Um Método para Identificação de Bad Smells a partir de Diagramas de Classes

Henrique Gomes Nunes[1], **Mariza A. S. Bigonha**[1],
Kecia A. M. Ferreira[2], **Flávio Airjan Madureira**[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais

[2]Departamento de Computação – Centro Federal de Educação Tecnológica de Minas Gerais

{henrique.mg.bh,airjanmadureira}@gmail.com, mariza@dcc.ufmg.br

kecia@decom.cefetmg.br

Abstract. *Este trabalho propõe um método para a identificação de bad smells, via métricas de software e seus respectivos valores referência, em sistemas orientados por objetos a partir de diagramas de classes. O método proposto foi avaliado em dois estudos: um com o objetivo de analisar os resultados do método aplicado a versões antigas e versões refatoradas de um conjunto de seis sistemas de software abertos; e outro para comparar os resultados do método com a análise manual. Os resultados sugerem que o método proposto mostra-se útil para a identificação dos bad smells considerados neste trabalho.*

Resumo. *This work defines a method and a tool to identify bad smells, using software metrics, in class diagrams. We carried out two studies to evaluate the proposed method: the first one aimed to evaluate the results of our method when applied to old versions as well as to refactored versions of six open source projects; in the second study, we compared the results of our method with the results of manual inspections. The results suggest that our method is able to identify the bad smells analyzed in this work.*

1. Introdução

Para Fowler [Fowler 1999], *bad smell* é um indicador de um possível problema estrutural em código-fonte, que pode ser melhorado via refatoração. Com o objetivo de contribuir para aprimorar a qualidade dos sistemas de software, alguns trabalhos têm sido desenvolvidos para identificar *bad smells* em software com o uso de métricas a partir de código fonte [Marinescu 2002, Lanza et al. 2006]. Todavia, é importante que problemas sejam identificados o mais cedo possível no ciclo de vida de um software como por exemplo na fase de modelagem, para que os problemas nas fases posteriores do desenvolvimento de software sejam reduzidos. Para isso, ser capaz de obter métricas a partir de modelos da UML é essencial [Soliman et al. 2010].

Dentre os diagramas definidos na UML, destaca-se o diagrama de classes, que representa classes de objetos e suas relações. A análise desse diagrama pode fornecer métricas relacionadas à manutenibilidade e complexidade do software logo no início do projeto. Porém, um projeto real necessita de uma ferramenta para automatizar este processo, pois é inviável calcular tais métricas manualmente.

O presente trabalho visa contribuir com a solução para o problema de identificar problemas estruturais em software a partir de modelos UML, em particular, a partir dos diagramas de classes. Para atingir esse objetivo, neste trabalho (1) foi definido um método de identificação de *bad smells* em sistemas de software a partir de diagrama de classes, aplicando métricas de software e seus valores referência e, (2) foi desenvolvida uma ferramenta denominada *UMLsmell* para permitir automatizar a coleta de métricas e a aplicação delas na identificação dos *bad smells* a partir de diagramas de classes. O método e a ferramenta propostos foram avaliados por meio de experimentos com sistemas de software abertos.

2. Método Proposto

Para a definição do método proposto, inicialmente foram identificados, dentre os *bad smells* descritos na literatura, aqueles que podem ser aplicados a modelos de classe da UML. Foram selecionados três deles:

- *God Class*: corresponde a classes com alta responsabilidade no sistema. Esse *bad smell* está associado às métricas que permitem avaliar se uma classe possui muitos métodos e se muitas classes dependem da classe avaliada.
- *Indecent Exposure*: ocorre quando uma classe está mal encapsulada. Esse *bad smell* está associado às métricas de atributos públicos que permitem avaliar o encapsulamento das classes.
- *Shotgun Surgery*: refere-se a classes que ao serem alteradas causam impacto em outras classes. Este *bad smell* está associado às métricas de relacionamentos do tipo aferentes que permitem avaliar se a alteração em uma classe implicará em alterações em outras classes.

