3
Estado da Arte e Trabalhos Relacionados

Neste capítulo resumimos alguns trabalhos existentes na literatura que se relacionam à abordagem de avaliação proposta nesta dissertação. O objetivo de todo método quantitativo de avaliação é garantir a qualidade de um processo ou produto de software. Na Seção 3.1 são abordados alguns conceitos relacionados à qualidade de software. Em seguida, apresentamos as principais métricas de software (Seção 3.2) e estado da arte em avaliação quantitativa para DSOA (Seção 3.3). Na Seção 3.4 é feita uma revisão de ferramentas que dão suporte a este paradigma e para finalizar o capítulo são apontadas algumas limitações dos trabalhos relacionados.

3.1.
Qualidade de Software

A definição de qualidade de software não é simples, pois o termo qualidade é bastante subjetivo e também porque software é intangível. Qualidade, de uma maneira geral, pode ser definida como "atendimento aos requisitos" ou "adequado para uso" [40] [61]. Entretanto, qualidade para o produto de software requer uma definição mais específica do que as definições tradicionais. Assim, alguns autores como Stephen Kan [40] relacionam qualidade de software à ausência de *bugs*, podendo ser medida pela taxa de erros (erros por milhares de linhas de código) ou pela confiabilidade (horas de operação sem falhas). Staa [62] define qualidade de um artefato de software como um conjunto de propriedades a serem satisfeitas em determinado grau, de modo que este satisfaça às necessidades de seus usuários e clientes. Em relação às definições acima, pode-se observar que elas geralmente abordam características explícitas do software, ou seja, características que podem ser claramente documentadas na especificação de requisitos. Por outro lado, há um conjunto de propriedades ou requisitos implícitos que deve ser observado e frequentemente não é mencionado como, por exemplo, desejo de dispor-se de elevada manutenibilidade.

Em [40], Kan coloca qualidade de software como “conformidade a requisitos funcionais e de desempenho explicitamente declarados, a padrões de desenvolvimento claramente documentados e a características implícitas que são esperadas de todo software profissionalmente desenvolvido”. Ou seja, se o software se adequar aos seus requisitos explícitos, mas deixar de cumprir seus requisitos implícitos, sua qualidade será suspeita [40]. Desta forma, para o contexto desta dissertação, várias características implícitas devem estar presentes em um software para que este seja considerado um produto de qualidade. Algumas destas características são frequentemente descritas e trabalhadas na literatura [20] [40] [57] [61] como: manutenibilidade, flexibilidade, testabilidade, reusabilidade, concisão, modularidade, simplicidade ou facilidade de compreensão.

Como mencionado no Capítulo 2, o uso de técnicas de DSOA promete melhoria de muitos destes requisitos de qualidade. Entretanto, alcançar um nível satisfatório de qualidade, mesmo em um sistema orientado a aspectos, não é trivial e requer um acompanhamento por medição e por abordagens sistemáticas de avaliação. Além disso, um processo automatizado torna-se crucial devido ao grande volume de informação que pode ser extraído do sistema. Nas seções a seguir são apresentadas métricas de software (Seção 3.2) e como estas vêm sendo usadas na avaliação e melhoria da qualidade de software orientado a aspectos (Seção 3.3).

3.2. Métricas de Software

O desenvolvimento de grandes sistemas é uma atividade que consome muito tempo e recursos. Sabendo-se que cada etapa deste processo requer esforço, é preciso prover informações para que a equipe tome decisões, trace planos, agende atividades e aloque recursos. Desta forma as métricas de software tornam-se necessárias para identificar onde é necessário efetuar as buscas por melhoria nos artefatos gerados no processo de desenvolvimento, sendo ainda uma fonte crucial para a tomada de decisões [20]. Muitas métricas têm sido propostas na literatura [14] [37] [47] [57], porém, neste trabalho é dado enfoque em métricas de produto. Métricas de produto devem ser fáceis de serem obtidas, pois com diferentes versões de um software, há bastante informação a analisar. Preferencialmente,

estas métricas devem ser passíveis de serem automatizadas. Nesta seção são abordadas as principais métricas que podem ser aplicadas a sistemas implementados sobre o paradigma de desenvolvimento orientado a objetos (OO) ou orientado a aspectos (OA). Elas são classificadas em quatro categorias: tamanho, acoplamento, coesão e separação de interesses.

