UMLsmell: uma Ferramenta de Detecção de *Bad Smells* em Softwares a partir de Modelos UML

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

[1]DCC/UFMG e [2]DECOM/CEFET-MG

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

Abstract. *Software metrics are useful in the assessment of code, and they may aid the identification of design anomalies, known as bad smells. Nevertheless, it is also important to identify such anomalies in the early stages of software life-cycle, as well as by means of representations of the software. This work presents a tool to identify bad smells via software metrics, in object-oriented software, having as entry UML class diagrams.*

Resumo. *Métricas de software são úteis para avaliar a qualidade do código-fonte e podem auxiliar na identificação as anomalias de projeto, conhecidos na literatura como bad smells. Todavia, é importante também que essas anomalias possam ser identificadas em fases iniciais do ciclo de vida do software e a partir de modelos do software. Este artigo propõe uma ferramenta para a identificação de bad smells, via métricas de software, em sistemas orientados por objetos a partir de diagramas de classes da UML.*

1. Introdução

As métricas de software podem auxiliar na identificação das anomalias de projeto, conhecidos na literatura como *bad smells* ([Fowler 1999]). 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, poucos trabalhos propuseram avaliar *bad smells* nas fases iniciais de projetos de software. Como avalia Sommerville [Sommerville 2007], quanto mais tardia for a a detecção de falhas em um projeto, maior será o custo de manutenção, afetando tempo e custo e, consequentemente, a qualidade do produto final.

Identificamos alguns problemas que ainda não possuem soluções disponíveis na literatura, e as poucas existentes ainda não são satisfatórias. São eles: (1) carência de métodos que identifiquem anomalias de projetos nas fases iniciais do desenvolvimento de software. (2) ausência de métodos e ferramentas que automatizem a identificação de *bad smells* em modelos UML; (3) há poucos experimentos que validam estratégias de detecção em modelos UML.

Contornar esses problemas é importante, pois a identificação de *bad smells* nas fases iniciais do projeto de software, como na fase de modelagem por exemplo, contribuem para que os problemas nas fases posteriores do desenvolvimento de software sejam reduzidos. Com o objetivo de contribuir para a solução destes problemas, este artigo apresenta uma ferramenta, denominada *UMLsmell*, que automaticamente, via métricas de software, identifica *bad smells* em sistemas orientados por objetos a partir de diagramas de classes de modelos UML.

2. Método de Identificação de *Bad Smells*

A propósito do trabalho de pesquisa em que se insere este artigo é a automatização na identificação de *bad smells* em diagramas de classe da UML. Para atingir esse objetivo, foi proposto um método baseado em métricas e seus valores referências que permitem identificar as partes que possivelmente apresentam anomalias de projeto. Para aplicar o método proposto foi projetada e implementada a ferramenta *UMLsmell*. Essa seção descreve esse método [Nunes et al. 2014].

Inicialmente, foram identificados, dentre os *bad smells* descritos na literatura, aqueles que podem ser aplicados a modelos de classe da UML. Os *bad smells* e as métricas que podem ser aplicadas para identificá-los são descritos a seguir.

- *God Class*: este *bad smell* está associado às métricas que calculam a quantidade de relacionamentos e as métricas que calculam o número de métodos da classe avaliada, que permitem avaliar se uma classe possui muitos métodos e se muitas classes dependem da classe avaliada.
- *Indecent Exposure*: este *bad smell* está associado às métricas de atributos públicos que permitem avaliar o encapsulamento das classes.
- *Shotgun Surgery*: 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.

Os valores referência usados foram definidos por Ferreira [Ferreira et al. 2012]. A escolha desses valores referência se deu por duas razões principais: (1) por que os mesmos consideram a frequência de vários softwares e não simplesmente a média, e (2) por que os valores referência foram avaliados empiricamente, e há poucos valores referência propostos na literatura. Dentre as métricas para as quais os autores propõem valores referência, três delas podem ser obtidas via diagramas de classes: número de conexões aferentes (NCA), número de métodos públicos (NMP) e número de atributos públicos (NAP). Os valores referência das métricas são classificados como: *Bom*: valores mais frequentes para métricas em softwares de boa qualidade; *Regular*: valores pouco frequentes para métricas em softwares de boa qualidade; *Ruim*: valores raros para métricas em softwares de boa qualidade.