2.1. Valores Referência

Neste trabalho foram utilizadas as seguintes métricas: Número de Conexões Aferentes (NCA), Número de Métodos Públicos (NMP) e Número de Atributos Públicos (NAP). A razão para a escolha dessas métricas é que elas são aplicáveis a diagramas de classes e possuem valores de referência previamente definidos na literatura (Tabela 1). As métricas NCA, NMP e NAP permitem identificar os *bad smells God Class*, *Shotgun Surgery* e *Indecent Exposure*.

Os valores referência usados foram definidos por Ferreira et al. [Ferreira et al. 2012]. A principal razão da escolha desses valores referência é que eles foram avaliados empiricamente anteriormente. Ferreira et al. [Ferreira et al. 2012] definiram valores referência para estas métricas e os classificaram como: *bom*, representando os valores mais frequentes para métricas em sistemas de software de boa qualidade; *regular*, correspondendo a valores pouco frequentes para métricas em sistemas de software de boa qualidade; *ruim*, correspondendo a valores raros para métricas em sistemas de software de boa qualidade. A idéia das faixas sugeridas como valores referência por Ferreira et al. é que uma vez que os valores são frequentes em sistemas de software, isso indica que eles correspondem à prática comum no desenvolvimento de software de alta qualidade, o que serve como um parâmetro de comparação de um software com os demais. Da mesma forma, os valores pouco frequentes indicam situações não usuais na prática, portanto, pontos a serem considerados como críticos.

Métrica	*Bom*	*Regular*	*Ruim*
NCA	até 1	2 a 20	superior a 20
NAP	0	1 a 10	superior a 10
NMP	até 10	11 a 40	superior a 40

Table 1. Valores referência definidos por Ferreira et al. [Ferreira et al. 2012].

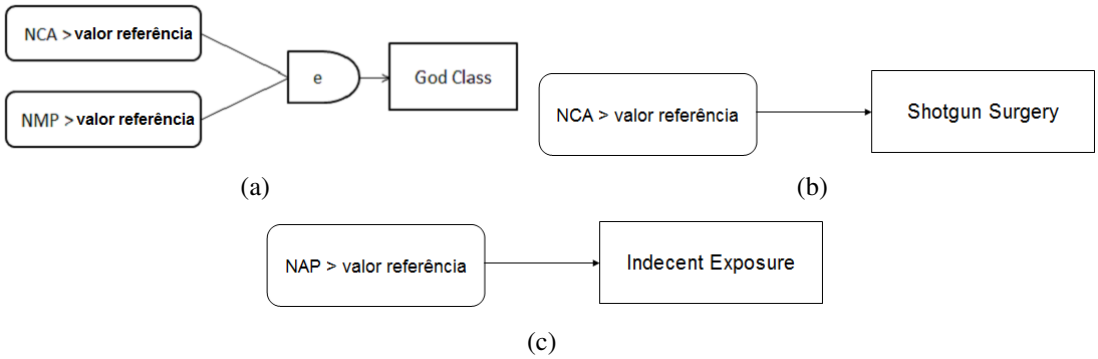

Figure 1. Estratégia de detecção para: (a) *God Class*, (b) *Shotgun Surgery* e (c) *Indecent Exposure*.

Neste trabalho, optou-se por considerar as faixas *regular* e *ruim* dos valores referência propostos por Ferreira et al. [Ferreira et al. 2012] na identificação de *bad smells* porque tais faixas correspondem a situações não usuais na prática de desenvolvimento de software.

2.2. Estratégias de Detecção

As estratégias de detecção propostas são apresentadas na Figura 1. As métricas que definem a estratégia de detecção do *bad smell God Class* consideram seus relacionamentos com outras classes e o tamanho da classe avaliada. O *God Class* é identificado via métricas NCA e NMP. A métrica NCA verifica a influência da classe avaliada sobre as outras classes do software, ou seja, a quantidade de classes que usam os serviços da classe avaliada. A métrica NMP permite avaliar o tamanho de uma classe e a quantidade de serviços a oferecer pela quantidade de métodos. Caso as duas métricas avaliadas estejam dentro da faixa *regular* ou *ruim* de acordo com os valores referência, o *God Class* é considerado como crítico. Caso contrário, a classe não é considerada como problemática.