3.2.1. Métricas de Tamanho

Algumas métricas são utilizadas para identificar a concisão, ou tamanho, de um sistema e estas tipicamente contam o número de ocorrência de um determinado elemento, como o número de subsistemas existentes (pacotes em Java ou AspectJ) ou o número de classes. Em certos casos, o número de classes pode ser generalizado para o **Tamanho do Vocabulário (VS)** [57], o que inclui tanto classes, quanto interfaces e aspectos. Métricas de tamanho também podem contar o número de membros de um componente do sistema [40]. Nesta contagem podem ser feito distinção de tipos específicos de membros, como ocorre nas métricas **Número de Atributos (NOA)** [47] [57], **Número de Operações (NOO)** [20] [22] e **Peso das Operações por Componentes (WOC)** [13] [14] [57].

O **Número de Linhas de Código (LOC)** [20] [47] é uma das medidas mais simples que pode ser obtida de um software. Entretanto, vários fatores podem gerar uma diferença significativa no resultado de sua contagem. Na programação em assembler, em que cada linha de código corresponde a um comando, a definição desta métrica é clara. Com as linguagens de mais alto nível, a contagem de linhas de código pode ser influenciada, dentre outras coisas, pela consideração ou não de comentários, linhas em branco e declarações que não são executadas – como delimitadores, por exemplo. Portanto, dependendo da forma que é contado, o número de linhas de código pode ser fortemente influenciado pelo estilo de programação adotado. Existem padrões para contagem de linhas de código [20] que diminuem a influência de estilo do programador. Entretanto, estes padrões nem sempre são implementados pelas ferramentas. Uma métrica de tamanho menos influenciada pelo estilo de programação e que pode ser utilizada como alternativa a LOC é a contagem do **Número de Comandos (NOS)** [20] [22].

3.2.2.
Métricas de Acoplamento

Métricas de acoplamento indicam o número de outros componentes que se relacionam a um determinado componente. A forma mais genérica de se medir acoplamento é feita pela métrica de **Acoplamento entre Componentes (CBC)** [13] [14]. Em CBC são considerados de forma homogênea todos os tipos de relacionamentos entre classes e aspectos como dependências por atributos, operações, heranças, conjuntos de junção ou declarações intertipo. Um tipo especial de acoplamento que é tratado de forma diferenciada por algumas métricas ocorre na hierarquia de herança entre classes e aspectos. Dois exemplos bem conhecidos são **Profundidade da Árvore de Herança (DIT)** [13] [14] [57] e **Número de Filhos (NOC)** [13] [14] [22]. DIT mede o número de níveis da hierarquia até que se atinja a raiz da árvore e NOC conta subtipos imediatos de um componente.

3.2.3.
Métricas de Coesão

Existem muitos estudos na literatura em relação a medidas de coesão interna de componentes [13] [14] [37] [57]. A maior parte das métricas propostas se baseia em relacionamentos entre métodos. Um exemplo clássico é a métrica **Perda de Coesão em Operações (LCOO)** [13] [14] [57] que mede o quão interligado estão as operações em relação ao compartilhamento de atributos. O valor desta métrica pode ser obtido pela contagem do número de pares de operações que compartilham atributos, menos o número de pares de operações que não compartilham nenhum atributo.

3.2.4.
Métricas de Separação de Interesses

Como visto no Capítulo 2, a motivação para o desenvolvimento orientado a aspectos é a modularização de interesses que podem não ser bem capturados por mecanismos e abstrações de outros paradigmas. Desta forma, novas métricas têm sido propostas recentemente para indicar o nível de espalhamento e entrelaçamento dos interesses através dos artefatos de software [22] [57]. Uma característica comum destas métricas, chamadas métricas de separação de

interesses (SI), é que elas precisam de uma atividade prévia de identificação dos elementos pertencentes aos interesses. Esta atividade é chamada de *sombreamento do interesse* [31] [57]. Além disso, tais métricas oferecem possibilidade de serem aplicadas tanto em sistemas OA como em sistemas OO e este fato viabiliza a comparação de resultados obtidos a partir de sistemas implementados segundo os dois paradigmas de desenvolvimento.

