3
Trabalhos Relacionados

O objetivo deste capítulo é situar o leitor quanto às características deste trabalho em relação ao que já está disponível na literatura, além de apontar possíveis vantagens e desvantagens comparativas. Para isto, os trabalhos relacionados foram classificados nas seguintes categorias: (i) métricas para diagramas de classes, (ii) detecção de problemas de *design* com base no código fonte e (iii) análise de *design* com base nos diagramas UML.

Na primeira categoria são apresentados trabalhos que procuram predizer a complexidade dos diagramas de classes a partir de um conjunto de métricas. Estes trabalhos se apóiam em diferentes perspectivas para avaliar a complexidade. (Genero & Piattini, 2001; Genero et al, 2002; Genero et al, 2003) utilizam um conjunto de 14 métricas, enquanto Zhou (2003) utiliza apenas uma métrica chamada de "entropia da distância".

Na segunda categoria estão os trabalhos relacionados a técnicas de detecção de problemas de *design* em código fonte. Além das estratégias de detecção propostas em (Lanza & Marinescu, 2006) e (Marinescu, 2002), que inspiraram este trabalho (Seção 3.2), essa categoria também inclui os trabalhos de Emden & Moonen (2002) e Munro (2005). Emden & Moonen definem uma abordagem para avaliar a qualidade dos *design* por meio da detecção de *code smells* em sistemas Java (2008). Munro apresenta duas estratégias para detectar os *bad smells*: *Lazy Class* e *Temporary Field* definidos em (Fowler et al, 1999).

Finalmente, na terceira categoria são apresentadas as ferramentas (SDMetrics, 2008) e (MetricView, 2005) que automatizam a análise da qualidade de um *design* utilizando diagramas UML. Além disso, são abordadas as características principais de ambas as ferramentas assim como suas vantagens e desvantagens em relação com a ferramenta proposta neste trabalho.

3.1.
Métricas para Diagramas de Classes

(Genero e Piattini, 2001; Genero et al 2002; Genero et al, 2003) apresentam um conjunto de métricas por eles definido para ser aplicado a diagramas UML, em particular diagramas de classes (Tabela 4). Para a definição deste conjunto, os autores se baseiam no fato de que as relações entre as classes constituem o fator-chave da complexidade dos diagramas de classes.

Tabela 4 – Métricas para diagramas de classes.

Nome da métrica	Descrição da métrica
Quantidade de relações por Associação (NAssoc)	Número total de associações entre classes.
Quantidade de relações por Agregação (NAgg)	Quantidade de relações parte-todo do tipo agregação entre as classes no diagrama.
Quantidade de Dependências (NDep)	Quantidade de relações de dependências entre as classes no diagrama.
Quantidade de Generalizações (NGen)	Quantidade de relações de generalização entre as classes no diagrama.
Quantidade de relações por Agregação entre Hierarquias (NAggH)	Quantidade total de relações de agregação entre classes pertencentes a diferentes hierarquias.
Maior profundidade na árvore de herança (MaxDIT)	Valor máximo do nível de profundidade na árvore de herança de cada classe do diagrama.
Quantidade de Classes (NC)	Quantidade total de classes no diagrama.
Quantidade de Atributos (NA)	Quantidade total de atributos no diagrama.
Quantidade de Métodos (NM)	Quantidade total de métodos no diagrama.
NAssocVC	Proporção entre a quantidade de associações e a quantidade de classes.
NAggVC	Proporção entre a quantidade de agregações e a quantidade de classes.
NGenVC	Proporção entre a quantidade de generalizações e a quantidade de classes.

Os autores se baseiam nas sugestões de Wohlin (Wohlin et al, 2000), Perry (Perry et al, 2000) e Briand (Briand et al, 1995) para desenvolver estudos experimentais a fim de avaliar a correlação entre as métricas propostas e o atributo da qualidade manutenibilidade. Para isto, foram examinados 27 diagramas de classes com diferentes graus de complexidade. Como resultado destes estudos, os autores concluíram que a maior parte das métricas propostas (NAssoc, NAgg,

NaggH, NGen, e MaxDIT) podem ser utilizadas como bons indicadores da manutenibilidade em diagramas de classes.

Analisando este conjunto de métricas, detectaram-se algumas desvantagens para a estimativa da complexidade. Primeiro, é difícil comparar as complexidades de diferentes diagramas de classes. Se os resultados de todas as métricas de um diagrama de classe são maiores do que os resultados obtidos em outro diagrama de classe, então, a comparação seria fácil. Porém, se alguns dos valores das métricas são maiores e outros menores, a tarefa de comparar a complexidade é dificultada. Além disso, as métricas, NC, NA, NM, NAssoc, NAgg, NDep, NGen, NGenH, NAggH, MaxDIT, e MaxHagg apresentam valores absolutos, enquanto NAssocVC, NAggVC e NGenVC são indicadores relativos. Se a quantidade e o tipo das relações entre as classes em dois diagramas fossem as mesmas, o diagrama de classe com mais classes teria baixo NAssocVC, NAggVC e NGenVC. Por causa disto, métricas relativas de complexidade não devem ser usadas para comparar a complexidade de dois diagramas de classes (Yi et al, 2004). Finalmente, pode ser mencionado que os autores atribuem o mesmo peso a todas as relações (composição, herança, dependência) entre as classes nos cálculos da correlação sem diferenciar o tipo.

