Padrões de Projeto e *Bad Smell*: Um Estudo Avaliativo Aplicando Métricas de Software

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

Abstract. *The software development process is a factor that needs to be very well planned to use oriented programming objects. A poorly designed software, can present problems in its structure or code, thus hampering their reuse and future maintenance. These problems in the structure or software code are called Bad Smell. Design patterns are reusable solutions to general problems that occur frequently within a software project. When applying a design pattern in the software development process, it is expected to achieve a reusable structure without the presence of bad smells. Based on this, the objective of this work is to apply metrics to identify bad smells in software developed with design patterns in order to investigate how this solution helps to combat these problems and situations which lead this software to submit those problems. To achieve the proposed objectives were accomplished: (1) a literature review to trace the metrics detection strategy, (2) survey of software to be analyzed, (3) extraction of metrics, (4) identification of bad smells and finally, (5) analysis and discussion of results.*

Resumo. *O processo de desenvolvimento de software é um fator que necessita ser muito bem planejado ao utilizar programação Orientada por Objetos. Um software mal planejado, pode apresentar problemas em sua estrutura ou código, prejudicando assim a sua reutilização e manutenções futuras. Esses problemas na estrutura ou código do software são chamados de Bad Smell. Os padrões de projeto são soluções gerais reutilizáveis para problemas que ocorrem com frequência dentro de um projeto de software. Quando se aplica algum padrão de projeto no processo de desenvolvimento do software, espera-se atingir uma estrutura reutilizável sem a presença de bad smells. Baseado nisso, o objetivo deste trabalho é aplicar métricas para identificação de bad smells em softwares desenvolvidos com padrões de projeto, a fim de investigar o quanto essa solução auxilia no combate a esses problemas e quais situações levam esses softwares a apresentar esses problemas. Para alcançar os objetivos propostos, foram realizados: (1) uma revisão da literatura para traçar a estratégia de detecção das métricas, (2) levantamento dos softwares a serem analisados, (3) extração das métricas, (4) identificação dos bad smells e por fim, (5) análise e discussão dos resultados obtidos.*

1. Introdução

A popularização da informática permitiu que tarefas complexas quando realizadas de forma manual, se tornassem simples e rápidas quando executadas por meio de um computador. A partir daí, uma série de desenvolvimento de softwares e sistemas de informação

tem sido feitos a fim de automatizar o controle de informações e garantir uma maior agilidade, segurança e organização no gerenciamento das mesmas. Essas vantagens propiciaram a expansão softwares para diversas outras áreas como: saúde, educação, economia, engenharia, dentre outras. Porém, desenvolver software com qualidade e baixo custo não é uma tarefa tão trivial.

A Engenharia de software surgiu com o intuito de organizar o desenvolvimento e ainda fornecer métodos e ferramentas para auxiliarem na criação de um software. Um dos principais métodos fornecidos são os padrões de projetos. De acordo com [Gamma 1994], padrão de projeto é um tipo de solução reusável geral para um problema recorrente no projeto do software. Eles são compostos de classes e objetos que possuem uma comunicação, e eles devem ser personalizados para poder resolver um problema geral em um contexto específico. Quando são aplicados na criação de um software, este tende a possuir uma estrutura flexível, reusável e com facilidade de manutenções.

Outros três termos empregados pela Engenharia de Software e que se relacionam diretamente com a qualidade de um software são: métrica, medida e medição. [Pressman 2006] define esses três termos como:

- **métrica:** é um padrão de medição para avaliar um atributo de algo relacionado ao software.
- **medida:** é o resultado da medição. Uma medida indica quantitativamente a presença de um atributo em determinado software.
- **medição:** é o ato ou efeito de medir algo de acordo com a métrica.

As métricas constituem um padrão de avaliação quantitativo, no qual o engenheiro de software se baseia para poder realizar alguma alteração no produto, projeto ou processo a fim de aumentar a qualidade do produto final e evitar que os softwares possuam sintomas em certas partes do código fonte que possam gerar problemas em sua estrutura, os chamados *bad smells*.

Segundo [Fowler and Beck 1999], *bad smells* são sintomas apresentados no código fonte de um software que possivelmente indica um problema mais profundo e necessita de refatoração do código. As regiões que apresentam esses sintomas não são considerados erros no software, porém prejudicam a qualidade do software e violam princípios da engenharia de software como modularidade, legibilidade e reúso. [Kerievsky 2005] destaca em seu trabalho o uso de padrões de projetos para a remoção desses sintomas do código.

Baseado nesse contexto, o objetivo desse trabalho é investigar e comparar softwares orientado por objetos que aplicam padrões de projeto, e por meio de uma metodologia baseada na extração de métricas, responder as seguintes questões de pesquisa:

- **QP1:** A aplicação de padrões de projetos elimina a ocorrência de *bad smells* em softwares?
- **QP2:** Em que tipos de situações um *bad smell* aparece ou ocorre em um software que aplica padrões de projeto?

