Identificação de Bad Smells em Softwares a partir de Modelos UML

Henrique G. Nunes[1], Mariza A. S. Bigonha[1], Kecia Aline Marques Ferreira[2]

[1] Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG) - Belo Horizonte – MG – Brasil

[2] Departamento de Computação– Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG) - Belo Horizonte – MG – Brasil.

[3]Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brazil

henrique.mg.bh@gmail.com, mariza@dcc.ufmg.br, keciamarques@gmail.com

Resumo. *Métricas de software podem auxiliar na identificação de desvios de projeto, conhecidos na literatura como bad smells. Embora vários trabalhos tenham sido desenvolvidos para identificar bad smells via a utilização de métricas, esses trabalhos são restritos por que os mesmos reportam apenas o uso de métricas. O trabalho de dissertação apresentado neste artigo visa contribuir nesse aspecto, propondo método e ferramentas para identificação de bad smells, via métricas de software, em sistemas orientado por objetos a partir de modelos UML.*

1. Caracterização do Problema

Assim como nos códigos-fonte, as métricas obtidas a partir de modelos *UML* podem permitir analisar confiabilidade, manutenibilidade e complexidade de um sistema. A *UML* surgiu com a proposta de padronizar questões relacionadas ao projeto orientado por objetos, permitindo assim que um sistema fosse representado independente da linguagem de programação utilizada. Dentre os seus diagramas, destaca-se o diagrama de classes que representa classes de objetos e suas relações - Yi et al. (2004). A análise desse diagrama pode nos fornecer diversas métricas relacionadas à manutenibilidade e complexidade do software logo no início do projeto, como o número total de classes, métodos e relacionamentos, por exemplo. Porém, em um projeto real é inviável calcular tais métricas manualmente - Girgis et al. (2009).

Diante desse problema, evidencia-se a importância de identificar problemas arquiteturais nas fases iniciais do projeto de software. Nesse sentido, o objetivo de minha dissertação consiste em definir um método de identificação de *Bad Smells* em softwares a partir de modelos *UML*. Para isso, especialistas avaliarão um conjunto de diagramas de classes e apontarão classes com *Bad Smells* pré-definidos. Em seguida, uma ferramenta desenvolvida durante o projeto de dissertação será utilizada para coletar métricas e identificar automaticamente classes com os mesmos *Bad Smells*. Por fim, os resultados dos especialistas e da ferramenta serão comparados. Por não possuir diagramas de classes disponíveis para os estudos, os mesmos foram obtidos através de engenharia reversa, simplesmente para avaliar a efetividade da metodologia proposta. Até o momento, apenas um experimento preliminar foi realizado.

2. Fundamentação Teórica

Devido à limitação de espaço desse artigo, citaremos sucintamente os principais termos relacionados ao trabalho da dissertação e suas principais referências.

2.1. Métricas

Sommerville (2011) define métrica como um valor numérico relacionado a algum atributo de software, permitindo a comparação entre atributos da mesma natureza. Ele argumenta que a medição nos permite realizar previsões gerais de um sistema e identificar componentes com *Bad Smells*. De acordo com o mesmo autor, uma métrica é qualquer tipo de medição que faça referência a um sistema, processo ou documento, por exemplo, a quantidade de linhas de códigos de um programa. Há também métricas propostas especificamente para análise de modelos *UML* - Yi et al. (2004). Estas métricas, assim como as métricas orientadas por objetos, tratam de métodos, atributos, herança, relacionamentos, dentre outros elementos orientados por objetos.

2.2. Bad Smell

Fowler (1999) define o termo *Bad Smell* como indicador de possível problema em código fonte, que pode ser melhorado através de refatoração.

Fowler (1999) e Marinescu (2002) definem *Bad Smells* para códigos-fonte. O trabalho dessa dissertação propõe identificar *Bad Smells* em *UML*, portanto primeiramente foi necessário criar uma lista com *Bad Smells* mais comuns na literatura e avaliar quais são possíveis de extrair de diagramas de classes.

Em seguida, para um experimento preliminar, utilizamos métricas e valores referência definidos em Ferreira et al. (2012) para identificar *Bad Smells*. Com tais métricas foi possível avaliar os seguintes *Bad Smells*:

- ***God Class***: Classes que tendem a centralizar a inteligência do sistema – Marinescu (2002).
- ***Shotgun Surgery***: A mudança em uma classe, gera pequenas mudanças em muitas outras classes – Marinescu (2002).
- ***Indecent Exposure***: Quando uma classe revela dados internos - Hamza (2008).

Caso o experimento ocorra conforme o esperado, iremos avaliar mais métricas e ***Bad Smells***.