O *Shotgun Surgery* é identificado via métrica NCA, que define quantas classes dependem da classe avaliada, uma vez que quando uma classe é alterada, todas as outras classes que dependem dela estão sujeitas a mudanças. O *Indecent Exposure* é um *bad smell* referente à falta de encapsulamento de classes. Um bom encapsulamento acontece quando os dados, atributos, de uma classe são ocultos e seus serviços, métodos, úteis para as demais classes, são públicos [Sommerville 2007]. Ele pode ser identificado via métrica NAP, que define quantos atributos de uma classe são públicos. A identificação do *Indecent Exposure* em uma classe e em qual faixa se encontra, *regular* ou *ruim*, é equivalente à métrica avaliada.

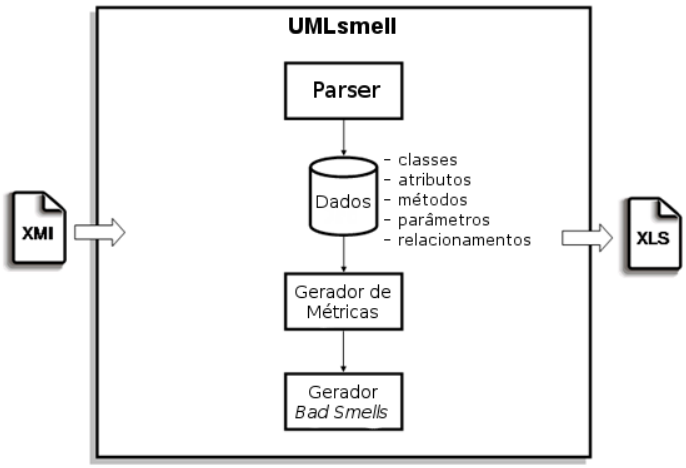

Figure 2. Estrutura da UMLsmell.

2.3. Ferramenta

A estrutura da ferramenta *UMLsmell* é mostrada na Figura 2. A ferramenta recebe como entrada um arquivo XMI, no qual é realizada a análise sintática com o objetivo de extrair as seguintes informações: classes, atributos, métodos, parâmetros e relacionamentos. Esses dados são armazenados internamente nas estruturas de dados definidas pelo programa para consultas posteriores, tal que, a partir desses dados é possível fornecer as informações de métricas das classes do software ao usuário. Essas métricas são, então, aplicadas para identificar *bad smells* conforme o método proposto neste trabalho. A ferramenta foi desenvolvida na linguagem Java e possui 3205 linhas de código.

3. Avaliação

Esta seção relata os experimentos realizados para avaliar o método proposto e os seus respectivos resultados. O objetivo desses experimentos foi avaliar o método para identificação de *bad smells* a partir de diagramas de classes.

3.1. Metodologia

Os experimentos investigaram as seguintes questões de pesquisa:

RQ1: as métricas de software e seus respectivos valores referência auxiliam a identificar *bad smells* em um software?

RQ2: os *bad smells* identificados pelo método em um diagrama de classes de um software são correspondentes àqueles identificados por avaliadores com formação em Engenharia de Software?

Para responder essas questões foram realizados dois conjuntos de experimentos:

- Análise de diagramas de classes de sistemas de software reestruturados: esse experimento avalia a identificação automática de *bad smells* em duas versões de cada diagrama, tal que uma versão é a refatoração da outra. Tendo em vista que uma refatoração é uma modificação realizada no software para melhorar sua estrutura,

espera-se encontrar mais *bad smells* na versão não refatorada, indicando que o método proposto é eficaz no reconhecimento de *bad smells* automaticamente.
- Avaliação Manual: este experimento avalia manualmente a identificação de *bad smells* e os compara com os resultados da *UMLsmell* a fim de avaliar a eficácia da estratégia de detecção proposta.