Para alcançar os objetivos propostos e responder essas questões de pesquisas, o trabalho foi dividido da seguinte forma. Seção 2, relata a pesquisa sobre os trabalhos relacionados para enfatizar a importância do tema. Seção 3, define uma estratégia de

detecção, faz um levantamento dos softwares a serem analisados e a extração das métricas. Seção 4, relata a execução dos experimentos a partir da metodologia proposta, e apresenta e discute os resultados dos softwares utilizados como estudo de caso. Seção 5, relata algumas ameças à validade do estudo. Seção 6, por fim, apresenta a conclusão da pesquisa realizada.

2. Trabalhos Relacionados

Quando se trabalha com identificação de *bad smell* aplicando métricas de software é necessário conhecer alguns termos importantes como: estratégia de detecção e valores referências. Esta seção inicialmente define esses termos, e em seguida, apresenta os trabalhos relacionados que inspiraram na elaboração deste trabalho de pesquisa.

Antes de iniciar a investigação sobre a presença de *bad smell* em algum software, é necessário definir uma estratégia de detecção. De acordo com [Bertrán 2009], estratégia de detecção consiste em uma expressão quantificável de uma regra, onde pode-se avaliar se os fragmentos de código fonte dos softwares estão em conformidade com a regra definida. Já [Marinescu 2002] define estratégia de detecção como uma conexão entre medidas de investigação e metas de nível superior por meio da definição de uma regra que quantifique e expresse uma meta de investigação. A estratégia de detecção aborda dois aspectos: mecanismos de filtragem de métricas e mecanismos de composição.

Os mecanismos de filtragem são filtros utilizados no conjunto inicial de dados que tem como objetivo reduzi-los ao passo que se possa verificar características especiais a partir dos resultados das métricas utilizadas.

Os mecanismos de composição são relações estabelecidas entre os filtros a fim de realizar uma mescla das métricas utilizadas e gerar resultados relevantes. Em [Marinescu 2002] e [Marinescu 2004] são definidos dois tipos de mecanismos de composição: ponto de vista lógico, onde são aplicados operadores lógicos às métricas, e ponto de vista de conjuntos, onde os filtros são agrupados em conjuntos.

Outro termo importante neste trabalho, valores referência, é definido por [Crespo et al. 1999] como uma forma de determinar a relação de uma métrica com *bad smells*, a fim de identificar a ocorrências destes em softwares.

Muitos trabalhos na literatura têm contribuído direto e indiretamente para a investigação de *bad smell* em softwares. [Marinescu 2002] define em seu trabalho diversas estratégias de detecção para interpretação e verificação dos resultados obtidos via medição de softwares orientado por objetos. Ainda, ao definir as estratégias de detecção, o autor relaciona e combina métricas com objetivos diferentes para a localização de vários *bad smells*.

[Ferreira 2011] realiza em seu trabalho uma identificação de valores referências para várias métricas presentes na literatura. Esses valores referências têm como finalidade definir limites para as métricas e indicar em qual tipo de classificação (Bom/Regular/Ruim) dado valor se encaixa. De acordo com [Ferreira 2011], essa é uma questão em aberto na literatura e sua solução é de grande valia para a tomada de decisão na produção de um software.

[Filó 2014] continua o trabalho iniciado por [Ferreira 2011] e propõe valores referências para outras diversas métricas na literatura. Em seu trabalho, foi utilizado uma

base de dados com 111 softwares, na qual foi aplicado um modelo empírico pré-definido com finalidade de derivar valores referências para 18 métricas. [Filó 2014], também constrói uma ferramenta para a realização da filtragem de métodos, classes e pacotes com base nos valores propostos, e correlaciona algumas métricas com dois tipos de *bad smell*, criando assim estratégias de detecção para os mesmos.

[Vale 2016] faz uma revisão sistemática da literatura a fim de identificar métodos de valores referência para métricas existentes. Após essa revisão, o autor realiza um estudo detalhado, no qual compara três métodos diferentes para cálculos de valores referência, e utiliza o conhecimento obtido desse estudo para propor seu próprio método, chamado *Vale's method*, no qual foram derivados valores referências para 8 métricas de softwares. Por fim, o autor desenvolve uma ferramenta para apoiar a derivação de métricas pro meio do método proposto e dos métodos comparados.

[Nunes 2014] propõe em seu trabalho um método para identificação de *bad smells* em diagrama de classes UML e constrói uma ferramenta para avaliação do método proposto. Inicialmente, o autor faz o levantamento de diversos *bad smells* existentes na literatura que podem ser aplicados ao modelo de classes UML e os associa com diversas métricas a fim de formar uma estratégia de detecção. Após a definição do método é construída a ferramenta UMLsmell para automatizar a coleta de métricas e aplicá-las na identificação dos *bad smells* utilizando o método proposto neste trabalho.

