Code Smell Analyzer: A Tool To Teaching Support Of Refactoring Techniques Source Code

T. F. M. Sirqueira, A. H. M. Brandl, E. J. P. Pedro, R. S. Silva and M. A. P. Araújo

***Abstract*— The present paper addresses the refactoring techniques defended by Martin Fowler (2002), and bad smells present in your work. A tool was developed to support teaching and assessing academic works called Code Smell Analyzer, which aims to identify bad smells codes, presenting how they can be corrected and avoided using refactoring techniques. The tool works integrated into the Eclipse IDE (Integrated Development Environment) and aims to assist in the teaching and assessment of academic works about bad smells and refactoring of source code.**

***Keywords*— Refactoring Techniques, Bad Smell Source, Code Smell Analyzer.**

I. INTRODUÇÃO

CADA vez mais a demanda de software vem crescendo e, com isso, a preocupação com a qualidade do software desenvolvido. A qualidade de um software é um aspecto sensível da percepção que não pode ser mensurado ou geometrizado. Para considerar que um software possua qualidade, não basta funcionar corretamente, mas necessita ser bem estruturado e que possa ser facilmente mantido [1].

A Engenharia de Software é uma disciplina preocupada com a aplicação de teoria, conhecimento e prática para o desenvolvimento efetivo e eficiente de sistemas de software que satisfaçam os requisitos dos usuários [2]

O desenvolvimento e a manutenção de software demandam profissionais cada vez mais qualificados, a fim de produzir software para os mais diferentes domínios e propósitos [3], mas como abordado por [4], a forma de ensino centrada no professor, ou os métodos tradicionais de ensino com base em livros texto e aulas expositivas, têm se mostrado insuficientes.

As abordagens mais comuns para ensinar Engenharia de Software incluem aulas expositivas, aulas de laboratório, entre outros. Entretanto, abordagens alternativas podem ajudar estudantes a aprender de maneira mais efetiva [5], como aplicando ferramentas que os auxiliem no aprendizado sobre boas práticas de desenvolvimento.

T. F. M. Sirqueira, Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais - Campus Juiz de Fora, Juiz de Fora, MG, Brasil, tassio.sirqueira@ifsudestemg.edu.br

A. H. M. Brandl, Faculdade Metodista Granbery, Juiz de Fora, MG, Brasil, allan.brandl@yahoo.com.br,

E. J. P. Pedro, Faculdade Metodista Granbery, Juiz de Fora, MG, Brasil, evandro.jp.pedro@gmail.com,

R. S. Silva, Faculdade Metodista Granbery, Juiz de Fora, MG, Brasil, souzasilva.ramon@gmail.com,

M. A. P. Araújo, Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais - Campus Juiz de Fora, Juiz de Fora e Universidade Federal de Juiz de Fora (UFJF), MG, Brasil, marco.araujo@ifsudestemg.edu.br.

Uma possibilidade para a criação de software com melhor qualidade, é aperfeiçoando o aprendizado dos estudantes sobre boas práticas de desenvolvimento, evitando os chamados maus cheiros de código fonte que, de acordo com [6], podem ser tratados utilizando técnicas de refatoração. Esse conjunto de técnicas consiste em modificar o código fonte do software sem afetar a sua funcionalidade, permitindo uma melhor compreensão do código e reduzindo o custo envolvido em manutenções posteriormente. Embora não exista uma estimativa exata com relação ao custo pertinente à manutenção de software, estudos como o de [7], apontam um percentual entre 67% e 75% do custo total, enquanto que, para [8], corresponde a um valor entre 67% e 90%.

A refatoração surgiu da necessidade dos desenvolvedores em adicionar ou modificar funcionalidades em código fonte que, na maioria dos casos, estavam desestruturados, difícil de serem compreendidos e com trechos duplicados, o que tornava a manutenção ainda mais custosa [9]. Apesar da refatoração visar a melhoria do código fonte, deve ser bem pensada e executada, para não trazer riscos como atrasos, inserção de falhas no sistema ou tornar o código de difícil entendimento para futuras manutenções.

Neste trabalho foram pesquisadas ferramentas que auxiliam na detecção desses maus cheiros e fornecem técnicas de refatoração que possam ser usadas na reestruturação do código fonte. Com base nas análises e estudos realizados, foi desenvolvida uma ferramenta em forma de *plug-in*, integrada ao *Eclipse IDE* (*Integrated Development Environment*), que detecta cinco maus cheiros em código fonte *Java* (Código Duplicado, Inveja dos Dados, Classe Grande, Método Longo e Comentário), oferecendo ao estudante uma ferramenta que auxilia no aprendizado de identificação de problemas de maus cheiros e como podem ser tratados. Além disso, a ferramenta desenvolvida pode ser utilizada pelo professor para avaliar o código fonte e a qualidade do software desenvolvido por seus estudantes em trabalhos acadêmicos.

Este trabalho possui 4 seções além desta introdução. A seção 2 apresenta os pressupostos teóricos e os trabalhos relacionados. Já na seção 3 é apresentada a ferramenta *Code Smell Analyzer*, objeto deste trabalho. Na seção 4 estão dispostas algumas análises realizadas com a ferramenta em ambiente acadêmico. Por fim, a seção 5 apresenta as conclusões e os trabalhos futuros.

II. PRESUPOSTOS TEÓRICOS

Para criar um software com qualidade tem-se que evitar os chamados maus cheiros em código fonte, que podem ser o uso excessivo de comentários, código duplicado, métodos com longa lista de parâmetros ou a construção incorreta da

arquitetura do software. Quando o software já está pronto e apresenta alguns desses maus cheiros, uma possibilidade de tratá-los é através de técnicas de refatoração, que consiste em alterar o código fonte a fim de eliminar os maus cheiros mantendo sua funcionalidade, o que pode ser caracterizado como uma manutenção preventiva ou de reengenharia, e visa melhorar o código fonte existente.