3.2. Diagramas de Classes dos Experimentos

Foram realizados experimentos a partir dos diagramas de classes dos seguintes sistemas de software abertos desenvolvidos em Java: *JHotdraw*, *Struts*, *HSqlDB*, *JSLP*, *Log4J* e *Sweethome 3D*. A seleção desses sistemas de software se deu por eles serem considerados como refatorados, conforme relatado pelos autores Dig et al. [Dig and Johnson 2005] e Soares et al. [Soares et al. 2011]. Isso significa que os desenvolvedores do software, a cada versão lançada, se preocuparam em melhorar a arquitetura desses sistemas de software. Para a escolha dos sistemas de software da avaliação manual, um conjunto de sistemas de software pequenos foram escolhidos no repositório Github e foram analisados por *UMLsmell*, para determinar quais deles possuiam mais *bad smells*. Nesse processo foram escolhidos o *Picasso* e *JSLP*, pois o fato desses sistemas de software serem pequenos viabilizou a inspeção manual pelos avaliadores.

Devido à carência de diagramas de classes para realizar os experimentos, os diagramas dos sistemas de software utilizados para esta avaliação foram obtidos a partir de engenharia reversa usando a ferramenta Enterprise Architect[1] para ler os códigos fonte e gerar os diagramas de classes.

Como a ferramenta Enterprise Architect não é capaz de criar todos os relacionamentos existentes no código fonte, utilizou-se o software Dependency Finder[2] que a partir do *bytecode* de um software é capaz de gerar um arquivo XML contendo os relacionamentos de dependência para cada classe do software. Para gerar o diagrama de classes completo automaticamente, a partir das informações obtidas na engenharia reversa do Enterprise Architect e do resultado reportado pelo Dependency Finder, neste trabalho foi desenvolvida uma ferramenta para criar automaticamente todos os relacionamentos de dependência existentes no código fonte. Essa ferramenta intermediária foi desenvolvida na linguagem Java e recebe como entrada o arquivo XMI do Enterprise Architect e o arquivo XML do Dependency Finder. A ferramenta realiza uma análise sintática do arquivo XML e os relacionamentos são criados no arquivo XMI da engenharia reversa feita pelo Enterprise Architect. Com isso, obteve-se um arquivo XMI resultante que contém os dados de todas as classes do software e dos relacionamentos existentes entre elas.

4. Resultados

Em praticamente todos os sistemas de software foram identificadas ao menos uma melhoria de *bad smells* de uma versão para outra refatorada. Esse resultado indica a eficácia do método proposto, visto que foram encontrados mais falhas na versão não refatorada dos sistemas de software avaliados. Essas melhorias ficaram evidentes nos sistemas de software *Jhotdraw* e *Sweethome 3D*. Nesses dois sistemas de software, a quantidade de *bad smells* foi muito superior na versão não refatorada do que na versão refatorada, concordando com os estudos de Dig et al. [Dig and Johnson 2005] e Soares et al.

[1] www.sparxsystems.com.au
[2] depfind.sourceforge.net

[Soares et al. 2011]. Nesses sistemas de software foram constatados que principalmente nos *bad smells Shotgun Surgery* e *God Class* a quantidade de classes que apresentavam esses problemas diminuiu, apontando que a estrutura do sistema realmente passou por melhorias. No entanto, nesses três casos, não se pode dizer o mesmo em relação ao *Indecent Exposure*, que em alguns casos a quantidade de classes que apresentavam esse *bad smell* foi mantida ou aumentou da versão não refatorada para versão refatorada. Tal fato pode ter ocorrido porque esse *bad smell* pode não ter sido alvo de atenções durante as refatorações.

Também foram observadas melhorias mais discretas nos sistemas de software *Struts*, *Log4j*, *HSqlDB* e *JSLP*. Neles, nos três *bad smells* avaliados, ocorreram menos melhorias da versão não refatorada para a versão refatorada, sendo que em alguns casos ocorreu aumento da quantidade de *bad smells*. Mas, ainda assim, foi possível detectar que a refatoração ocorreu em alguns casos nos quais os *bad smells* reduziram, por exemplo, o *Indecent Exposure* no software *Struts*.