[Cardoso 2015] faz uma análise exploratória a fim de investigar a coocorrência de *bad smells* em softwares que fazem uso de padrões de projetos. O autor realizou uma revisão da literatura para compreender o estado da arte sobre o tema padrões de projetos e anomalias de software, e depois utilizou ferramentas para identificação de padrões de projeto e *bad smells* nos softwares utilizados no experimento. Os resultados finais foram analisados, e os padrões de projetos foram correlacionados de acordo com os *bad smells* apresentados para estes softwares.

Em resumo, os trabalhos apresentados nesta seção mostram a importância do tema explorado neste trabalho na atual realidade da Engenharia de Software. O trabalho de [Marinescu 2002] fornece estratégias de detecção para localização de *bad smells* em código. [Ferreira 2011], [Filó 2014] e [Vale 2016] propõe valores referências para as principais métricas utilizadas no processo de medição de software. Por fim, [Nunes 2014] e [Cardoso 2015] investigam ocorrência de *bad smells* em diagramas de classes UML e softwares com padrões de projetos, e por isso são os que mais se aproximam-se presente trabalho.

3. Metodologia

A metodologia utilizada para a confecção deste trabalho baseou-se nas seguintes etapas: definição de uma estratégia de detecção, definição dos softwares analisados, coleta de métricas e o planejamento dos experimentos realizados.

A primeira etapa preocupou-se com a definição de uma estratégia de detecção para avaliação dos softwares. Como existem várias estratégias propostas, foi realizada uma revisão na literatura de possíveis estratégias de detecção que poderiam ser aplicadas a esse trabalho. Durante essa revisão, dois trabalhos chamaram a atenção para a as estratégias propostas: [Marinescu 2002] e [Filó 2014].

[Marinescu 2002] propôs várias estratégias de detecção para três níveis diferentes: métodos, classes e pacotes. Essas estratégias eram muito interessantes e envolviam diversos tipos de *bad smells*. Porém as métricas propostas para identificação desses sintomas eram pouco conhecidas pela comunidade e não eram suportadas por ferramentas conhecidas como por exemplo o *metrics*. Assim, a coleta dessas métricas tornaria um empecilho para a concretização deste trabalho.

[Filó 2014] propõe em seu trabalho duas estratégias de detecção que utilizam métricas bastante conhecidas pela comunidade acadêmica e que são suportadas por ferramentas de coletas de métricas como o *metrics*. Essas estratégias são concentradas na identificação dos *bad smells God Class* e *Long Method* e utilizam os valores referências derivados neste mesmo trabalho como fator de indicação de qualidade das métricas no respectivo software analisado. Segundo [Lanza et al. 2006], o sintoma *God Class* ocorre quando uma classe executa muito trabalho e delega detalhes menores a outras classes. [Fowler and Beck 1999] relata que o sintoma *Long Method* é problemático porque muitas vezes eles contém grande quantidade de informações que ficam enterradas pela lógica complexa. [Filó 2014] também menciona que quanto mais classes e métodos o software possui com valores referências das métricas na faixa Bom, esses artefatos possuem menos probabilidade de conter o respectivo sintoma. Assim, para a verificação e identificação de possíveis *bad smell* em software, a investigação utiliza as faixas de valores Regulares e Ruins que são indícios de algum problema ou sintoma no software. Essas duas estratégias foram validas pelo autor em seu trabalho com dois estudos de casos para ambas, porém, nenhum trabalho na literatura ainda havia utilizado-as.

Baseado nisso, foram escolhidas as estratégias de detecção propostas por [Filó 2014], as quais utilizam para o *bad smell God Class* as métricas: Métodos Ponderados por Classes (WMC), Número de Métodos (NOM), Número de Atributos (NOF) e Ausência de Coesão (LCOM), e para o *bad smell Long Method* as métricas: Linhas de código por método (MLOC), Profundidade de Blocos Aninhados (NBD) e Complexidade de McCabe (VG). A partir disso, foram investigados os sintomas de *God Class* e *Long Method* nos softwares utilizados. A segunda etapa dessa metodologia baseou-se na definição da amostra de softwares a serem utilizadas nesse trabalho. Para a definição dos mesmos, foi utilizada uma base de dados chamada *Qualitas.class Corpus*, que é uma versão compilada do *Qualitas Corpus* proposta por [Tempero et al. 2010] e que foi disponibilizada por [Terra et al. 2013] para a comunidade científica. O *Qualitas Corpus* é uma base com uma coleção de software de código aberto desenvolvido em Java que são disponibilizados para estudos empíricos de artefato de código [Filó 2014].