A métrica **Difusão de Interesse em Componentes (CDC)** [31] [57] conta o número de componentes cujo propósito principal é contribuir para a implementação do interesse avaliado. Além disso, CDC conta também o número de componentes que fazem referência aos componentes principais do interesse. Para melhor entender esta métrica, utilizamos o diagrama de classe da Figura 6 que apresenta a implementação orientada a objetos de um editor de figuras. Os elementos sombreados destacam o código utilizado para implementar o padrão de projeto *Observer* [26]. Supondo que se queira verificar o espalhamento do interesse referente ao padrão *Observer* neste sistema é obtido valor 5 para CDC, uma vez que todas as classes e interfaces do diagrama possuem algum código referente a este interesse.

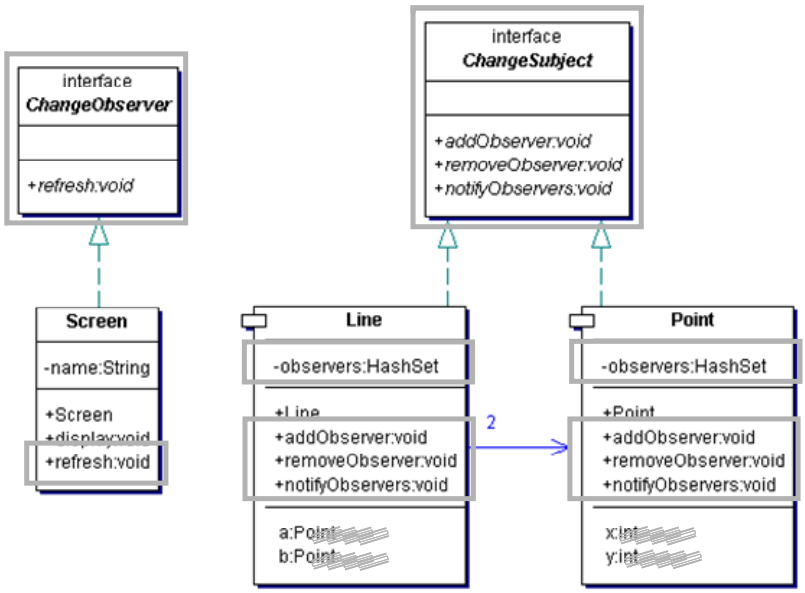

Figura 6 – Diagrama de classes OO do editor de figuras

Outras duas métricas de SI propostas na literatura [31] [57] são: **Difusão de Interesse em Operações (CDO)** e **Difusão de Interesse em Linhas de Código (CDLOC)**. A primeira (CDO) conta o número de operações cujo propósito principal é contribuir par a implementação do interesse avaliado. Esta métrica

conta ainda os métodos, construtores e adendos que acessam alguma das operações principais do interesse. A contagem do espalhamento de interesse referente ao padrão *Observer* sobre as operações no sistema da Figura 6 resulta em quinze. A segunda métrica (CDLOC) conta o número de pontos de transição existentes no código entre o interesse avaliado e os demais interesses do sistema. Para melhor entendimento desta métrica é apresentado o sombreamento de código da classe `Line` na Figura 7. Este sombreamento divide o código em áreas sombreadas e não sombreadas. Os pontos de transição são os pontos em que ocorre a mudança entre áreas sombreadas e não-sombreadas ou vice-versa. No caso da classe `Line` o número de pontos de transição é 8, ou seja, a contagem de CDLOC para esta classe resulta em oito.

```
public class Line
                  implements ChangeSubject {
   private HashSet observers = new HashSet();
   private Point a, b;

   public Line(Point x, Point y) {
      this.a = x;
      this.b = y;
   }

   public Point getA() { return a; }
   public Point getB() { return b; }

   public void setA(Point x) {
      this.a = x;
      notifyObservers();
   }

   public void setB(Point y) {
      this.b = y;
      notifyObservers();
   }

   public void addObserver(ChangeObserver o) {
      this.observers.add(o);
   }
   public void removeObserver(ChangeObserver o) {
      this.observers.remove(o);
   }
   public void notifyObservers() {
      for (Iterator e = observers.iterator(); e.hasNext();)
         ((ChangeObserver)e.next()).refresh(this);
   }
}
```