Para realizar a refatoração, a primeira coisa é identificar os maus cheiros no código fonte, essa atividade pode ser realizada de forma manual ou com auxílio de ferramentas computacionais que, conforme abordado por [10], tais ferramentas de detecção de maus cheiros em código fonte carecem de maturidade com relação a intervenções manuais e, para [11], a maioria das ferramentas de refatoração não possuem mecanismos que permitam ao utilizador especificar suas próprias pré-condições de análise, permitindo configurar quais parâmetros definem que um método é longo, por exemplo. Grande parte das ferramentas identificadas na pesquisa estão preocupadas em simplesmente identificar e tratar os maus cheiros de código fonte, e não em avaliar e apoiar o aprendizado de estudantes sobre como evitar os maus cheiros e, quando detectados, quais as técnicas possíveis para correção.

A ferramenta *Code Smell Analyzer*, ao identificar um mau cheiro no código fonte, exibe ao aluno uma descrição sobre o porquê trata-se de um mau cheiro e apresenta, com base no trabalho de [6], técnicas possíveis para resolver o problema, além do detalhamento de como utilizá-las, de forma que o aluno possa entender o motivo que a ferramenta classificou o código como mau cheiro e o ensina como trata-lo, caso se depare com um exemplo similar.

Em [12], foi apresentado um *plug-in* do *Eclipse* chamado *SmellChecker* onde, através de métricas de software, é possível detectar problemas em classes e maus cheiros nos códigos, dentre eles, Método Longo, Lista de Parâmetros Longa, Código Duplicado, Generalidade Especulativa e Campo Temporário. O trabalho ainda apresenta uma funcionalidade para detectar a probabilidade de um desses maus cheiros ocorrerem no código fonte em análise. Outros trabalhos são o *TrueRefactor* [10] e *JCodeCanine* [13]. Esses trabalhos são bem próximos ao desenvolvido nesse artigo, mas não tem como foco o apoio ao aprendizado sobre identificação de maus cheiros de código e a indicação de técnicas de refatoração que podem tratar o mau cheiro.

Durante a pesquisa foi realizada uma revisão sistemática, que é uns dos meios existentes para identificar, avaliar e interpretar toda pesquisa pertinente a uma pergunta de pesquisa em particular [14], onde se buscou identificar ferramentas para detecção de maus cheiros e refatoração de código fonte. Para a realização da revisão sistemática, foi utilizada a *string* de busca *(“Refactoring tool” <OR> “Ferramenta de Refatoração”) <AND> (“Code Smell” <OR> “Cheiro de Código”) <AND> (“Software Engineering” <OR> “Engenharia de Software”) <AND> (“Bad Smell” <OR> “Mau Cheiro”)* contendo palavras chaves da pesquisa.

Essa *string* de busca foi aplicada sobre a base de dados do *Google Scholar*, sendo essa a única base disponível no momento da revisão, e retornou um total de 35 artigos. Dentre os resultados obtidos, foram aplicados os filtros de inclusão e exclusão onde o artigo deveria tratar de mau cheiro em código fonte *Java* e estar disponível na *Web* e, como segundo filtro, apenas as ferramentas que estavam disponíveis para utilização. Os trabalhos de [12], [10] e [13] não puderam ser analisados por não estarem disponíveis no momento da pesquisa. O resultado pode ser observado na Tabela I.

TABELA I. COMPARATIVOS DOS TRABALHOS RELACIONADOS.

Ferramenta	Tipo	Linguagem	Maus Cheiros Tratados	*Open Source*
IntelliJ IDEA [15]	IDE	Java, Java Web	Código Duplicado	Sim
JDeodorant [16]	*Plugin*	Java	Inveja dos dados, Método Longo, Classe Grande	Sim
PMD [17]	Software	Java, JavaScript, XML XSL	Código Duplicado, Método Longo, Classe Grande, Intimidade Inadequada, Classes de dados	Sim
InFusion [18]	IDE	Java, C, C++	Classe Grande, Classe de Dados, Código Duplicado, Inveja dos Dados e Cirurgia com Rifle	Não
InCode [19]	IDE	Java, C, C++	Classe Grande, Classe de Dados, Código Duplicado e Inveja dos Dados	Não

Os ambientes de desenvolvimento como as *IDE’s Eclipse* e *NetBeans* também foram descartados uma vez que, apesar de conterem métodos de refatoração automáticos, não detectam maus cheiros. Com base nesses trabalhos, criou-se a ferramenta *Code Smell Analyzer*.

III. *CODE SMELL ANALYZER*

O objetivo da ferramenta *Code Smell Analyzer* é identificar maus cheiros em código fonte e informar para o estudante como, através da refatoração, problemas identificados podem ser tratados e, para uso do professor, permitir avaliar a qualidade dos trabalhos desenvolvidos pelos estudantes. A *Code Smell Analyzer* foi desenvolvida em *Java* como um *plug-in* para trabalhar integrado ao *Eclipse IDE*, na versão 3.5 ou superior.