Para que pudesse existir uma variabilidade nas amostras, foram escolhidos alguns softwares que já continham alguns estudos na literatura e softwares que ainda não haviam sido investigados. Assim, foram definidos, cinco softwares: *Hibernate*, *JhotDraw*, *Webmail*, *Kolmafia* e *Weka*. Os três primeiros, já possuíam alguns estudos na literatura em relação a *bad smell*. [Cardoso 2015] é um exemplo de trabalho que utiliza esses softwares para fazer uma correlação da ocorrência de *bad smell* com padrões de projeto. Os dois últimos softwares ainda não possuíam estudos em relação a investigação de *bad smells* presentes na literatura.

Para validar a existência dos padrões de projeto neste software, foi utilizada uma ferramenta proposta por [Tsantalis et al. 2006], chamada *Design Pattern Detection using*

Tabela 1. Padrões de Projetos encontrados nos softwares analisados.

	Hibernate	Jhotdraw	Kolmafia	Webmail	Weka
Factory Method	X	X	X	X	X
Prototype		X			
Singleton	X	X	X	X	X
Adapter	X	X	X		X
Command	X	X	X	X	X
Composite	X	X	X	X	X
Decorator	X	X	X		X
Observer	X	X	X		X
State	X	X	X	X	X
Strategy	X	X	X	X	X
Bridge	X	X	X	X	X
Template Method	X	X	X	X	X
Visitor					
Proxy	X				X

Similarity Scoring 4 (DPDSS). De acordo com o autor, essa ferramenta aplica um algoritmo chamado *Similarity Scoring*, em que a modelagem é realizada por meios de grafos direcionados representados por matrizes quadráticas.

Os padrões de projetos encontrados nos softwares após a aplicação da ferramenta estão representados na **Tabela 1**.

Definidos os softwares a serem a analisados neste estudo, começou-se a execução da terceira etapa. A terceira etapa implicou na coleta das métricas para análises deste estudo. Na base de softwares *Qualitas.class Corpus*, além dos softwares, existem arquivos em formatos xml com a extração das métricas referentes aos softwares dessa base. Como a maioria dos softwares utilizados neste trabalho foram retirados dessa base, e as métricas propostas para avaliação dos mesmos também já estavam coletadas nesses arquivos xml, esses arquivos foram aproveitados economizando assim bastante tempo em relação a coleta de métricas.

Apenas um software, *Kolmafia*, não encontra-se armazenado nessa base de dados. Por esse motivo, foi realizado o *download* do código aberto desse software, e foram coletadas as suas métricas. Para a realização da coleta de métricas do *Kolmafia*, foram utilizadas como ferramentas a *IDE eclipse 4.2 Juno*, como ambiente para a coleção das métricas, e o *plugin metrics 1.3.6*, foi instalado na *IDE eclipse* e é o responsável por realizar a coleta das métricas. Após a coleta das métricas deste software, essas foram exportadas e salvas em um arquivo de formato xml.

Após a etapa de coleta de métricas, iniciou a quarta etapa desta metodologia que implicou na aplicação das estratégias de detecção nas métricas colhidas, a qual representou o planejamento dos experimentos realizados neste trabalho. [Filó 2014], além das estratégias de detecção, construiu uma ferramenta em seu trabalho chamada Raftool. Essa ferramenta tem como objetivo principal realizar a filtragem de métodos, classes e pacotes que possuam medições anômalas de métricas de softwares orientados por objetos, no contexto do processo de medição de software. Baseada nessa definição da ferramenta,

esta foi integrada a esse trabalho com o propósito de retornar as classes ou métodos que apresentassem anomalias, a partir da filtragem utilizada via a estratégia de detecção.

Como está sendo usada duas estratégias de detecção, uma para *God Class* e outra para *Long Method*, foi necessária a transformação das estratégias de detecção em uma expressão de filtragem para que pudessem ser utilizadas na ferramenta Raftool. Nessa ferramenta, os valores referência das métricas que compõem a estratégia de detecção não são utilizados de forma numérica. Eles são representados pelas seguintes palavras chaves: COMMON, que corresponde as faixas Bom/Frequente das métricas, CASUAL, que corresponde as faixas REGULAR/OCASIONAL das métricas, e UNCOMMON, que corresponde as faixas Ruim/Raro das métricas. Ainda, é necessário atribuir como *id* para cada uma dessas palavras chaves a métrica a qual cada palavra-chave refere-se.

