1
Introdução

Apesar de ser um mecanismo poderoso para avaliar e controlar a qualidade do *design* de software, métricas de software apresentam uma limitação intrínseca: é difícil obter interpretações adequadas a partir de medições de design (Marinescu, 2004). O valor de uma métrica pode indicar uma anomalia no código, mas deixa o desenvolvedor quase sempre sem indicação da causa da anomalia. Isso gera um impacto negativo na relevância dos resultados de medições (Marinescu, 2004; Lanza & Marinescu, 2006).

Para tratar dessa limitação das métricas, alguns pesquisadores propuseram mecanismos para formular regras baseadas em métricas que capturam desvios dos princípios e heurísticas de bom *design* orientado a objetos (Sahraoui et al, 2001; Martin, 2002, Emden & Moonen, 2002; Marinescu, 2004; Munro, 2005; Lanza & Marinescu, 2006). Marinescu (2004) chama esses mecanismos de estratégias de detecção. Estratégias de detecção ajudam os desenvolvedores a detectar e localizar problemas de *design* de um software ainda antes de se partir para a implementação. Uma estratégia de detecção é uma condição lógica composta por métricas que detecta elementos do *design* com problemas específicos. Por meio do uso de estratégias de detecção, o desenvolvedor pode localizar diretamente classes e métodos afetados por um problema de *design* particular, em vez de ter que inferir o problema a partir de um extenso conjunto de valores anormais de métricas. Este mecanismo vem sendo cada vez mais explorado. Recentemente foi publicado um livro sobre o assunto (Lanza & Marinescu, 2006). Além disso, foram desenvolvidas ferramentas, como (*InCode*, 2008), e ambientes de desenvolvimento, como o *Together* (2008) que já incorporam em suas características a automatização de algumas estratégias de detecção de anomalias de *design* OO recorrentes, tais como os chamados "*bad smells*" (Fowler, 1999).

A maioria dos trabalhos realizados até hoje propõem ou estudam estratégias de detecção e modelos da qualidade com base exclusivamente em informações extraídas do código fonte. No entanto, detectar e corrigir problemas de *design*

apenas depois de se investir na implementação do sistema pode ser caro e impraticável. Além disso, com o aumento da importância da engenharia de software dirigida por modelos (Atkinson & Kühne, 2003; Schmidt, 2006), a detecção de anomalias de *design* precisa ser realizada desde as representações de mais alto nível do sistema, como por exemplo, diagramas UML. Existem até ferramentas que automatizam a aplicação de métricas de *design* em diagramas UML (SDMetrics, 2008; MetricView, 2005). Porém elas não se dedicam a aplicação de estratégias de detecção nem de modelos da qualidade e sofrem das mesmas limitações das métricas de código.

É importante, portanto, definir estratégias de detecção que possam ser aplicadas a diagramas UML. Além disso, é preciso avaliar se essas estratégias são capazes de identificar os mesmos elementos problemáticos do *design* que as estratégias de detecção aplicadas ao código identificam. Nesse contexto, este trabalho propõe um conjunto de estratégias de detecção a serem aplicadas a diagramas de classes. Mais especificamente, foram definidas as versões para modelos das estratégias definidas por Lanza & Marinescu (2006) para detecção no código dos problemas de *design* chamados de: *God Class* (classes que tentam centralizar a funcionalidade do sistema) (Riel, 1996), *Data Class* (classes que fornecem mais dados que serviços), *Long Parameter List*, *Shotgun Surgery* (classes cuja modificação implica muitas pequenas modificações em muitas outras classes), *Misplaced Class* (classes definidas no pacote errado) e *God Package* (pacote grande, com muitas classes que não inter-relacionam-se entre si e que é utilizado por muitas outras classes) (Fowler, 1999). As estratégias de detecção utilizam métricas já existentes. Estas métricas são especializadas para serem aplicadas em diagramas de classes. Além disso, foi desenvolvida uma ferramenta com o objetivo de automatizar o controle da qualidade em modelos UML. Para isto, a ferramenta permite aplicação automatizada das estratégias propostas e modelos da qualidade. O motivo da escolha de juntar as estratégias de detecção com modelos da qualidade já disponíveis na literatura é que com sua aplicação conjunta se pode obter uma quantificação dos atributos da qualidade que são de interesse para os usuários e, ao mesmo tempo, identificar as entidades de *design* que influenciam negativamente nos resultados obtidos destes atributos.

A ferramenta desenvolvida foi usada no contexto de dois estudos experimentais de diferentes domínios, com características, níveis de complexidade

distintos. Esses estudos serviram como avaliações iniciais da utilidade e usabilidade da ferramenta desenvolvida. O primeiro estudo experimental avalia as estratégias de detecção propostas para modelo, no contexto de nove sistemas, de acordo com o grau de acurácia, precisão e *recall* com que elas identificam as mesmas classes com problemas de *design* que suas correspondentes para código. O segundo estudo envolveu a utilização da ferramenta para aplicar o modelo da qualidade QMOOD (Bansiya & Davis, 2002) em uma seqüência de versões do framework JHotdraw (2006) com o objetivo de avaliar a utilidade do modelo QMOOD utilizando diagramas UML.

O restante deste documento está organizado da seguinte forma. O capítulo 2 dá uma visão geral sobre características de bom *design* OO e medição de software. O capítulo 3 apresenta um estudo da literatura relacionada a este trabalho. O capítulo 4 apresenta as estratégias de detecção propostas para modelos, assim como as que deram origem a elas, suas correspondentes estratégias para código fonte. O capítulo 5 mostra em detalhes a arquitetura e implementação da ferramenta desenvolvida para apoiar a análise da qualidade em modelos UML. Os dois estudos experimentais realizados com o uso da ferramenta são descritos no capítulo 6. Finalmente, o capítulo 7 apresenta algumas conclusões e possíveis trabalhos futuros. Os Anexos A, B e C mostram, respectivamente, a estrutura dos arquivos XML gerados pela ferramenta desenvolvida neste trabalho, o arquivo gerado para armazenar os resultados obtidos após a aplicação das estratégias de detecção e os dados obtidos no primeiro estudo experimental.