A ferramenta *Code Smell Analyzer* trata inicialmente de cinco diferentes tipos de maus cheiros de código fonte, que são de acordo com [6]:

- Classe Grande: quando uma classe realiza diversas tarefas acaba criando variáveis de instância em exagero, com isso passa a ter muitas linhas de código fonte, métodos longos e a possibilidade de código duplicado, além da necessidade de comentários para tentar explicar cada parte do mesmo. Isso torna a classe mais complexa e de difícil manutenção. Para resolver isso, pode-se aplicar refatorações para extrair métodos, classes ou subclasses;
- Código Duplicado: muitas das vezes o problema mais comum de mau cheiro, o código duplicado pode

ocorrer quando se tem expressões em dois métodos parecidos, ou a mesma expressão em duas subclasses irmãs. Outro caso é quando dois métodos fazem a mesma tarefa usando algoritmos diferentes, nesse caso a melhor opção seria escolher o algoritmo mais claro e menos complexo. Uma forma de corrigir esse mau cheiro no código fonte é através da extração de métodos, onde retira-se a parte duplicada e transforma-se em um método novo, e todos os locais onde existiam esse determinado trecho de código passa a invocar o método criado;

- Método Longo: apesar do uso de linguagens OO (Orientadas a Objetos), muito desenvolvedores iniciantes tendem a criar métodos longos e complexos, seja pela quantidade exagerada de parâmetros ou sequências intermináveis de delegações. Isso força o desenvolvedor a criar longos comentários explicando a função do código, o que acarreta em mais um mau cheio de código. Para solucionar isso, a melhor opção, muitas vezes, é subdividir o método em métodos menores, utilizando a técnica de refatoração de extrair método. Com isso, evita-se o método longo e o excesso de comentários;

- Inveja dos Dados: esse tipo de mau cheiro ocorre quando uma classe possui um determinado método que passa a usar em excesso dados advindos de outra classe. A eliminação dessa estrutura consiste em levar o método para a classe onde os dados requeridos estão. Esse problema pode ser tratado utilizando a técnica mover método ou, em outros casos, quando apenas parte do método sofre a inveja, a forma de tratar esse mau cheiro é usando a técnica de extrair método ou mover método;

- Comentários: o uso de comentários no código fonte, apesar de não interferir no processamento do sistema e não possuir a necessidade de manutenção, com base no seu uso excessivo, pode indicar a ocorrência de trechos de código mal estruturados, e criar o que pode ser considerado como poluição visual no código fonte.

A princípio foram selecionados esses maus cheiros por se tratarem dos mais comuns em equipes de desenvolvimento e por serem passíveis de identificação através de algoritmos de análise do código fonte. No futuro, espera-se expandir a ferramenta para detectar outros tipos de maus cheiros.

Para uso da ferramenta, dentro da *IDE Eclipse*, seu acesso pode ser realizado por meio do menu de contexto clicando sobre a classe que se deseja analisar. Quando executado, ao encontrar algum mau cheiro, a ferramenta retorna uma janela sugerindo a forma correta de tratá-lo, a fim de eliminá-lo.

Assim, a ferramenta *Code Smell Analyzer* analisa os maus cheiros no código fonte com base em parâmetros armazenados na ferramenta, que podem ser personalizados com base na necessidade do utilizador. Caso os parâmetros não tenham sido informados para a ferramenta, é solicitado ao utilizador que os forneça, para que assim as análises possam ser realizadas. Esses parâmetros dizem respeito à quando considerar que um método é longo, ao tamanho da classe para ser considerada grande e quando o uso de comentários passa a se tornar um mau cheiro no código fonte.

A Fig. 1 apresenta as métrica extraídas do SCA (Sistema de Controle Acadêmico), um sistema desenvolvido por estudantes do Curso Superior de Bacharelado em Sistemas de Informação do IF Sudeste MG, Campus Juiz de Fora, que foi utilizado para avaliar a utilização da ferramenta *Code Smell Analyzer*. Para demonstrar o uso da ferramenta, as análises a seguir serão aplicadas sobre a classe "Aluno", em destaque na Fig. 1, que apresenta 251 linhas de código fonte, 90 linhas de comentários e um método com 17 parâmetros.

Ao ser executada a análise da classe pela ferramenta *Code Smell Analyzer*, serão identificados maus cheiros de código fonte, apresentando uma janela com os problemas localizados e orientações de como, utilizando técnicas de refatoração, podem ser solucionados, apoiando estudantes nas técnicas para tratar os maus cheiros, conforme a Fig. 2.

Tree	NOP	VG	LOC	LOCm
SCA	184	1	2913	752
java.model	88	1	913	188
Turma	11	1	86	1
Professor	31	1	217	6
Disciplina	11	1	103	30
Curso	14	1	150	43
Avaliacao	4	1	43	0
Aluno	17	1	251	90
java.dao	8	1	614	60
java.controller	82	1	1295	462
java.Exception	6	1	91	42

Figura 1. Métricas extraídas do software SCA.

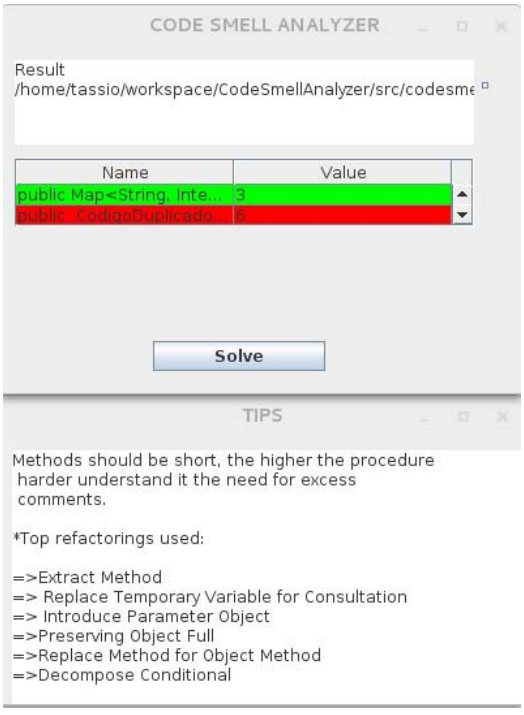

Figura 2. Análise da classe "Aluno" pela ferramenta, apresentando a janela contendo as técnicas de refatoração.