As expressões de filtragem utilizadas nesse trabalho foram:

- (UNCOMMON[WMC] AND UNCOMMON[NOF] AND UNCOMMON[NOM] AND UNCOMMON[LCOM]) OR (UNCOMMON[WMC] AND UNCOMMON[NOF] AND UNCOMMON[NOM] AND CASUAL[LCOM]) OR (UNCOMMON[WMC] AND UNCOMMON[NOF] AND CASUAL[NOM] AND UNCOMMON[LCOM]) OR (UNCOMMON[WMC] AND UNCOMMON[NOF] AND CASUAL[NOM] AND CASUAL[LCOM]) OR (UNCOMMON[WMC] AND CASUAL[NOF] AND UNCOMMON[NOM] AND UNCOMMON[LCOM]) OR (UNCOMMON[WMC] AND CASUAL[NOF] AND UNCOMMON[NOM] AND CASUAL[LCOM]) OR (UNCOMMON[WMC] AND CASUAL[NOF] AND CASUAL[NOM] AND UNCOMMON[LCOM]) OR (UNCOMMON[WMC] AND CASUAL[NOF] AND CASUAL[NOM] AND CASUAL[LCOM]) OR (CASUAL[WMC] AND UNCOMMON[NOF] AND UNCOMMON[NOM] AND UNCOMMON[LCOM]) OR (CASUAL[WMC] AND UNCOMMON[NOF] AND UNCOMMON[NOM] AND CASUAL[LCOM]) OR (CASUAL[WMC] AND UNCOMMON[NOF] AND CASUAL[NOM] AND UNCOMMON[LCOM]) OR (CASUAL[WMC] AND UNCOMMON[NOF] AND CASUAL[NOM] AND CASUAL[LCOM]) OR (CASUAL[WMC] AND CASUAL[NOF] AND UNCOMMON[NOM] AND UNCOMMON[LCOM]) OR (CASUAL[WMC] AND CASUAL[NOF] AND UNCOMMON[NOM] AND CASUAL[LCOM]) OR (CASUAL[WMC] AND CASUAL[NOF] AND CASUAL[NOM] AND UNCOMMON[LCOM]) OR (CASUAL[WMC] AND CASUAL[NOF] AND CASUAL[NOM] AND CASUAL[LCOM])
- (UNCOMMON[MLOC] AND UNCOMMON[NBD] AND UNCOMMON[VG]) OR (UNCOMMON[MLOC] AND UNCOMMON[NBD] AND CASUAL[VG]) OR (UNCOMMON[MLOC] AND CASUAL[NBD] AND UNCOMMON[VG]) OR (UNCOMMON[MLOC] AND CASUAL[NBD] AND CASUAL[VG]) OR (CASUAL[MLOC] AND UNCOMMON[NBD] AND UNCOMMON[VG]) OR (CASUAL[MLOC] AND UNCOMMON[NBD] AND CASUAL[VG]) OR (CASUAL[MLOC] AND CASUAL[NBD] AND UNCOMMON[VG]) OR (CASUAL[MLOC] AND CASUAL[NBD] AND CASUAL[VG])

A primeira expressão de filtragem refere-se ao *bad smell God Class*. Ela é composta de uma combinação de valores de referência regular e ruim (CASUAL e UNCOMMON) para as quatro métricas associadas a esse *bad smell* (WMC, NOF, NOM e LCOM).

Tabela 2. Resultados obtidos para *God Class*.

Software	Quantidade de classes que apresentaram God Class	Quantidade Totais de Classes	Percentual de classes com God Class
Hibernate	527	7711	6,83%
JhotDraw	122	1061	11,50%
Kolmafia	385	3225	11,94%
Webmail	15	129	11,63%
Weka	467	2401	19,45%

Nessa expressão de filtragem cada combinação é formada com relacionamento AND e ocorre um agrupamento de todas as combinações utilizando o relacionamento OR.

A segunda expressão de filtragem refere-se ao *bad smell Long Method*. Assim na expressão gerada para o *God Class*, ela é composta de uma combinação de valores referências regular e ruim (CASUAL e UNCOMMON) para as três métricas associadas a esse *bad smell* (NBD, VG, e MLOC). Nessa expressão também ocorre um relacionamento de AND para a formação de cada combinação e um relacionamento de OR para o agrupamento final das combinações.

4. Resultados

A seção 4.1 apresenta os resultados obtidos após a execução dos experimentos, e a seção 4.2 apresenta a discussão dos resultados, onde são respondidas as questões de pesquisas propostas para este trabalho.

4.1. Apresentação dos Resultados

Após a definição da metodologia utilizada neste trabalho, começou-se a execução dos experimentos. Com já foi falado, utilizou-se a ferramenta Raftool, construída por [Filó 2014], na qual foram aplicadas as expressões de filtragem de acordo com o *bad smell* a ser filtrado no software.

O primeiro experimento realizado, foi a filtragem por classes, onde o principal objetivo era identificar quais classes pertences aos softwares utilizados nesse trabalho poderiam apresentar o *bad smell God Class*. Para isso, foi aplicado a expressão de filtragem, mostrada na seção anterior, referente a esse *bad smell* e os resultados obtidos são ilustrados na **Tabela 2**.