Zhou (2003) define uma métrica chamada de: "entropia da distância entre os elementos". Esta métrica é utilizada como único indicador para determinar a complexidade de um diagrama de classes. Diferente dos trabalhos de Genero (Genero e Piattini, 2001; Genero et al 2002; Genero et al, 2003), são dados pesos às relações entre as classes no diagrama. Depois, cada diagrama de classes é transformado em um grafo de dependência ponderado utilizando um conjunto de regras. Para quaisquer dois diagramas de classes diferentes, os correspondentes grafos serão também diferentes, já que as relações entre os elementos não são as mesmas. Finalmente, a complexidade de um diagrama de classes é dada pela "entropia da distância" do grafo ponderado correspondente. Para o cálculo desta métrica o autor utiliza as fórmulas de "Entropia condicionada" $H(X|Y)$[2] e "Informação mútua" $I(X|Y)$[3] definidas em (Conver e Tomas, 1991). Neste

[2] A entropia condicionada é a medida da quantidade de informação necessária, em média, para descrever X dado que se conhece Y.

[3] A informação mútua mede a informação de Y descrita por X.

contexto, X e Y representam duas variáveis aleatórias que descrevem a probabilidade de cada nó do grafo ponderado ter arestas de entrada e saída adicionais com todos os nós do grafo.

A principal desvantagem desta métrica é que o cálculo é complexo já que, primeiramente, o grafo ponderado correspondente tem que ser construído além da própria aplicação da métrica sobre o grafo. É importante destacar que no trabalho o autor não apresenta um estudo empírico para validar o comportamento da métrica, e provar sua suposição de que as relações entre os elementos são o fator principal para estimar a complexidade de um diagrama de classes. Esse estudo foi feito anos depois por Tong & Fangjun (2004) em que as métricas foram avaliadas experimentalmente mediante a utilização de testes estatísticos. O estudo mostrou que existe uma correlação positiva entre a métrica definida e os diferentes fatores da manutenibilidade (capacidade de compreensão, facilidade de extensão e flexibilidade).

3.2. Detecção de Problemas de *Design* no Código Fonte

Emden & Moonen (2002) descrevem uma abordagem para a detecção e visualização automatizada de *code smells* para sistemas desenvolvidos na linguagem Java. Os *code smells* são definidos pelos autores como metáforas para descrever padrões que são geralmente associados a *design* incorreto e a más práticas de programação. Visando mostrar a usabilidade da abordagem proposta, os autores apresentam um protótipo por eles desenvolvidos, a ferramenta jCosmo, para automatizar a inspeção do código a fim de localizar práticas de programação incorretas. Finalmente, são apresentados os resultados práticos obtidos mediante a execução de um estudo de caso onde a ferramenta desenvolvida é utilizada. A principal limitação dessa abordagem é o fato de ser muito concentrada em questões de implementação, por exemplo: convenções de código, conversões de tipos e não nos problemas próprios de *design*.

Munro (2005) examina a utilização de métricas de software a fim de identificar *bad smells* em código fonte Java. Baseado em (Marinescu, 2002), Munro oferece duas estratégias para a localização e detecção dos seguintes *bad smells*: *Lazy Class* e *Temporary Field*, definidos em (Fowler, 2000). Visando

avaliar a precisão das estratégias de detecção propostas foram desenvolvidos: (i) um protótipo para a automatização deste mecanismo e (ii) um estudo de caso envolvendo dois sistemas pequenos. O estudo revelou a princípio, uma boa precisão das estratégias propostas para a detecção os *bad smells* tratados. Analisando este trabalho, podemos notar que existem algumas semelhanças com o proposto nesta dissertação. Primeiramente, ambos trabalhos são baseados em (Marinescu, 2002) para a definição de estratégias baseadas em métricas para a detecção de *bad smells*, em nosso caso aplicadas a modelos. Além disso, são realizados estudos com o objetivo de avaliar a precisão das estratégias propostas na detecção dos problemas de *design*. Porém, Munro (2005) possui a mesma limitação que Marinescu (2002) e que todos os outros trabalhos que se baseiam no código fonte para a detecção de problemas de *design*: corrigir os problemas de *design* depois de se investir na implementação do sistema pode ser caro e impraticável.