Quando o objetivo é a análise de código duplicado, a ferramenta solicita ao estudante as classes que devem ser analisadas, um trecho de código duplicado pode ser visto na Fig. 3. Com base nos parâmetros previamente configurados, é retornada uma janela com os trechos de código que apresentam o mau cheiro em questão, conforme a Fig. 4, e as possíveis técnicas que podem ser utilizadas para tratá-lo. A primeira coluna apresenta o trecho de código duplicado e na segunda coluna a quantidade de vezes que ele se repete entre as classes analisadas.

Figura 3. Trecho de código fonte duplicado.

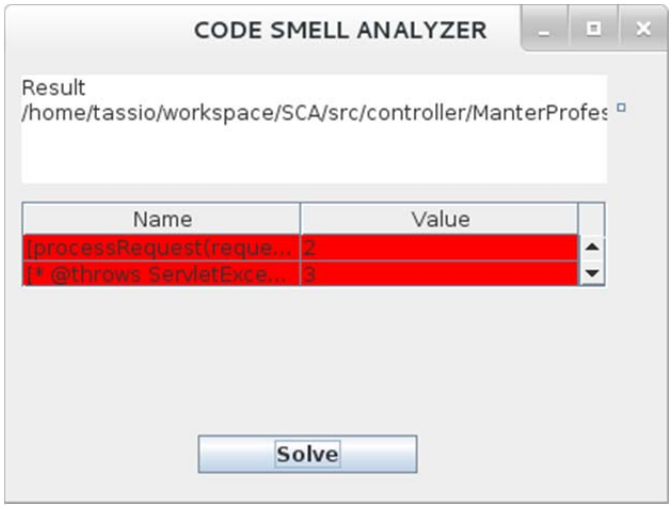

Figura 4. Análise de código fonte duplicado pela ferramenta.

Já na análise de Inveja dos Dados, a ferramenta verifica a dependência de cada método e retorna uma tabela listando se existe esse mau cheiro no código fonte. Conforme a Fig. 5, o valor zero representa que não existe inveja dos dados na classe analisada.

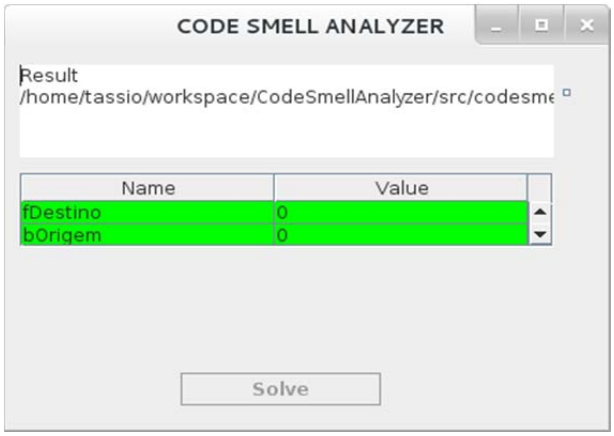

Figura 5. Análise de inveja dos Dados pela ferramenta.

No caso do mau cheiro no código fonte relacionado ao excesso de comentários, a ferramenta apresenta uma tabela com os resultados da análise. Caso o número de comentários estejam causando mau cheiro no código fonte por conta da poluição visual gerada, a ferramenta destaca qual tipo de comentário está causando o mau cheiro em vermelho nos resultados, como apresenta a Fig. 6. A ferramenta *Code Smell Analyzer* classifica os comentários no código fonte de duas formas:

- Comentários simples: são os comentários que não ultrapassam uma linha e são iniciados pelo uso do caractere "/" duas vezes, ou seja, "//";
- Comentários multilinhas: são os comentários que além de possuírem mais de uma linha, são delimitados pelo uso dos caracteres especiais "/*" no início do comentário e "*/" para delimitar o fim do mesmo.

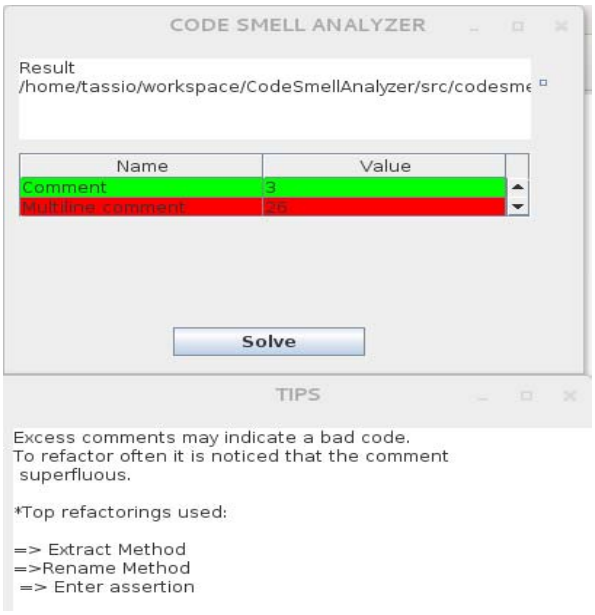

Figura 6. Análise de mau cheiro dos comentários no código fonte.

Como mostra a Fig. 6, a primeira coluna vem identificando o tipo de comentário e na segunda coluna a quantidade de vezes que o tipo de comentário se repetiu dentro da classe analisada.

Caso a ferramenta não detecte nenhum mau cheiro no código fonte por conta do uso de comentários, o resultado apresentado na tabela da análise vem com a linha em verde, como na Fig. 6, caso o excesso de comentários esteja causando o mau cheiro, é destacado em vermelho.

Caso as análises não encontrem nenhum dos tipos de maus cheiros considerados no código fonte, a ferramenta retorna uma janela informando que não há nada com necessidade de ser refatorado dentro dos parâmetros definidos na ferramenta, considerando os maus cheiros tratados.