A idéia básica das faixas *bom*, *regular* ou *ruim* é a seguinte: uma vez que os valores são frequentes em softwares, 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. Por esta razão, Nunes et al. [Nunes et al. 2014] optaram por considerar as faixas *regular* e *ruim* na identificação de *bad smells*. Essa abordagem é adotada em UMLsmell. Os valores referência para as métricas utilizadas são mostrados na Tabela 1.

3. *UMLsmell*

Esta seção apresenta a arquitetura de *UMLsmell* e a metodologia usada para seu desenvolvimento, incluindo decisões de projeto, funcionalidades oferecidas e sua implementação.

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

Tabela 1. Valores referência definidos em [Ferreira et al. 2012].

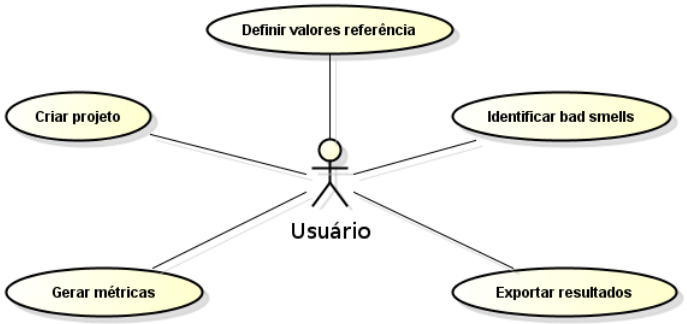

Figura 1. Casos de Uso da UMLsmell.

3.1. Requisitos de *UMLsmell*

O diagrama de casos de uso da UMLsmell é mostrado na Figura 1. Os casos de uso são os seguintes:

- Definir valores referência: permite que o usuário personalize os valores referência em *regular* e *ruim* para cada métrica coletada pela ferramenta. Esta funcionalidade foi implementada para que o usuário tenha liberdade de escolher os valores referência que preferir utilizar.
- Criar projeto: cria um arquivo que armazena todas as alterações feitas nos valores referência. Esta funcionalidade é útil caso o usuário altere os valores referência conforme sua necessidade e pretenda salvá-los para utilizações posteriores da *UMLsmell*.
- Gerar métricas: permite que o usuário possa selecionar, dentre as métricas existentes no sistema, aquelas para as quais ele deseja gerar relatórios para análises posteriores. Esta funcionalidade permite ao usuário verificar os valores das métricas selecionadas para as classes do software.
- Identificar *bad smells*: é o ponto central da ferramenta. Sua funcionalidade consiste em receber como entrada um arquivo XMI, que corresponde ao diagrama de classes a ser analisado, e identificar via métricas, em quais classes foram identificados os *bad smells* e em qual nível, *regular* ou *ruim*, conforme o método para identificação de *bad smells* proposto neste trabalho.
- Exportar resultados: permite que a análise das métricas e dos *bad smells* sejam exportados para o formato de planilha, aceito pela maioria das ferramentas de planilhas disponíveis atualmente.

3.2. Implementação de *UMLsmell*

A arquitetura de UMLsmell é ilustrada na Figura 3. *UMLsmell* recebe como entrada um arquivo XMI, no qual é realizada a análise sintática com o objetivo de extrair as seguintes

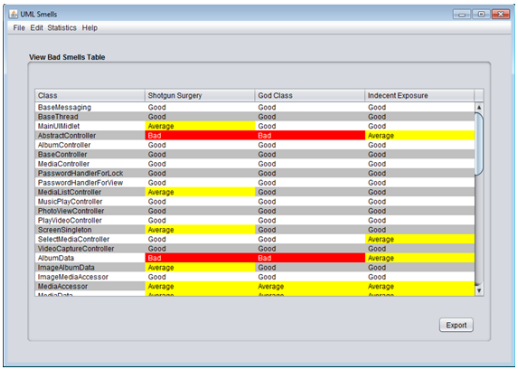

Figura 2. Telas com as Estatísticas dos *Bad Smells* da *UMLsmell*.