2.3. Estratégia de Detecção

Marinescu (2004) define Estratégia de Detecção como *"uma expressão quantificável de uma regra, que permite avaliar se fragmentos de código estão de acordo com essa regra"*. Para definir formalmente alguns *Bad Smells*, Marinescu (2004) e Lanza & Marinescu (2006) utilizam técnicas das Estratégias de Detecção como mecanismos de filtragem e composição, que contemplam dentre outras coisas métricas e valores referência.

Bertrán (2009) afirma que por meio da adaptação de Estratégias de Detecção definidas por Marinescu (2004), projetistas podem localizar diretamente classes e

métodos em modelos *UML* afetados por um *Bad Smell* particular, ao invés de deduzir o problema a partir de um extenso conjunto de valores anormais de métricas.

3. Trabalhos Relacionados

Há na literatura algumas ferramentas que têm como objetivo fornecer dados sobre um projeto de software por meio do uso de métricas de modelos *UML*, dentre as quais destacam-se as propostas por: Girgis et al. (2009), Soliman et al. (2010), Nuthakki et al. (2011) que realizam coleta de métricas em diagramas de classes. Porém, essas ferramentas limitam-se à coleta de métricas.

Marinescu (2002) realizou um dos principais trabalhos sobre estratégias de detecção de *Bad Smells* em *software* orientado por objetos a partir de métricas de software. As estratégias definidas por ele se baseiam em definir valores limites para as métricas utilizadas na detecção de *Bad Smells*. Porém, ele não define precisamente os valores limites a serem considerados.

Bertrán (2009) utiliza métricas de software para definir regras que identifiquem *Bad Smells* em diagramas de classes. Todavia, da mesma forma que Marinescu (2002), não são definidos, de forma sistemática, valores referência das métricas para a identificação dos *Bad Smells*. O trabalho de Bertrán (2009) explora a identificação de problemas estruturais em *software* a partir de modelos *UML*, todavia é restrito na quantidade de métricas em que se baseia.

A definição de valores referência de métricas de software é uma lacuna atual. Ferreira et al. (2012) propõem um método para identificação de tais valores e identificam valores referência para seis métricas de *software* orientado por objetos. Os valores foram avaliados na identificação de problemas estruturais em código fonte de programas Java.

O trabalho proposto nessa dissertação visa propor melhorias nos métodos de identificação de problemas arquiteturais em software orientado por objetos a partir de modelos *UML*. Para isso, será definido um método baseado em métricas e seus valores referência. A principal diferença entre a abordagem adotada no presente trabalho e a de trabalhos anteriores é que iremos avaliar novas possíveis métricas para representar os *Bad Smells* definidos por Fowler (1999) e não nos limitando ao que é definido em Marinescu (2002) e Bertrán (2009). Outra diferença será a escolha de métricas com valores referência definidos na literatura, ao invés de utilizar valores arbitrários. A forma de avaliação também é diferenciada, onde ao invés de compararmos os resultados com códigos-fonte, a avaliação será feita por especialistas e posteriormente comparada com os resultados gerados automaticamente por uma ferramenta desenvolvida durante a dissertação.

4. Estado Atual do Trabalho

A presente dissertação acaba de finalizar um experimento preliminar. Para realizar o experimento, primeiramente pesquisamos um conjunto de *Bad Smells* que poderiam ser identificados em diagramas de classes. Inicialmente, optamos por utiliza as métricas definidas por Ferreira et al. (2012) para representar alguns desses *Bad Smells*, mas posteriormente, nos experimentos que virão, pretendemos ampliar o conjunto de

métricas e *Bad Smells*. As métricas utilizadas e os *Bad Smells* que elas representam estão na Tabela 1.

Tabela 1. Bad Smells e métricas utilizados nos experimentos preliminares

Bad Smell	Referência	Métricas	Valores Referência
God Class (Large Class)	Fowler (1999) Marinescu (2002)	Número de métodos públicos (NMP)	médio: 11 >= 40 ruim: > 40
		Número de conexões aferentes (NCA)	médio: 2 >= 20 ruim: >20
Shotgun Surgery	Marinescu (2002)	Número de conexões aferentes (NCA)	médio: 2 >= 20 ruim: >20
Indecent Exposure	Hamza (2008)	Número de atributos públicos (NAP)	médio: 1 >= 10 ruim: >10

As hipóteses avaliadas no experimento preliminar foram:

H.0 Os resultados obtidos automaticamente pela ferramenta desenvolvida no trabalho de dissertação, estão de acordo com o que foi identificado pelos especialistas ?

H.1 A utilização dessas métricas e seus valores referência foram úteis para identificação de *Bad Smells* ?