Apesar da ferramenta ainda estar em desenvolvimento, abordando apenas cinco dos vinte e dois maus cheiros descritos por [6], a *Code Smell Analyzer* se mostra útil como forma de apoio ao ensino sobre maus cheiros em código fonte onde, aplicando sua análise em trabalhos acadêmicos, pode identificar pontos a serem corrigidos, como será apresentado na seção IV deste trabalho.

IV. AVALIAÇÃO DA FERRAMENTA *CODE SMELL ANALYZER*

Um dos desafios enfrentados no ensino de Engenharia de Software (ES) é suprir a demanda de uso de métodos de ensino que permitam tornar o processo de ensino e de aprendizagem mais efetivo [22]. Em função disso, a forma tradicional de ensino centrada no professor, faz com que falte aos estudantes oportunidades para aplicação prática dos conceitos [23]. Uma forma de melhorar a situação é através do uso de métodos de ensino complementares como, por exemplo, o uso de ferramentas computacionais.

A ferramenta *Code Smell Analyzer* foi avaliada em dois estudos. No primeiro, a ferramenta foi empregada na verificação de 20 trabalhos acadêmicos, sendo esses trabalhos de estudantes do 3º ano do Curso Técnico Integrado em Informática, e do 4º e 5º períodos do curso superior de Bacharelado em Sistemas de Informação, ambos do Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais, Campus Juiz de Fora.

A análise foi realizada pelos professores das disciplinas de programação com o objetivo de identificar problemas de maus cheiros nos códigos fonte nos trabalhos desenvolvidos nas disciplinas, e assim poderem aprofundar com os estudantes técnicas de refatoração, e como os maus cheiros podem ser evitados. Foram realizadas análises dos trabalhos buscando-se identificar os maus cheiros de Método Longo, Comentário, Classe Grande e Inveja dos Dados.

Um código mal escrito pode trazer uma série de problemas. A utilização de meios para melhorar a estrutura das aplicações facilita futuros trabalhos como manutenção e evolução do software. O foco deste estudo é demostrar a importância de refatoração do código fonte, quando existe a necessidade de ser aplicada e como deve ser realizada.

Não é objetivo aqui discutir o nível de conhecimento dos estudantes, uma vez que, para cursar as referidas disciplinas, pré-requisitos sobre os conteúdos utilizados necessitam ter sido cursados, mas auxiliar no aprendizado para identificação dos maus cheiros em código fonte e técnicas de refatoração.

Os trabalhos analisados se tratavam de um Sistema de Controle Acadêmico, desenvolvidos em *Java Web* no modelo *MVC (Model-View-Controller)*, contendo uma média de 23 classes, com cerca de 5 métodos por classe e em torno de 104 linhas de código fonte cada classe. A Fig. 7 apresenta o resultado da análise dos trabalhos que apresentaram algum mau cheiro dentre os tratados pela ferramenta, considerando os 20 trabalhos desenvolvidos.

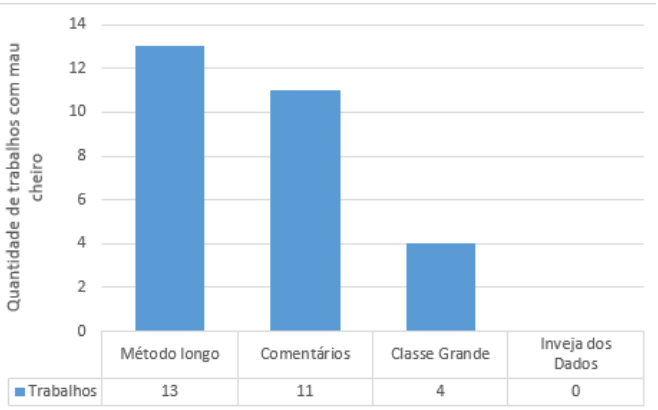

Figura 7. Análise dos trabalhos acadêmicos pela ferramenta.

Após a análise dos trabalhos acadêmicos, identifica-se, com base no uso da ferramenta *Code Smell Analyzer*, que dos 20 trabalhos, 4 (20%) apresentavam classes grandes, já o mau cheiro Inveja dos Dados não foi identificado em nenhum dos trabalhos analisados. Os maus cheiros de Método Longo e Comentários, foram identificados em 13 (65%) e 11 (55%) trabalhos, respectivamente.

Com base na análise anterior, foi realizada uma segunda verificação para identificar quantas ocorrências para cada um dos maus cheios listados seriam encontrados nos trabalhos. Conforme apresenta a Fig. 8, pode-se verificar que o mau cheiro Classe Grande se repete em um total de 10 vezes entre os 4 trabalhos onde foram encontrados, o mau cheiro Método Longo se repete 58 vezes entre os 13 trabalhos onde foram encontrados, e o mau cheiro Comentário também se repete 58 vezes entre os 11 trabalhos onde foi identificado.

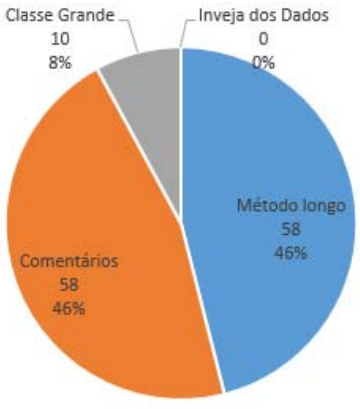

Figura 8. Número de vezes que o mau cheiro foi identificado no código fonte.

Entre os maus cheiros relacionados ao uso de comentários, muitos foram identificados como sendo provenientes da geração automática de código pelos estudantes, onde a própria *IDE* gera os comentários, como observado, na geração dos

métodos de *GET* e *SET* de atributos das classes no sistema, e outra parte está relacionada com a explicação de métodos longos, onde nesse caso os comentários foram feitos pelos estudantes durante o desenvolvimento do software. Entre os trabalhos que apresentavam o mau cheiro Classe Grande, em todos os casos, também apresentavam mau cheiro de Método Longo e Comentários.