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 a funcionalidade *Gerar métricas* possa fornecer as informações de métricas das classes do software ao usuário. Essas métricas permitem que a funcionalidade *Identificar bad smells* detecte os *bad smells* do software analisado.

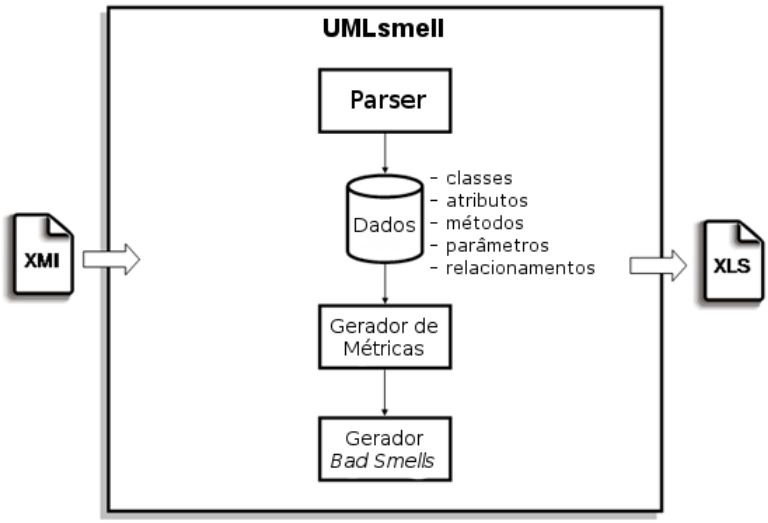

Figura 3. Estrutura da UMLsmell.

A ferramenta foi desenvolvida na linguagem Java e contém 3205 linhas de código. *UMLsmell* foi projetada e desenvolvida por dois programadores, um aluno de mestrado e outro de iniciação científica. A Figura 2 exibe o diagrama de classes, das camadas de controle e modelo, usado para desenvolver a ferramenta. Na camada de modelo foram criadas quatro classes: *ClassData*, *Attribute*, *Method* e *Relationship*. Essas classes são responsáveis por armazenar na memória as informações coletadas durante a análise sintática.

- *ClassData*: classe responsável por informações relativas às classes e interfaces do software avaliado. Também mantém vetores com dados sobre atributos, métodos e relacionamentos das classes.

- *Attribute*: classe responsável pelas informações relativas aos atributos do software. Informa o nome de um atributo, tipo e tipo de acesso.
- *Method*: classe responsável pelas informações relacionadas aos métodos do software. Informa o nome do método, seu tipo de acesso e possui uma lista do tipo *Attribute* com os parâmetros de entrada do método.
- *Relationship*: classe responsável pelas informações relativas aos relacionamentos entre as classes do software. Informa as classes nas duas pontas do relacionamento e qual o tipo de relacionamento.

Existe também um conjunto de classes que possui como superclasse a classe Metric, tal que todas as suas subclasses são as métricas existentes no sistema. O padrão de projeto *Strategy*, apresentado por Gamma et al. [Gamma et al. 2006], foi usado nesta parte da implementação, de tal forma que as subclasses herdam as informações gerais das métricas, como nome, valor e valor referência, e, as subclasses, via polimorfismo, implementam o método *generateMetric*, tal que cada subclasse possui sua forma de calcular a respectiva métrica. Esse método permite facilmente criar outras métricas no sistema.