1 2 3 4 5 6 7 8

Figura 7 – Sombreamento da classe `Line` do editor de figuras

3.3. Avaliação de Software Baseado em Métricas

Como discutido na Seção 3.1, a qualidade de software depende de características implícitas como reusabilidade e manutenibilidade. No entanto, tais características são normalmente difíceis de serem avaliadas e só podem ser medidas tardiamente no processo de desenvolvimento. Para contornar este problema, as abordagens de avaliação existentes que são baseadas em métricas procuram usar atributos mensuráveis dos artefatos de software para predizer características de qualidade. Uma forma comum de se fazer este mapeamento é utilizando modelos de qualidade [4] [22] [57].

Modelos de qualidade são geralmente construídos sob a forma de uma árvore, em que os vértices mais próximos da raiz representam as características de qualidade que se deseja avaliar e os vértices mais próximos das folhas representam atributos de mais fácil medição. Pela bibliografia revisada, os primeiros modelos de qualidade construídos desta forma foram propostos na década de setenta por McCall *et al*. [50] e Boehm *et al*. [4]. Recentemente, entretanto, Sant'Anna *et al*. [57] propuseram um modelo de qualidade, revisado em [22], para avaliação de software orientado a aspectos. Este modelo utiliza métricas de software para predizer características de manutenibilidade e reusabilidade como apresentado a seguir.

3.3.1. Um *Framework* de Avaliação Orientado a Aspectos

Sant'Anna *et al*. [31] [57] propõem um *framework* baseado em métricas para avaliação de software orientado a aspectos. Além do conjunto de métricas, seu *framework* possui um modelo de qualidade para medir graus de reusabilidade e manutenibilidade. O modelo mapeia tais características de qualidade em atributos mensuráveis a partir do código fonte, como separação de interesses, coesão, acoplamento e tamanho. Este modelo é composto por quatro níveis: características externas, fatores, atributos internos e métricas. A figura a seguir ilustra o modelo de qualidade proposto por Sant'Anna.

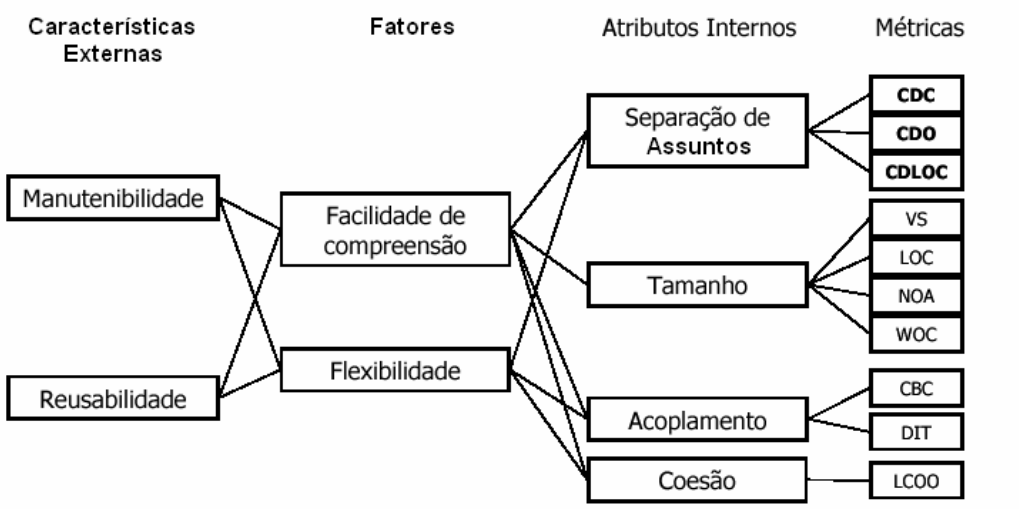

Figura 8 – Modelo de qualidade proposto por Sant'Anna *et al*. [57]