A ferramenta *Code Smell Analyzer* se mostrou eficiente na identificação dos maus cheiros de código fonte considerados, auxiliando os professores no ensino e verificação de boas práticas de desenvolvimento e técnicas de refatoração, revendo as técnicas de ensino de forma a enfatizar os problemas causados por maus cheiros em código fonte. Além da verificação utilizando a ferramenta *Code Smell Analyzer*, foi realizada a análise manual pelos professores, como forma de verificar se todos os maus cheiros presentes no código fonte do SCA foram encontrados, essa análise manual serviu de comparação para avaliar a eficiência da ferramenta em questão, que demostrou atender o objetivo proposto.

No segundo estudo, realizado com 13 estudantes do 5º período do Curso de Bacharelado em Sistemas de Informação do Instituto Federal de Educação, Ciência de Tecnologia do Sudeste de Minas Gerais, Campus Juiz de Fora, foi preparada uma aula sobre maus cheiros de código fonte e, logo após, entregue aos estudantes o código fonte de um sistema contendo uma série de maus cheiros inseridos propositalmente, maus cheiros esses identificáveis pela ferramenta *Code Smell Analyzer*.

Foi solicitado que cada estudante fizesse a análise manual do código e marcasse em um formulário quais os maus cheiros que identificou, por meio da observação, e em qual classe ou método os mesmos estariam presentes. Além disso, cada estudante marcou no formulário as possíveis técnicas de refatoração empregadas na solução do mau cheiro por ele classificada. Após a análise manual pelos estudantes, foi solicitado que a mesma análise fosse realizada com a ferramenta *Code Smell Analyzer*, com o intuito de comparar os resultados, onde assim pode-se verificar a eficiência da ferramenta no auxílio sobre o apoio ao ensino de maus cheiros no código fonte e das técnicas de refatoração indicadas pela ferramenta. Assim o estudante teve a possibilidade de verificar se o que havia classificado como mau cheiro era de fato, utilizando a ferramenta para esta função.

Na análise realizada de forma manual pelos estudantes para encontrar inveja dos dados, presentes nas classes "CursoDAO", "ProfessorDAO" e "TurmaDAO", o total de estudantes que identificaram o mau cheiro pode ser visto na Fig. 9.

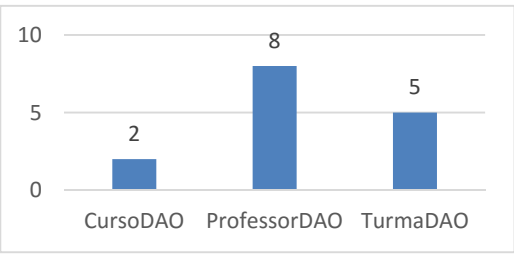

Figura 9. Análise manual para identificação de Inveja dos Dados.

Já para a identificação do mau cheiro de Classe Grande, existente na classe "Aluno", o total que o identificou foram 10 estudantes, como pode ser visto na Fig. 10.

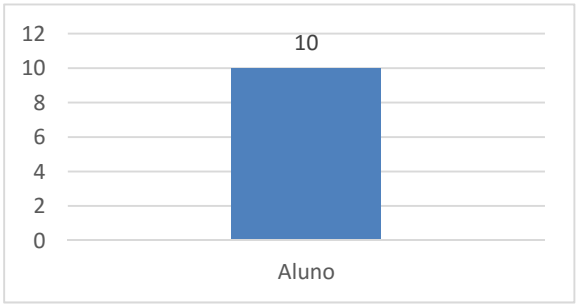

Figura 10. Análise manual para identificação de Classe Grande.

Já o mau cheiro de método longo, estava presente nas classes "Professor" e "Turma" e foi encontrado por apenas um estudante, como pode ser visto na Fig. 11.

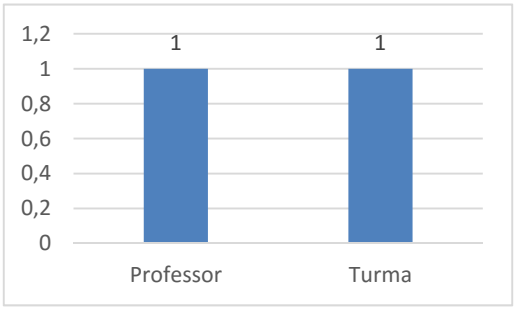

Figura 11. Análise manual para identificação de Método Longo.

Já o mau cheiro de Comentários, foi o mais identificado entre os estudantes, estava presente em 16 classes no sistema e a quantidade de estudantes que o identificou em cada classe pode ser visto na Fig.12.

Por fim, foi verificado o nível de acerto através da busca manual para identificação dos maus cheiros realizadas pelos estudantes, como pode ser visto na Fig. 13. O nível de acertos foi baixo na busca manual, tornando necessário a utilização da ferramenta para auxiliar os estudantes na identificação dos maus cheiros em código fonte.

O ganho da ferramenta está justamente nela explicar para o aluno quais os maus cheiros que existem na classe analisada e como podem ser tratados. Apesar das demais ferramentas listadas identificarem os maus cheiros, não trazem explicação do porque ocorreu e a explicação de quais as técnicas existem para tratar cada mau cheiro, apenas removendo-o.

Ao nos referirmos a ferramenta como um facilitador no processo de ensino e de aprendizagem sobre os maus cheiros em códigos fontes, não nos prendemos a nenhuma corrente pedagógica especifica. É importante destacar que a ferramenta não substitui o professor, mas o auxilia em suas aulas sobre boas práticas no desenvolvimento de software.