Após a execução do primeiro experimento, observou-se que os softwares apresentaram uma quantidade significativa de classes com o *bad smell God Class*, uma vez que quando aplica-se padrões de projetos em um software, espera-se uma software com uma estrutura flexível, livre de *bad smells*. Quando analisado o percentual dos dados apresentados, percebeu-se que o *Hibernate* foi o software que apresentou uma menor fatia desses sintomas, e o *Weka* foi o que apresentou uma maior fatia. Os demais softwares mantiveram-se praticamente na mesma faixa de percentual. Apesar de o percentual ser um dado importante, quando foram analisados a quantidade de classes que apresentaram esse *bad smell*, percebeu-se que os softwares apresentaram grande quantidade de classes com esses sintomas. Apenas o *Webmail* que apresentou uma quantidade menor desse sintoma, pelo fato de possuir menos classes que os demais softwares.

Tabela 3. Resultados obtidos para *Long Method*.

Software	Quantidade de métodos que apresentaram Long Method	Quantidade Totais de Métodos	Percentual de métodos com Long Method
Hibernate	2883	48234	5,98%
JhotDraw	995	7633	13,04%
Kolmafia	5400	28078	19,23%
Webmail	131	1091	12,01%
Weka	3822	20871	18,31%

O segundo experimento realizado, foi a filtragem por métodos, onde o principal objetivo era identificar quais métodos pertences aos softwares utilizados nesse trabalho poderiam apresentar o *bad smell Long Method*. Para isso, foi aplicado a expressão de filtragem, mostrada na seção 3, referente a esse *bad smell* e os resultados obtidos foram mostrados na **Tabela 3**.

Após a execução do segundo experimento, observou-se que os softwares também apresentaram uma quantidade significativa de métodos com o *bad smell Long Method*. Quando analisado o percentual dos dados apresentados, percebe-se que o *Hibernate* novamente foi o software que apresentou uma menor fatia desses sintomas, e desta vez o *Kolmafia* foi o que apresentou uma maior fatia. Os demais softwares tiveram uma variação entre 12% a 18%, dos quais o Weka apresentou um maior percentual com 18,31%. Novamente, quando analisadas a quantidade de métodos que apresentaram esse *bad smell*, percebeu-se que um alto teor de métodos com esses sintomas.

Depois de executar esses dois primeiros experimentos, houve uma suspeita em relação a estratégia de detecção indicar classes e métodos com a presença desses sintomas, sendo que estes não possuem, ou seja, os falsos positivos. Para verificar essa suspeita, foi criada uma outra expressão de filtragem, considerando apenas a faixa Ruim para das métricas, foi executado dois experimentos com essa filtragem, e depois realizada uma verificação das classes e métodos presentes nos dois primeiros experimentos e ausentes nos dois últimos para confirmar se estas poderiam ou não ser consideradas falsos positivos. As expressões utilizadas para esses dois últimos experimentos foram:

- UNCOMMON[WMC] AND UNCOMMON[NOF] AND UNCOMMON[NOM] AND UNCOMMON[LCOM]
- UNCOMMON[MLOC] AND UNCOMMON[NBD] AND UNCOMMON[VG]

A primeira expressão de filtragem foi utilizada para o *bad smell God Class* e a segunda para o *bad smell Long Method*. Após executar esses dois últimos experimentos, foi retornado um número bem menor de classes e métodos em comparação com os dois primeiros. Assim, foi realizado uma inspeção das classes e métodos presentes no resultado dos dois primeiros experimentos e ausentes dos resultados desses dois últimos. Para realizar essa verificação, foi utilizada uma ferramenta chamada *Jdeodorant* [Tsantalis and Chatzigeorgiou 2009] que é capaz de identificar *bad smells* em softwares. Essa ferramenta, assim como o *metrics*, é um *plugin* que é instalado na *IDE Eclipse* e executado no projeto desejado por meio desse ambiente de desenvolvimento.

Assim, após passar o *Jdeorant*, nos softwares, foram verificadas as classes que apresentaram os *bad smell God Class* e *Long Method* e comparadas com os resultados

do experimento. Após essa comparação, percebeu-se que os dois primeiros experimentos estavam corretos, a expressão de filtragem proposta para eles realmente identificou as classes que possuíam os sintomas investigados, e a estratégia proposta por [Filó 2014] realmente é válida para a identificação de *God Class* e *Long Method* em softwares.

4.2. Discussão dos Resultados

Após a execução de todos os experimentos, as classes que apresentaram problemas de *God Classes* e *Long Method* foram analisadas, a fim de obter respostas para as questões de pesquisas proposta para este trabalho.

QP1: A aplicação de padrões de projetos elimina a ocorrência de *bad smells* em softwares?

Quando trabalha-se com softwares que fazem uso dos padrões de projeto, espera-se que esses possuam uma boa estrutura e esteja livre de *bad smell*.