As características de qualidade são colocadas nos dois primeiros níveis do modelo de qualidade apresentado na Figura 8. Manutenibilidade e a reusabilidade estão presentes no primeiro nível e são denominadas características externas. Flexibilidade e simplicidade (ou facilidade de compreensão), presentes no segundo nível, são denominadas fatores. Esta distinção é feita porque, segundo os autores [31] [57], simplicidade e flexibilidade influenciam as duas características de qualidade do nível anterior. Além disso, o modelo de qualidade é originalmente proposto para avaliar a manutenibilidade e a reusabilidade de software orientado a aspectos.

No terceiro nível do modelo são colocados os quatro atributos mensuráveis a partir de artefatos de software: separação de interesses, tamanho, acoplamento e coesão. Estes atributos se baseiam em princípios bem estabelecidos da Engenharia de Software e conectam aos fatores do nível anterior e a um conjunto de métricas no nível seguinte. De acordo com o atributo que se propõe medir, as métricas se dividem em quatro categorias relativas aos quatro atributos internos. A separação de interesse é medida por **Difusão de Interesse em Componentes (CDC)**, **Difusão de Interesse em Operações (CDO)** e **Difusão de Interesse em Linhas de Código (CDLOC)**. O tamanho do sistema é medido em função do **Tamanho do Vocabulário (VS)**, **Número de Linhas de Código (LOC)**, **Número de Atributos (NOA)** e **Peso das Operações por Componentes (WOC)**. O grau de acoplamento entre os componentes é medido por **Acoplamento entre Componentes (CBC)** e **Profundidade da Árvore de Herança (DIT)**.

Finalmente, a coesão de cada módulo do sistema é medida pela **Perda de Coesão em Operações (LCOO)**. Métricas de software, incluindo os presentes neste modelo de qualidade, são abordadas na seção anterior deste documento.

3.4. Ferramentas de Desenvolvimento Orientado a Aspectos

No contexto de suporte automatizado para o DSOA, uma grande quantidade de ferramentas está disponível e pode ser encontrada na literatura [13] [15] [56] [58]. Entretanto, a maioria destas ferramentas é destinada à visualização [56] e *mineração*[9] [15] [58] de interesses transversais. Três ferramentas de suporte ao DSOA são brevemente apresentadas nesta seção: *Concern Manipulation Environment (CME)* [15], *Feature Exploration & Analysis Tool (FEAT)* [56] e uma ferramenta de medição [13]. As duas primeiras propõem formas de encontrar e visualizar elementos que implementam um interesse transversal específico.

3.4.1. Concern Manipulation Environment (CME)

O *Concern Manipulation Environment (CME)* [15] é na verdade um conjunto de ferramentas que promete um suporte ao DSOA através das várias fases do ciclo de vida do software. Este ambiente de programação é apoiado pela IBM [15] e permite que o desenvolvedor utilize diversas ferramentas e tecnologias simultaneamente. Inicialmente, o CME provê suporte a duas importantes tecnologias de DSOA: AspectJ [45] e separação multidimensional de interesses com Hyper/J [64]. Além destas, a proposta do ambiente é que este seja aberto para incorporar outras tecnologias e ferramentas.

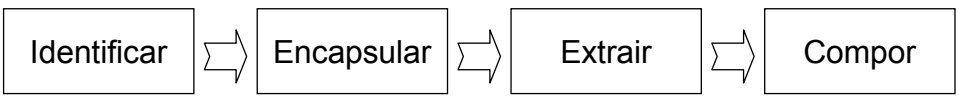

Figura 9 – Atividades do *Concern Manipulation Environment*

O CME trabalha para melhorar a separação de interesses em sistemas de software oferecendo suporte a quatro atividades que ajudam os desenvolvedores a amenizar problemas de código espalhado e entrelaçado. As duas primeiras

[9] Tradução para o termo *mining*.