Na camada de controle, foi criada uma classe, a *DataCollector*. Essa classe é responsável por ler o arquivo XMI, coletar todas as informações e criar as instâncias da camada de entidade. Essa classe funciona como o analisador sintático do arquivo XMI. Para coletar os dados necessários para gerar as métricas que permitirão a identificação de *bad smells*, é necessário definir quais são as informações relevantes em um diagrama de classes. Neste trabalho foram usadas as seis informações sugeridas por Genero [Genero et al. 2005], são elas: pacotes, classes, atributos, métodos, parâmetros e relacionamentos. Informações extraídas do diagrama de classes são armazenadas em blocos muito bem definidos no XMI. Blocos do tipo <*element*> representam tanto os pacotes, quanto as classes e interfaces. O que os diferencia é o atributo *"xmi:type"* definido nas *tags* desses blocos. Fora do bloco <*element*>, também existem os blocos <*packagedElement*> que definem todas informações como classes, interface, atributos, métodos, parâmetros e relacionamentos presentes em um pacote. Dentro do bloco <*element*>, existem os blocos <*attribute*>, <*operation*> e <*links*> que fornecem todas informações referentes aos atributos, métodos e relacionamentos do elemento em questão, classe, interface ou pacote. Dentro do bloco <*operation*> existem os blocos <*parameter*> que são os parâmetros do método em questão. Informações referentes a nomes, níveis de acesso (privado, público e protegido), número de identificação dos elementos e todas outras informações referentes a cada item, encontram-se dentro dos blocos ou em forma de atributo em cada bloco.

3.3. Interface com o Usuário

A ferramenta possui as seguintes interfaces de usuário:

- Tela com a Lista de Projetos Criados: tela inicial que apresenta uma lista de todos os projetos já criados para serem carregados.
- Telas para a Definição dos Valores das Métricas: são duas telas, uma para selecionar uma métrica e outra para definir os valores referência da métrica selecionada.
- Telas com as Estatísticas das Métricas: são duas telas, uma para selecionar as métricas e outra para exibir os valores das métricas do arquivo XMI carregado.
- Telas com as Estatísticas dos *Bad Smells*: são duas telas, uma para selecionar os *bad smells* a serem avaliados, e outra com os valores dos *bad smells* para cada classe avaliada. A interface é apresentada na Figura 4.