A ferramenta desenvolvida nessa dissertação tem como objetivo identificar *Bad Smells* em modelos *UML* a partir de diagramas de classes. Como não possuímos um repositório de diagramas de classes a disposição, utilizamos a engenharia reversa do *software Enterprise Architecture* para que fosse possível realizarmos nosso experimento e verificar se nossa metodologia identifica *Bad Smells* nos diagramas. Exportamos o diagrama de classes para o formato de arquivo *XMI*, por ser o modelo mais utilizado nos estudos avaliados do referencial teórico.

Os estudos preliminares foram realizados no diagrama de classes obtido a partir da engenharia reversa do *software Mobile Media*, versão 9. O *software* é feito em *Java*, possui 56 classes e 4 interfaces e trata-se de um aplicativo móvel para gerenciar imagens, músicas e vídeos.

Dois especialistas avaliaram o diagrama de classes para apontar classes com os *Bad Smells* avaliados. O primeiro especialista é um aluno de graduação que já cursou disciplinas relacionadas a programação modular e trabalha em projetos de iniciação científica na área de linguagens de programação. O segundo especialista é um Analista de Sistemas, que possui experiência de 8 anos em diagramas *UML* e programação Orientada a Objetos. Ambos apontaram as classes problemáticas em seu ponto de vista. Em seguida, o diagrama de classes foi submetido a análise da ferramenta. Os resultados reportados pela ferramenta foram comparados com as conclusões obtidas pela análise dos especialistas.

5. Avaliação dos Resultados

Para avaliar o método proposto, foram comparados os resultados das classes avaliadas como problemáticas pelos especialistas com as classes indicadas para análise pela ferramenta desenvolvida.

Para avaliação do *Indecent Exposure*, utilizamos a métrica NPA como regra. Sete classes atingiram valores regulares para a métrica, variando entre 2 a 8. Embora elas não apresentem valores ruins para a métrica, elas não foram implementadas dentro dos valores ideias para a métrica. Uma classe atingiu um valor acima do valor referência considerado crítico, pois declara 17 atributos como públicos. As classes identificadas pelos especialistas estavam todas dentro dos valores ruins ou regulares identificados pela ferramenta.

Bertrán (2009) utiliza duas métricas para calcular *Shotgun Surgery*: CC, que verifica se um parâmetro faz referência a alguma classe ou método não nativo do Java; CA verifica se um atributo faz referência a uma classe não nativa do *Java*. Ambas métricas geram conexões aferentes, que de acordo com Ferreira et al. (2012) é dada pelo número de classes externas ao agrupamento em questão que dependem das classes constituintes do agrupamento. Outro item que gera impactos em outras classes que não foi considerado pela autora é a herança, que também é uma conexão aferente. Então, para calcular esse *Bad Smell* utilizamos a métrica de cálculos de conexões aferentes.

De acordo com nossa ferramenta, duas classes estão com valores ruins para a métrica NCA, de acordo com os valores referência definidos por Ferreira et al. (2012), sendo 24 e 25 o valor encontrado para as duas. Pela ferramenta, ainda foram identificadas outras classes com valores entre 16 a 9 como valores regulares. Todas essas classes foram citadas na avaliação dos especialistas. Porém, outras classes com valores de 2 a 6, foram apontadas como regulares pela ferramenta, mas não foram citadas pelos especialistas, porém isso não prejudica o estudo, uma vez que todas as classes avaliadas pelos especialistas foram indicadas pela ferramenta para avaliação.

A métrica NPM associada à NCA nos possibilita identificar ocorrência do *Bad Smell God Class*. Em nossos experimentos, uma classe apresentou valores razoáveis em ambas as métricas. Duas outras classes possuem um nível de alerta na métrica NPM e atingiram valores ruins na métrica NCA. O resultado é confirmado pela análise manual, que concluiu que a estrutura de ambas podem ser divididas em outras classes.

Esse foi apenas um experimento preliminar. O próximo passo agora é ampliar o que já foi feito. Sabemos que será necessário avaliar novas métricas e *Bad Smells* a fim de obter valores mais próximos do que é definido pelos especialistas. Porém, com o que já foi feito, temos um indicativo de que o método possibilitará identificar de forma automática *Bad Smells* em diagramas de classes.

Para avaliar o método proposto, foram comparados os resultados das classes avaliadas como problemáticas pelos especialistas com as classes indicadas para análise pela ferramenta desenvolvida.

Para avaliação do *Indecent Exposure*, utilizamos a métrica NPA como regra. Sete classes atingiram valores regulares para a métrica, variando entre 2 a 8. Embora elas não apresentem valores ruins para a métrica, elas não foram implementadas dentro dos valores ideias para a métrica. Uma classe atingiu um valor acima do valor referência

considerado crítico, pois declara 17 atributos como públicos. As classes identificadas pelos especialistas estavam todas dentro dos valores ruins ou regulares identificados pela ferramenta.