atividades do CME são não-invasivas, ou seja, os desenvolvedores podem usá-las para adquirir melhor entendimento e organização do software sem ter que fazer uso explicitamente de DSOA. As duas últimas atividades são aplicações de técnicas do DSOA para extração e composição de interesses do sistema. Estas quatro atividades são apresentadas na Figura 9 e descritas a seguir:

1. **Identificação de interesses**: códigos existentes frequentemente envolvem uma variedade de interesses e muitos deles não foram adequadamente encapsulados em módulos. A atividade de identificação de interesses procura responder a questões como: "Quais partes do software pertencem a este ou aquele interesse?". Em CME, a identificação de um interesse envolve exploração dos artefatos usando uma combinação de navegação, consultas, análise e mineração.
2. **Encapsulamento de interesse**: uma vez que os elementos sintáticos do interesse tenham sido identificados, o desenvolvedor pode encapsular o interesse em uma estrutura lógica. Ou seja, os elementos sintáticos do interesse e seus relacionamentos são representados em um modelo do CME como elementos de primeira ordem.
3. **Extração de Interesse**: como mencionado na atividade anterior, o encapsulamento de um interesse resulta em uma separação lógica. Para muitos propósitos este grau de separação é suficiente, mas para outros o desenvolvedor pode querer sua separação física, extraindo o código em um artefato separado.
4. **Composição de interesses**: os artefatos dos interesses que foram fisicamente separados podem ser integrados (ou compostos) de diferentes formas. O resultado desta composição é um software que contenha as funcionalidades de todos os interesses envolvidos.

3.4.2. Feature Exploration & Analysis Tool (FEAT)

Feature Exploration & Analysis Tool (FEAT) [56] é uma ferramenta implementada como um *plugin* da plataforma Eclipse [66] que permite localizar, descrever e analisar o código que implementa um interesse em um sistema Java. O ambiente CME e a ferramenta FEAT se propõem a resolver problemas semelhantes do desenvolvimento de software. Ambos abordam a identificação dos

elementos de um determinado interesse e sua visualização de forma mais estruturada. Entretanto, utilizando a estrutura de navegação de FEAT pode-se localizar o código que implementa um interesse e salvá-lo em uma representação abstrata chamada *grafo de interesses*[10] [56]. Esta representação pode também ser usada para investigar os relacionamentos existentes entre o interesse capturado e o código base da aplicação.

O desenvolvedor cria um grafo de interesses pela aplicação de iterativas consultas no modelo do programa e determinando quais elementos e relacionamentos retornados pelas consultas contribuem para implementação do interesse desejado. FEAT disponibiliza um conjunto predefinido de consultas que se classificam em dois grupos:

- *Fan-in*: retorna todos os vértices no modelo do programa que depende do elemento selecionado.
- *Fan-out*: retorna todos os vértices ligados ao elemento selecionado através de arestas que saem deste elemento.

3.4.3. Uma Ferramenta de Medição

Ceccato e Tonella [13] argumentam sobre as diferenças existentes entre os desenvolvimentos OO e OA, especialmente no que se referem às novas formas de acoplamento introduzidas pelo paradigma de aspectos. Estes autores apresentam um conjunto de métricas e uma ferramenta para sua automatização. As métricas são basicamente extensões de métricas OO de Chidamber e Kemerer [14], exceto pelas de acoplamento que são revisadas para se adequar às novas abstrações do DSOA. A ferramenta proposta somente aplica medições em programas descritos na linguagem AspectJ [66] e a análise sintática do código é feita utilizando a linguagem de transformação de programas TXL [16].

A Figura 10 apresenta os cinco módulos que compõem a ferramenta de medição orientada a aspectos de Ceccato e Tonella. O primeiro módulo da ferramenta recebe como entrada toda classe, interface e aspecto do programa e faz uma engenharia reversa OO. Ou seja, detecta a estrutura dos componentes em termos de seus atributos, operações e relacionamentos de herança. Toda essa

[10] Tradução do inglês *Concern Graph.*

informação é armazenada em uma estrutura de dados. O segundo módulo faz a engenharia reversa mais avançada em que cada aspecto é processado para detecção de declarações intertipo. O resultado desta atividade também é armazenado na estrutura de dados utilizada pelo módulo anterior. O módulo seguinte da ferramenta detecta os relacionamentos de chamada de método e acesso a atributo. No quarto módulo é feita a resolução dos pontos de junção existentes no código dos aspectos, e então, os adendos são associados aos elementos interceptados pelos pontos de junção. Finalmente, o último módulo da ferramenta é responsável pela aplicação das métricas. O cálculo do valor de cada métrica é feito apenas pela aplicação de consultas à estrutura de dados.