3.3. Análise de *Design* em Diagramas UML

SDMetrics (2008) é uma ferramenta desenvolvida com o propósito de permitir analisar as propriedades estruturais do *design* utilizando os diagramas UML (1999). Para isto, métricas são utilizadas para quantificar propriedades do *design* tais como: tamanho, acoplamento, e complexidade. Sendo assim, a ferramenta tem como objetivos principais: (i) estabelecer estratégias para identificar problemas potenciais de *design* nas primeiras etapas do ciclo de desenvolvimento do software, e, portanto, (ii) diminuir o esforço das etapas de implementação e teste.

A ferramenta recebe como entrada arquivos XMI (2000), os quais descrevem os modelos UML que representam o *design* dos sistemas a serem avaliados. Visando conseguir estender a aplicação da ferramenta a novas versões do formato XMI, foram definidos metamodelos e arquivos de transformação. Atualmente, a ferramenta apresenta compatibilidade com as versões XMI 1.0, 1.1, 1.2. 2.0 e 2.1.

Na análise do *design*, são usadas métricas que coletam informação proveniente de diferentes diagramas: classes, seqüência, atividade, estados, caso

de uso, e componentes. Além disso, regras definidas são utilizadas com o objetivo de detectar *design* incompleto, incorreto, redundante, dependências circulares, entre outras. Porém, estas regras apenas se concentram na estrutura dos diagramas. Outra característica importante de SDMetrics é seu modulo de exportação dos dados. Visando a usabilidade dos resultados das medições feitas, a ferramenta permite sua exportação a diferentes formatos: HTML, XML, Microsoft Excel.

Podemos notar algumas semelhanças entre SDMetrics e a ferramenta proposta neste trabalho, como a flexibilidade para a inclusão de novas versões de arquivos XMI e para a definição de novas métricas a serem utilizadas na análise do *design*. É importante destacar que, diferentemente da nossa ferramenta, SDMetrics não automatiza, mediante o uso de regras ou heurísticas, o processo de identificar componentes do *design* que possam influenciar negativamente a qualidade do sistema em etapas posteriores. Conseqüentemente, a ferramenta possui as mesmas limitações que abordagens baseadas na interpretação isolada dos resultados das métricas: não é capaz de indicar as causas das anomalias reportadas pelas métricas. Isso gera um impacto negativo na relevância dos resultados de medições. Além disso, a ferramenta não estima a qualidade do *design* utilizando modelos da qualidade já definidos e corretamente avaliados. Por causa disto, a qualidade do *design* não pode ser avaliada de acordo com o domínio do sistema que eles representam. Sistemas, por exemplo, que têm como objetivo maximizar a reusabilidade serão avaliados utilizando os mesmos mecanismos que aqueles cujo objetivo é maximizar a flexibilidade.

MetricView (2005) é uma ferramenta desenvolvida dentro do projeto EmpAnADa (*Empirical Analysis of Architecture and Design Quality*) da Universidade de Tecnologia de Eindhoven. Esta ferramenta automatiza a aplicação de métricas que utilizam de forma combinada informação proveniente de diversos diagramas: classes, seqüência, atividade e estado. Por exemplo, a *complexidade* de uma classe é medida em função da complexidade de seus métodos. Para isto, são utilizadas informações provenientes dos diagramas de estados e de classes. Os desenvolvedores se baseiam no fato de que as execuções dos métodos são responsáveis pelas mudanças de estados. Portanto, eles supõem que, à medida que os métodos são responsáveis por uma quantidade maior de mudanças de estados, tornam-se mais complexos. Entre as outras métricas

utilizadas encontram-se: quantidade de classes por caso de uso, quantidade de casos de uso por classe, quantidade de métodos e ausência de coesão entre métodos. Além da estimativa das propriedades do *design*, MetricView permite a detecção de inconsistências e incompletudes nos diagramas UML mediante o uso de regras.

MetricView apresenta os resultados das medições junto com os diagramas UML. Acreditamos que a forma de representar os resultados em cada uma das componentes permite aos usuários uma melhor localização dos possíveis problemas de *design*. Porém, com esta estratégia os engenheiros de software devem analisar cada um dos componentes e determinar, de acordo com os valores das métricas, se o componente apresenta ou não problemas. Esta tarefa pode ser trabalhosa, ainda mais se os sistemas analisados possuírem uma quantidade elevada de componentes. Além desta limitação, a ferramenta apresenta uma arquitetura fechada que não oferece nenhuma flexibilidade para a inserção de novas métricas. Além disso, a importação dos modelos é compatível apenas com o formato XMI gerado pela ferramenta Rational Rose (2008). Semelhante a SDMetrics, as técnicas de detecção de problemas definidas em MetricView se concentram na estrutura dos diagramas e a qualidade do *design* não é estimada utilizando modelos da qualidade já definidos e corretamente avaliados.