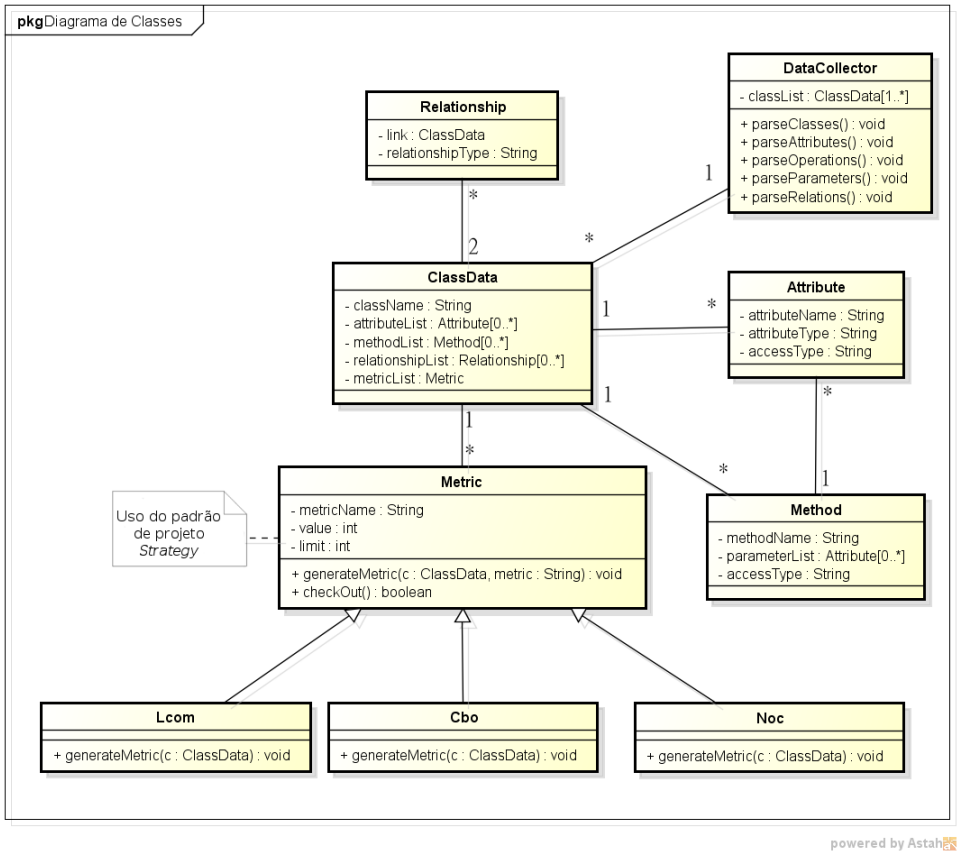

Figura 4. Diagrama de Classes da UMLsmell.

4. Trabalhos Relacionados

Girgis [Girgis et al. 2009] apresenta uma ferramenta capaz de calcular automaticamente diversas métricas relacionadas ao diagrama de classes. O objetivo dos autores é que a partir delas seja possível tomar decisões acerca de um projeto de software. Eles dividem as métricas em quatro grupos: métricas de complexidade, métricas de tamanho de software, métricas de acoplamento e métrica de herança, todas extraídas de diagramas de classes. A ferramenta recebe como entrada um diagrama de classe, porém no formato de uma *Extensible Markup Language* chamado XMI. A ferramenta coleta as seguintes informações para as classes do modelo UML: atributos, métodos e relacionamentos. As métricas são geradas a partir desses dados e os resultados são exibidos para o usuário.

Soliman et al. [Soliman et al. 2010] apresentam uma ferramenta que coleta seis métricas CK. Essas métricas são extraídas de diagramas da UML, que são representados em arquivos do tipo XMI. O trabalho de Nuthakki et al. [Nuthakki et al. 2011] se diferencia dos outros trabalhos uma vez que a ferramenta proposta por eles, UXSOM, é capaz de extrair métricas em diferentes formatos de arquivos gerados pelas ferramentas de modelagem. UXSOM suporta diagramas de classes exportados para os formatos XML, UXF e XMI, e, utiliza os arquivos de representação de diagramas gerados pelas seguintes ferramentas: ArgoUML, UMLet, ESS-MODEL, Magic Draw e Enterprise Architect.

Marinescu et al. [Marinescu et al. 2005] definem uma ferramenta, o iPlasma, para automatizar algumas de suas estratégias de detecção para *bad smells*. A ferramenta foi

feita para detectar *bad smells* em código fonte. Bertrán [Bertrán 2009] desenvolveu uma ferramenta, denominada QCDTool, com o objetivo de automatizar o controle de qualidade em modelos UML, ou seja, a aplicação automatizada das estratégias propostas e dos modelos de qualidade. Muito embora o trabalho de Bertran tenha definido estratégias de detecção que podem ser aplicadas a diagramas UML, algumas questões foram deixadas em aberto: (1) [Bertrán 2009] relata, por exemplo, a importância de redefinir os valores referência utilizados em seus estudos experimentais, definidos por [Marinescu 2002], pois considera que a técnica utilizada por [Marinescu 2002] não seja a melhor forma de definí-los. (2) [Bertrán 2009] relata também que os experimentos foram realizados apenas por uma pessoa e que o contexto e tamanho dos softwares utilizados não diferem tanto. (3) A autora ainda sugere que novos experimentos sejam realizados para identificar novos *bad smells* em outros diagramas da UML.