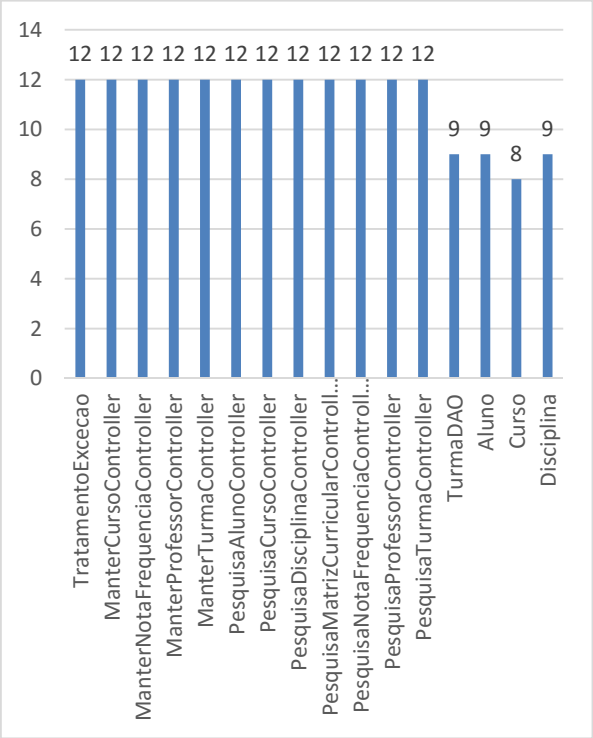

Figura 12. Análise manual para identificação do mau cheiro Comentário.

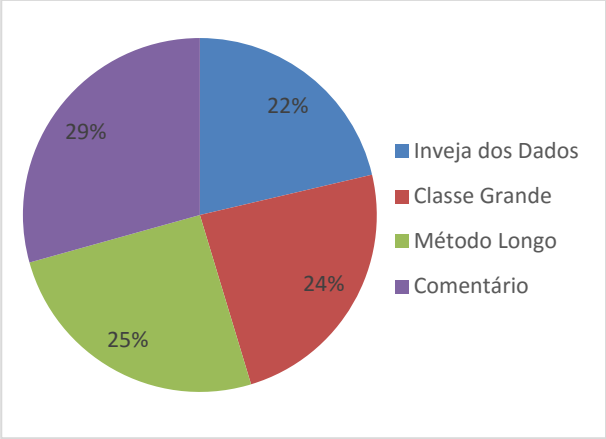

Figura 13. Percentual de acerto na identificação de mau cheiro de forma manual.

Os ganhos com o uso da ferramenta, auxiliando os alunos na identificação dos maus cheiros, foram observados durante o segundo estudo pois, além da aula ministrada pelo professor, a atividade prática permitiu aos alunos identificar nos exemplos de código fonte, quando o mesmo pode ser classificado como mau cheiro e na aplicação das técnicas para resolver esses problemas.

Ainda não se pode tirar conclusões definitivas sobre a efetividade da ferramenta no apoio ao ensino das técnicas propostas, e pretende-se realizar novos estudos experimentais com a ferramenta sobre novas perspectivas. O objetivo com um novo estudo é verificar o grau de compreensão dos alunos sobre maus cheiros em código fonte por meio da divisão em dois grupos, um utilizando a ferramenta e outro sem o uso, posteriormente confrontando quais dos grupos absorveu melhor o conteúdo e consegue melhor identificar as situações em que ocorrem.

V. CONCLUSÃO

Com base no estudo realizado, foram encontradas poucas ferramentas que tratam da identificação de maus cheiros de código fonte e têm como foco o apoio ao ensino de identificação e técnicas de refatoração para tratar os problemas de maus cheiros em código fonte. Tendo em vista a complexidade envolvida no desenvolvimento de software, com a utilização dessa ferramenta os autores visam contribuir para que estudantes e desenvolvedores possam compreender e evitar a inserção de maus cheiros no desenvolvimento de suas aplicações.

A ferramenta *Code Smell Analyzer* é capaz de identificar cinco maus cheiros, sendo eles, Código Duplicado, Inveja de Dados, Método Longo, Classe Grande e Comentários. Por ser um *plug-in* do ambiente *Eclipse* facilita sua utilização, uma vez que não é preciso instalar uma nova ferramenta para que o código fonte criado seja analisado.

As vantagens da ferramenta *Code Smell Analyzer* com relação às demais citadas, é que não somente identifica os maus cheiros no código fonte, mas também explica o porquê se trata de um mau cheiro e apresenta, de forma detalhada, as possibilidades que o aluno possui para fazer a correção. A ferramenta pode ser utilizada tanto por alunos durante a criação de software, de forma a evitarem ou removerem maus cheiros no código fonte, bem como por professores, para avaliar o software desenvolvido pelos alunos. Algumas limitações que a ferramenta apresenta são em relação ao número de maus cheiros identificados, identificando 5 dos 22 maus cheiros classificados por [6] e não suportando até então código em linguagem C e C++, como é o caso de algumas das ferramentas listadas.

A atividade de manutenção de software é a atividade que exige maior esforço dentre as atividades de engenharia de software [20], e quando se tem códigos mal escritos, tende a ser ainda pior. Utilizando meios de tornar a estrutura do software melhor, facilitará os trabalhos de manutenções dessas aplicações no futuro, com custo menor, seja financeiro, de tempo ou de esforço [21]. Espera-se que esta ferramenta contribua na compreensão de maus cheiros em código fonte, visto que, nos estudos preliminares, demonstrou potencial para tal apoio.

Como trabalhos futuros, o objetivo é expandir a ferramenta para que possa aumentar a gama de maus cheios identificados, novas técnicas de refatoração e implementação do cálculo de métricas, além de permitir que a ferramenta trabalhe com outra linguagens além de JAVA. Outro objetivo do estudo é a realização de novos experimentos como forma de avaliar a contribuição da ferramenta desenvolvida neste trabalho, para o apoio ao ensino sobre maus cheiros de código fonte e técnicas de refatoração de código, bem como sua utilização pelo professor.