Baseado nessa teoria, foram escolhidos alguns softwares para serem avaliados que fazem intenso uso de padrões em sua estrutura. Após a execução dos experimentos realizados, percebeu-se a presença de um índice significativo de *bad smell* nesses softwares.

Analisando a **Tabela 1** e a **Tabela 2**, percebe-se que o *Hibernate* foi o software que apresentou o menor percentual de sintomas *God Class* e *Long Method* comparado com os demais softwares. Porém, quando analisa a quantidade de classes e métodos que apresentaram *bad smell*, observa-se que é um valor bem significativo de classes.

O *Webmail* para a avaliação de *God Class*, possui um percentual maior de classes contaminadas que o*Hibernate*, software que apresentou o menor percentual, porém, quando foi analisado a quantidade real de classes, foi observado que é um valor bem menor que os demais softwares, sendo assim, o software que apresentou menos *bad smell*. O motivo do percentual alto occorre devido ao fato dessa aplicação possuir menos quantidades de classes totais que compõe todo o software.

Para o sintoma de *Long Method*, como já foi falado, o *Hibernate* novamente apresentou o menos percentual de ocorrências, porém, quando analisada a quantidade de métodos que apresentam esses sintomas, percebe-se um número alto de métodos.

Para esse *bad smell*, nenhum software conseguiu apresentou uma quantidade total de ocorrências menor que 100, porém, assim como o *God Class*, novamente o *Webmail* mostrou-se como o software com menor quantidade de ocorrência desses sintomas.

Uma informação importante que foi encontrada durante a análise é que para todas as classes que apresentaram o sintoma de *God Class*, existia pelo menos um método nesta classe com o sintoma de *Long Method*. Os motivos para essa ocorrência serão explicados na resposta para a próxima questão de pesquisa.

Assim, a partir da análise realizada nos experimentos, pode-se afimar que a aplicação de padrões de projeto em um software não elimina a ocorrência de *bad smell* em um software.

QP2: Em que tipos de situações um *bad smell* aparece ou ocorre em um software que aplica padrões de projeto?

Outra análise realizada nos experimentos foi a identificação dos motivos que leva-

ram esses softwares a apresentarem *bad smell*. Ao verificar o código fonte deles, foram listados alguns problemas encontrados e que podem ser as possíveis causas desse sintomas.

Começando pelo *bad smell God Class*, o primeiro problema encontrado na verificação do código fonte das classes com esses sintomas foi a grande quantidade de atributos e métodos existentes nessas classes. A presença desses itens em demasia em uma classe, além de prejudicar a modularidade do software, restringe a inteligência de um sistema a uma classe central do sistema, gerando assim classes Deus.

Outro problema encontrado foi a presença de métodos grandes e complexos nessas classes. Essa complexidade ocorre devido ao uso de muitas estruturas de dados como *if*, *for*, e *while* dentro uma da outra em grande quantidade. Além desse problema prejudicar também a modularidade dessas classes, eles prejudicam a legibilidade e o entendimento do código, tornando assim difícil a realização de manutenções nesses códigos.

A presença de classes aninhadas e mais de uma classe por arquivo foram outros fatores encontrados nesses softwares que além de prejudicar legibilidade e quebrar algumas boas práticas de programação, como os idiomas de programação por exemplo, contribuem para que essas classes centralizem a inteligência apenas em uma única classe.

Para o *bad smell Long Method*, o primeiro problema encontrado foi a extensão de alguns métodos. Muitos métodos possuíam uma grande quantidade de linhas de códigos, o que é uma característica típica desse *bad smell*, e um dos motivos para a identificação desses métodos com esses sintomas.

Outro problema encontrado foi a grande quantidade de blocos de códigos existentes nesses métodos. O uso de estruturas de códigos em grande quantidade como: *switch* e *ifs*, são considerados blocos de código porque de acordo com a condição exigida por essas estruturas, pode ser executada um conjunto de instruções diferentes. A presença dessas estruturas em grande quantidades nos métodos, além de prejudicar a legibilidade e entendimento do código, torna o método muito complexo. Assim, essa elevação de complexidade dos métodos com a presença desses blocos de código, acredita-se que seja a explicação da presença de *bad smell Long Method* de no mínimo uma classe que apresentou *Long Method*.

Por fim, a falta de modularização dos métodos, também foi outro fator que contribuiu para a presença de *Long Method* nos métodos desses software. Quando existem métodos que tendem a ter uma grande quantidade de linhas de códigos deve-se modularizar as funcionalidades deste em pequenas outras funções para evitar justamente suas extensões e garantir uma boa legibilidade e entendimento do código.

5. Ameaças à Validade do Estudo

A principal limitação e ameaça à validade deste estudo, consiste no fato de que os resultados obtidos nesse trabalho não podem ser generalizados, uma vez que foi usada para avaliação uma amostra pequena de softwares.