Os trabalhos de Girgis, Soliman e Nuthakki discutem acerca de extração de métricas a partir de modelos UML e consideram o diagrama de classes como o modelo predominante quando o foco é medição de software a partir da UML. Outro ponto comum nos trabalhos destes 3 autores está relacionado com o uso do formato de arquivo XMI para representar as métricas coletadas do diagrama de classes. Porém, nenhum desses três trabalhos relaciona as métricas com atributos externos de um sofware, ou seja, não fornece informações relevantes acerca do projeto a ser desenvolvido. O trabalho de Marinescu apresenta uma solução para transformar métricas em informações relevantes para detectar *bad smells*. Porém, esse trabalho lida apenas com informações coletadas em códigos fonte, que em geral passam a existir apenas em fases já avançadas do ciclo de vida dos softwares.

A ferramenta apresentado neste artigo visa contribuir para a solução do problema de identificar anomalias de projeto em fases iniciais do ciclo de vida do sistemas. Para isso, este trabalho se baseia nos seguintes aspectos: (1) define na ferramenta valores referência propostos sistematicamente na literatura; (2) a metodologia aplicada na ferramenta não depende de código fonte e avalia quantitativa e qualitativamente os resultados; (3) a ferramenta está disponível para uso aberto.

5. Conclusão

Este artigo descreveu as principais funcionalidades de *UMLsmell*, a ferramenta desenvolvida para identificar automaticamente *bad smells* em diagramas de classes. Também foram apresentados a arquitetura e as classes usadas em seu desenvolvimento. A ferramenta foi desenvolvida baseando-se em uma metodologia para identificação de *bad smells* em diagramas de classes. O objetivo da ferramenta é identificar falhas de projeto nas fases iniciais do ciclo de vida do software.

UMLsmell foi utilizada previamente em dois experimentos: (1) o primeiro comparou resultados da identificação de *bad smells* em duas versões de 6 softwares, uma refatorada e outra não, a fim de comprovar que a versão refatorada apresentaria menos *bad smells*; (2) o segundo comparou avaliação manual de 15 especialistas com os resultados gerados pelo *UMLsmell*. No primeiro experimento os softwares refatorados apresentaram uma diminuição de *bad smells* em relação aos softwares não refatorados. No segundo experimento os resultados da avaliação feita por especialistas foi próxima dos resultados gerados pelo *UMLsmell*. Isso mostra que o *UMLsmell* é capaz de identificar *bad smells*

em diagramas de classes. Os experimentos são apresentados no trabalho de Nunes et al. [Nunes et al. 2014].

Com os resultados obtidos neste trabalho, identificam-se os seguintes trabalhos futuros: (1) implementar novas métricas na ferramenta que permitam a identificação de outros *bad smells*; (2) implementar outros módulo de identificação de *bad smells* em diagramas de colaboração da UML, como diagramas de sequência, a fim de verificar se é possível identificar outros *bad smells* além daqueles considerados neste artigo.

Referências

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

Ferreira, K. A. M., Bigonha, M. A. S., Bigonha, R. S., Mendes, L. F. O., 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.

Gamma, E., Johnson, R., Helm, R., and Vlissides, J. (2006). *Padrões de Projetos: Soluções Reutilizáveis*. Bookman, Porto Alegre, RS.

Genero, M., Piattini, M., and Calero, C. (2005). A survey of metrics for uml class diagrams. *Journal of Object Technology*, 4(9):59–92.

Girgis, M., Mahmoud, T., and Nour, R. (2009). Uml class diagram metrics tool. In *Computer Engineering Systems. ICCES 2009.*, pages 423 –428.

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

Marinescu, C., Marinescu, R., Mihancea, P. F., and Wettel, R. (2005). iplasma: An integrated platform for quality assessment of object-oriented design. In *In ICSM (Industrial and Tool Volume*. Citeseer.

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

Nunes, H. G., Bigonha, M. A. S., Ferreira, K. A. M., and A., M. F. (2014). Um método para identificação de bad smells a partir de diagramas de classes. In *Proceedings of the 2nd Workshop on Software Visualization, Evolution and Maintenance*, pages 102–109. SBC.

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

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.