A avaliação manual foi realizada por 15 especialistas com conhecimento em Engenharia de Software que não tinham informações da metodologia utilizada no estudo realizado. Foram entregues dois diagramas de classes aos especialistas e a descrição textual de cada *bad smell*, afim de que os especialistas identificassem quais classes possuiam quais problemas. O experimento identificou resultados muito próximos dos apresentados pela *UMLsmell*. Para os *bad smells Shotgun Surgery* e *Indecent Exposure*, as classes identificadas como problemáticas pela *UMLsmell* foram as classes com maior frequência de identificação pelos avaliadores. Das classes em que *UMLsmell* acusou a presença do *bad smell God Class* a maioria também foi identificada com esses *bad smells* pelos avaliadores. Quatro classes identificadas como problemática pelos avaliadores, não foram identificadas como problemáticas pela *UMLsmell*. Esses resultados indicam que as estratégias de detecção definidas neste estudo, estão próximas dos resultados dos avaliadores. No caso de *God Class*, houve uma diferença maior entre os resultados do método proposto e a análise manual. Uma explicação possível para isso é que os avaliadores provavelmente consideraram apenas o tamanho das classes, enquanto a estratégia de detecção do método proposto considera também a quantidade de classes que utilizam a classe avaliada, porém tal afirmação só poderia ser comprovada entrevistando os especialistas após o experimento manual.

Os resultados dos experimentos realizados em sistemas de software tidos como refatorados mostram que em todos os sistemas de software foram identificadas ao menos uma melhoria de *bad smells* de uma versão para outra refatorada. Esse resultado indica que o método proposto é capaz de detectar automaticamente os *bad smells* considerados neste trabalho, visto que foram encontradas mais falhas na versão não refatorada dos sistemas de software avaliados. Com isso, conclui-se que a resposta para RQ1 é positiva. Da mesma forma, os resultados da avaliação manual se mostraram muito próximos daqueles produzidos por *UMLsmell*, o que indica que a resposta para a RQ2 também é positiva.

5. Trabalhos Relacionados

Os trabalhos de Marinescu [Marinescu 2002, Marinescu 2004] são os mais próximos do presente trabalho. Eles apresentam uma solução para transformar métricas em informações relevantes para detectar *bad smells* e avaliam a proposta a partir de um estudo de

caso com um software proprietário de pequeno porte. Bertrán [Bertrán 2009] define estratégias de detecção a serem aplicadas a diagramas de classe, mas evidencia a importância de redefinir os valores referência utilizados em seus estudos experimentais, definidos por Marinescu [Marinescu 2002], pois considera que a técnica utilizada por Marinescu [Marinescu 2002] não seja a melhor forma de definí-los.

As principais contribuições do presente trabalho em relação aos trabalhos prévios é que ele: utiliza valores referência definidos e avaliados na literatura para as estratégias de detecção; desenvolveu as ferramentas adequadas para a aplicação do método; avaliou a estratégia proposta com uma quantidade maior de programas, de maior porte e livres, o que faculta a replicação do estudo.

6. Limitações

A principal limitação deste trabalho é a pouca quantidade de métricas que têm valores referência definidos na literatura. Isso limita a quantidade de *bad-smells* que puderam ser avaliados neste trabalho. Outra limitação importante é que o ideal seria a utilização de diagramas de classes para a realização dos estudos de avaliação do método proposto. Todavia, é difícil obter esse tipo de diagrama em projetos abertos e, em geral, há restrições por parte das organizações em fornecer qualquer dado sobre seus sistemas de software proprietários. Devido a isso, os diagramas foram obtidos via engenharia reversa a partir do código fonte de sistemas de software abertos. A utilização de valores referência obtidos a partir de código fonte e não de diagramas de classes também pode influenciar na ocorrência de mais resultados do tipo falso positivo, pois diagramas de classes são mais abstratos que códigos fonte. Apesar dessas limitações, os resultados da avaliação do método proposto indicam que ele é capaz de auxiliar, de forma automática, a identificação de *bad smells* em software orientado por objetos a partir de diagrama de classes, utilizando, para isso, métricas de software e seus respectivos valores referência.