Para uma generalização, é necessário uma extensão desse estudo e expandir o conjunto de softwares a fim confirmar todos os resultados desse estudo em uma amostra maior. Somente assim, estes resultados poderiam ser generalizados.

6. Conclusão

Neste trabalho foi proposta uma avaliação de softwares com padrões de projeto a fim de avaliar e investigar se essas soluções realmente ajudam no combate a *bad smells* e analisar quais as principais situações que levam esses software a apresentarem ocorrências de *bad smell*, aplicando métricas de softwares.

A partir disso, foi utilizada uma busca na literatura de uma estratégia de detecção para auxiliar na aplicação das métricas nos softwares e identificação dos *bad smells*. A estratégia utilizada nesse trabalho foi proposta por [Filó 2014], por possuir métricas conhecidas pela comunidade acadêmica e existirem ferramentas que auxiliam na coleta.

Definida a metodologia deste trabalho, foram realizados alguns experimentos, onde foram aplicados a estratégia de detecção definida. Em meio a esses experimentos, surgiu uma suspeita em relação a presença de falsos positivos. Para esclarecer essa suspeita, na fase de experimentos, foram realizados alguns experimentos auxiliares, onde pode ser validado a estratégia de detecção e acabar com as suspeitas existentes.

Após a execução dos experimentos foi realizada uma análise dos resultados, onde foi confirmada que a aplicação de padrões de projetos não eliminam a ocorrência de *bad smell*. Ainda foram verificadas possíveis situações de levam a essas ocorrências. Ao analisar essas situações, percebe-se que essas situações estão muito relacionadas com a mal uso dos padrões de projeto e a falta de uso de alguns conceitos propostos pela engenharia de software, como: modularização, legibilidade, dentre outros, em paralelo com a aplicação dos padrões projeto.

Portanto, a conclusão desse trabalho consiste no fato de que os padrões de projeto por si só não resolvem todos os problemas presentes em software. É necessário uma conciliação dessa técnica com várias outras propostas pela engenharia de software a fim de garantir que o software possua uma estrutura que se mantenha flexível e reutilizável.

Referências

Bertrán, I. M. (2009). *Avaliação da qualidade de software com base em modelos uml*. PhD thesis, PUC-Rio.

Cardoso, B. (2015). Análise de coocorrências de anomalias e padrões de projetos em sistemas de software. Master's thesis, Universidade Federal de Minas Gerais, Departamento de Ciência da Computação, Curso de Pós-Graduação em Ciência da Computação, Minas Gerais.

Crespo, Y., López, C., and Marticorena, R. (1999). Relative thresholds: Case study to incorporate metrics in the detection of bad smells. In *QAOOSE 2006 Proceedings*, pages 109–118.

Ferreira, K. A. M. (2011). *Um modelo de predição de amplitude da propagação de modificações contratuais em software orientado por objetos*. PhD thesis, UFMG.

Filó, T. G. S. (2014). Identificação de valores referência para métricas de software orientado por objeto. Master's thesis, Universidade Federal de Minas Gerais, Departamento de Ciência da Computação, Curso de Pós-Graduação em Ciência da Computação, Minas Gerais.

Fowler, M. and Beck, K. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

Gamma, E. (1994). *Design patterns: elements of reusable object-oriented software*. Pearson Education India.

Kerievsky, J. (2005). *Refactoring to patterns*. Pearson Deutschland GmbH.

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

Marinescu, R. (2002). *Em Measurement and Quality in Object-Oriented Design*. PhD thesis, 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. IEEE.

Nunes, H. G. (2014). Identificação de bad smells em softwares a partir de modelos uml. Master's thesis, Universidade Federal de Minas Gerais, Departamento de Ciência da Computação, Curso de Pós-Graduação em Ciência da Computação, Minas Gerais.

Pressman, R. S. (2006). *Engenharia de software*. MacGraw Hill, Rio de Janeiro, 6ª edição edition.

Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., and Noble, J. (2010). The qualitas corpus: A curated collection of java code for empirical studies. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, pages 336–345. IEEE.

Terra, R., Miranda, L. F., Valente, M. T., and Bigonha, R. S. (2013). Qualitas. class corpus: A compiled version of the qualitas corpus. *ACM SIGSOFT Software Engineering Notes*, 38(5):1–4.

Tsantalis, N. and Chatzigeorgiou, A. (2009). Identification of move method refactoring opportunities. *Software Engineering, IEEE Transactions on*, 35(3):347–367.

Tsantalis, N., Chatzigeorgiou, A., Stephanides, G., and Halkidis, S. T. (2006). Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909.

Vale, G. A. (2016). A benchmark-based method to derive metric thresholds. Master's thesis, Universidade Federal de Minas Gerais, Departamento de Ciência da Computação, Curso de Pós-Graduação em Ciência da Computação, Minas Gerais.