Pretende-se ainda tornar a ferramenta de domínio público para que outros estudantes e professores possam utilizá-la e contribuir para sua evolução, até que seu uso possa ser expandido além da sala de aula, colaborando com equipes de desenvolvimento e manutenção de software.

REFERÊNCIAS

[1] ROCHA, Ana Regina Cavalcante da; MALDONADO, José Carlos; WEBER, Kival Chaves. Qualidade de software. 2001.

[2] CASSEL, L. et al. Computer science curriculum 2008. Association for Computing Machinery (ACM)/IEEE Computer Society. Retrieved December, v. 10, p. 2009, 2008.

[3] BILLA, Cleo Z.; CERA, Márcia C. Utilizando Resolução de Problemas para aproximar Teoria e Prática na Engenharia de Software. 2012.

[4] FERREIRA, Bruna et al. Apoiando o Ensino de Qualidade de Software. FEES-Fórum de Educação em Engenharia de Software. 2014.

[5] PRIKLADNICKI, Rafael et al. Ensino de engenharia de software: desafios, estratégias de ensino e lições aprendidas. FEES-Fórum de Educação em Engenharia de Software, p. 1-8, 2009.

[6] FOWLER, Martin. Refatoração: Aperfeiçoamento e Projeto. Bookman, 2004.

[7] BHATT, P.; Shroff, G.; Misra, A.K. (2004) "Dynamics of software maintenance", ACM SIGSOFT Software Engineering Notes, v. 29, n. 5 (September), p. 1-5.

[8] POLO, M.; Piattini, M.; Ruiz, F.; Calero, C. (1999) "Roles in the maintenance process", ACM SIGSOFT Software Engineering Notes, v. 24, n. 4 (July), p. 84-86.

[9] MOREIRA, Lucas Neto. Avaliação de qualidade de software baseada em métricas estáticas: um estudo de caso no Tribunal de Contas da União. Brasília: UnB, 2015.

[10] GRIFFITH, Isaac; WAHL, Scott; IZURIETA, Clemente. TrueRefactor: An automated refactoring tool to improve legacy system and application comprehensibility. In: Proceedings of the ISCA 24rd International Conference on Computer Applications in Industry and Engineering. 2011.

[11] LIANG, Yan. Code refactoring under constraints. Tese de Mestrado. The University of Alabama TUSCALOOSA. 2011.

[12] PESSOA, Tiago Alexandre Simões. A semi-automatic approach to code smells detection. Tese de Mestrado. Universidade Nova de Lisboa. 2011.

[13] NONGPONG, Kwankamol. Feature envy factor: A metric forautomatic feature envy detection. In: Knowledge and Smart Technology (KST), 2015 7th International Conference on. IEEE, 2015. p. 7-12.

[14] KITCHENHAM, B. Procedures for Performing Systematic Reviews, Keele Technical Report SE0401 and NICTA Technical Report 0400011T.1, 2004.

[15] JEMEROV, Dmitry. Implementing refactorings in IntelliJ IDEA. In: Proceedings of the 2nd Workshop on Refactoring Tools. ACM, 2008. p. 13.

[16] FOKAEFS, Marios et al. JDeodorant: identification and application of extract class refactorings. In: Proceedings of the 33rd International Conference on Software Engineering. ACM, 2011. p. 1037-1039.

[17] PMD. Disponível em http://pmd.sourceforge.net. Acessado em 16/04/2015.

[18] InFusion. Disponível em http://www.intooitus.com/products/infusion Acessado em 16/04/2015.

[19] InCode. Disponível em https://www.intooitus.com/products/incode. Acessado em 16/04/2015.

[20] Sneed, H.M. (2003) "Critical Success Factors in Software Maintenance", In: Proceedings of the International Conference on Software Maintenance, Amsterdam, The Netherlands, September.

[21] PADUELLI, Mateus M.; SANCHES, Rosely. Problemas em manutenção de software: caracterização e evolução. In: 3º Workshop de Manutenção de Software Moderna (WMSWM'2006)–Simpósio Brasileiro de Qualidade de Software (SBQS'2006). Vila Velha, ES. 2006.

[22] SANTOS, R. P., SANTOS, P. S. M., WERNER, C. M. L. e TRAVASSOS, G. H. (2008) "Utilizando Experimentação para Apoiar a Pesquisa em Educação em Engenharia de Software no Brasil", In: I FEES - I Fórum de Educação em Engenharia de Software, 2008, Campinas.

[23] SHAW, K. e DERMOUDY, J. (2005) "Engendering an Empathy for Software Engineering", In Proc. Seventh Australian Computing Education Conference (ACE2005). p. 135-144. Newcastle, Australia.

Tássio Ferenzini Martins Sirqueira é Mestrando em Ciência da Computação pela UFJF, Bacharel em Sistemas de Informação pelo CESJF e Técnico em Informática Industrial pelo SENAI-JFN. Atualmente é professor EBTT do Instituto Federal de Educação Ciência de Tecnologia do Sudeste de Minas Gerais – Campus Juiz de Fora.

Allan Henrique Moreira Brandl é graduado em Sistemas de Informação pela Faculdade Metodista Granbery.

Evandro José Pereira Pedro é graduado em Sistemas de Informação pela Faculdade Metodista Granbery.

Ramon De Souza Silva é graduado em Sistemas de Informação pela Faculdade Metodista Granbery.

Marco Antônio Pereira Araújo é doutor e mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ. Especialista em Métodos Estatísticos Computacionais e bacharel em Informática pela Universidade Federal de Juiz de Fora (UFJF). Professor adjunto e membro do Programa de Pós-graduação em Ciência da Computação da UFJF e professor adjunto do Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais - Campus Juiz de Fora.