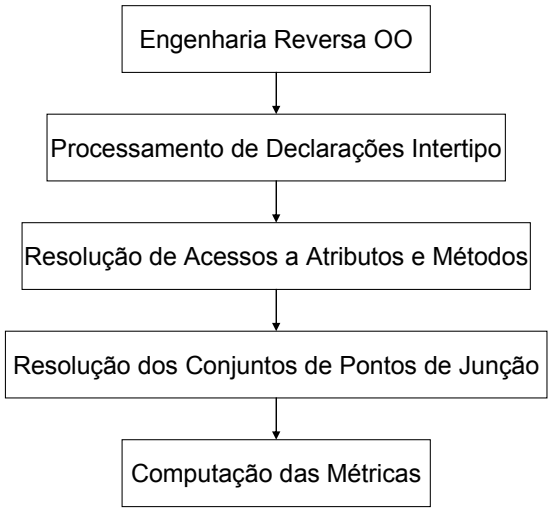

Figura 10 – Módulos da ferramenta de Ceccato e Tonella [13]

3.5. Limitações dos Trabalhos Relacionados

A qualidade de software deve ser acompanhada durante todo o processo de desenvolvimento. Nas seções 3.1 e 3.2 vimos que as abordagens de avaliação mais difundidas utilizam medições de atributos quantificáveis para predizer as características de qualidade dos artefatos. Sant'Anna *et al*. [57] propõem um *framework* de avaliação para desenvolvimento orientado a aspectos baseado em métricas (Subseção 3.3.1). Além do conjunto de métricas, seu *framework* possui um modelo de qualidade que mede graus de reusabilidade e manutenibilidade. Por outro lado, tal abordagem não inclui um conjunto de regras explícitas para auxiliar a interpretação dos resultados das medições e que possam ser automatizadas. Esta

ausência deixa uma lacuna entre o processo de medição e a interpretação dos valores. Além disso, Sant'Anna *et al*. [57] não apresentam um método sistemático para guiar o uso das métricas e uma forma de coordenar a seqüência de atividades para análise dos valores obtidos. Neste contexto, o trabalho apresentado nesta dissertação se caracteriza como uma extensão do trabalho de Sant'Anna *et al*. e tem o intuito de preencher lacunas deixadas pelo trabalho anterior.

Diversas ferramentas de suporte ao DSOA têm sido propostas e implementadas [13] [51] [56] [58], sendo três delas brevemente apresentadas na Seção 3.4. Duas destas ferramentas, CME e FEAT, têm como propósito auxiliar a identificação de elementos que implementam um determinado interesse. Este tipo de ferramenta pode ser considerado como complementar à abordagem desta dissertação, uma vez que elas tratam uma parte do problema não endereçado neste trabalho. A terceira ferramenta (Subseção 3.4.3), proposta por Ceccato e Tonella [13], aborda a medição de software orientado a aspectos. Essa ferramenta se asemelha ao módulo de medição da ferramenta proposta nesta dissertação (Capítulo 6). Porém, Ceccato e Tonella não apresentam um método organizado em etapas para interpretação dos valores e avaliação dos sistemas. Tal ferramenta se limita ao processo de medição, não dando suporte automatizado à identificação de eventuais problemas nas implementações sendo avaliadas. Este suporte pode ser dado pela definição e implementação de regras heurísticas que auxiliem a análise das medidas obtidas. Outra diferença importante entre a ferramenta de Ceccato e Tonella [13] e a proposta nesta dissertação diz respeito ao conjunto de métricas suportadas, por exemplo, medições de separação de interesses não são suportadas pela ferramenta daqueles autores.