistemas que primam por um certo nível de qualidade é uma tarefa que exige muito esforço. Diante dos paradigmas de programação existentes (OO, OA), o uso inadequado de certos mecanismos de implementa-

ção pode levar a baixa qualidade do sistema, ocasionando a uma baixa reusabilidade e manutenibilidade. A abordagem quantitativa dos estudos empíricos objetiva identificar um relacionamento causa-efeito, onde é possível relacionar os dados coletados, através das métricas de software, ao paradigma utilizado no desenvolvimento do sistema. Sendo assim, com a avaliação da modularidade e da estabilidade é possível identificar quais aspectos da qualidade precisam ser melhorados no sistema. Neste sentido, neste capítulo foram descritos de forma sucinta os atributos de modularidade utilizados neste trabalho: separação de *concerns*, acoplamento, coesão, tamanho e iteração entre *concerns*.