Bertrán (2009) utiliza duas métricas para calcular *Shotgun Surgery*: CC, que verifica se um parâmetro faz referência a alguma classe ou método não nativo do Java; CA verifica se um atributo faz referência a uma classe não nativa do Java. Ambas métricas geram conexões aferentes, que de acordo com Ferreira et al. (2012) é dada pelo número de classes externas ao agrupamento em questão que dependem das classes constituintes do agrupamento. Outro item que gera impactos em outras classes que não foi considerado pela autora é a herança, que também é uma conexão aferente. Então, para calcular esse *Bad Smell* utilizamos a métrica de cálculos de conexões aferentes.

De acordo com nossa ferramenta, duas classes estão com valores ruins para a métrica NCA, de acordo com os valores referência definidos por Ferreira et al. (2012), sendo 24 e 25 o valor encontrado para as duas. Pela ferramenta, ainda foram identificadas outras classes com valores entre 16 a 9 como valores regulares. Todas essas classes foram citadas na avaliação dos especialistas. Porém, outras classes com valores de 2 a 6, foram apontadas como regulares pela ferramenta, mas não foram citadas pelos especialistas, porém isso não prejudica o estudo, uma vez que todas as classes avaliadas pelos especialistas foram indicadas pela ferramenta para avaliação.

A métrica NPM associada à NCA nos possibilita identificar ocorrência do *Bad Smell God Class*. Em nossos experimentos, uma classe apresentou valores razoáveis em ambas as métricas. Duas outras classes possuem um nível de alerta na métrica NPM e atingiram valores ruins na métrica NCA. O resultado é confirmado pela análise manual, que concluiu que a estrutura de ambas podem ser divididas em outras classes.

Esse foi apenas um experimento preliminar. O próximo passo agora é ampliar o que já foi feito. Sabemos que será necessário avaliar novas métricas e *Bad Smells* a fim de obter valores mais próximos do que é definido pelos especialistas. Porém, com o que já foi feito, temos um indicativo de que o método possibilitará identificar de forma automática *Bad Smells* em diagramas de classes.

6. Contribuições

O resultado positivo do experimento preliminar nos permite realizar experimentos mais amplos. Com esses novos experimentos pretendemos:

1. Identificar quais métricas melhor representam os aspectos relacionados a *Bad Smells* em modelos *UML*.
2. Identificar e avaliar valores referências definidos na literatura por outros estudos.
3. Definir um método de identificação de *Bad Smell* a partir de diagramas de classe da UML, utilizando métricas de software e seus valores referência.
4. Disponibilizar uma ferramenta que dê suporte para engenheiros de *software* na identificação de *Bad Smells* logo no início do projeto.

7. Conclusões

Os resultados desse experimento preliminar indica que a análise reportada pela ferramenta e pelo método proposto estão de acordo com a análise dos especialistas. Na continuidade do trabalho, implementaremos a coleta de outras métricas e estudaremos outros *Bad Smells* que possam ser identificados com auxílio delas e de seus valores referência. Serão realizados experimentos com outros *softwares* abertos a fim de se avaliar o método proposto.

Referências

Bertrán, I. M. (2009). Avaliação da qualidade de software com base em modelos uml.

Ferreira, K. A. M. ; Bigonha, M. A. S. ; Bigonha, R. S. ; Mendes, L. F. O. ; Almeida, H. C. . Identifying thresholds for object-oriented software metrics. The Journal of Systems and Software, v. 85, p. 244-257, 2012.

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

Girgis, M.; Mahmoud, T. & Nour, R. (2009). Uml class diagram metrics tool. In Computer Engineering Systems, 2009. ICCES 2009. International Conference on, pp. 423 –428.

H. Hamza, S. Counsell, T. Hall, and G. Loizou. 2008. Code smell eradication and associated refactoring. In Proceedings of the 2nd conference on European computing conference (ECC'08), Costin Cepisca, Guennadi A. Kouzaev, and Nikos E. Mastorakis (Eds.). World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 102-107.

Lanza, M.; Marinescu, R. & Ducasse, S. (2005). Object-Oriented Metrics in Practice. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Marinescu, R. (2002). In Measurement and Quality in Object-Oriented Design.

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

Nuthakki, M. K.; Mete, M.; Varol, C. & Suh, S. C. (2011). Uxsom: Uml generated xml to software metrics. SIGSOFT Softw. Eng. Notes, 36(3):1--6.

Soliman, T.; El-Swesy, A. & 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, pp. 1 –6.

Sommerville, I. (2007). Engenharia de software 8a ed.

Yi, T.; Wu, F. & Gan, C. (2004). A comparison of metrics for uml class diagrams. SIGSOFT Softw. Eng. Notes, 29(5):1--6.