7. Conclusão

Neste trabalho foi proposto um método para permitir identificar desvios de projeto de software nas fases iniciais do ciclo de vida do sistema. O método proposto se baseia em extrair métricas de diagramas de classes e usar os valores referência propostos na literatura para tais métricas para identificar *bad smells*. Foram identificados os *bad smell* que podem ser extraídos de diagramas de classes e, então, foram identificadas na literatura as principais métricas que podem ser extraídas de diagramas de classes. Em relação às métricas, duas características foram levadas em consideração: primeiro, as métricas deveriam possuir valores referência propostos na literatura e segundo, as métricas deveriam representar melhor os *bad smells* que podem ser identificados em diagramas de classes. Os valores referência usados para relacionar as métricas aos *bad smells Shotgun Surgery*, *God Class* e *Indecent Exposure* foram definidos por Ferreira et al. [Ferreira et al. 2012]. Também foi projetada e implementada uma ferramenta, denominada *UMLsmell*, que permite aplicar o método proposto. A ferramenta identifica automaticamente *bad smells* em diagramas de classes.

Para avaliar o método, foram conduzidos dois estudos. O primeiro foi realizado com o objetivo de verificar se o método proposto identifica os *bad smells* considerados neste trabalho, a partir da comparação entre versões de sistemas de software que, de

acordo com relatos na literatura, passaram por refatorações. O segundo estudo teve por objetivo comparar os resultados reportados pelo método e a avaliação manual. Os resultados dos estudos sugerem que o método proposto auxilia a identificar automaticamente a presença de *bad smells* em diagramas de classes.

Os seguintes trabalhos futuros são identificados a partir dos resultados do presente trabalho: a identificação de valores referência para outras métricas, que será importante para viabilizar a detecção de outros *bad smells* com o método proposto; avaliar o uso de outras métricas para avaliar os *bad smells* considerados neste trabalho a fim de refinar as estratégias de detecção sugeridas; estender a ferramenta proposta para viabilizar a identificação de outros *bad smells*.

References

Bertrán, I. M. (2009). Avaliação da qualidade de software com base em modelos uml. Dissertação da PUC-RJ. Rio de Janeiro, RJ, Brasil. Pontifícia Universidade Catolica do Rio de Janeiro.

Dig, D. and Johnson, R. (2005). The role of refactorings in api evolution. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 389–398. IEEE.

Ferreira, K. A., Bigonha, M. A., Bigonha, R. S., Mendes, L. F., and Almeida, H. C. (2012). Identifying thresholds for object-oriented software metrics. *Journal of Systems and Software*, 85(2):244–257.

Fowler, M. (1999). *Refactoring: improving the design of existing code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Lanza, M., Marinescu, R., and Ducasse, S. (2006). *Object-Oriented Metrics in Practice*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Marinescu, R. (2002). In *Measurement and Quality in Object-Oriented Design*, Tese de Doutorado. Timisoara, Romênia. University of Timisoara.

Marinescu, R. (2004). Detection strategies: metrics-based rules for detecting design flaws. In *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*, pages 350 – 359.

Soares, G., Catao, B., Varjao, C., Aguiar, S., Gheyi, R., and Massoni, T. (2011). Analyzing refactorings on software repositories. In *Software Engineering (SBES), 2011 25th Brazilian Symposium on*, pages 164–173. IEEE.

Soliman, T., El-Swesy, A., and Ahmed, S. (2010). Utilizing ck metrics suite to uml models: A case study of microarray midas software. In *Informatics and Systems (INFOS), 2010 The 7th International Conference on*, pages 1 –6.

Sommerville, I. (2007). *Engenharia de Software 8a ed.* São Paulo: Pearson Addison Wesley.