Democracia em Xeque

Um Estudo Comparativo sobre Detecção de *Code Smells*

Henrique Gomes Nunes, Lucas Francisco da Matta Vegi, Victor Pezzi Gazzinelli Cruz, Eduardo Figueiredo
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, Minas Gerais, Brasil
{henrique.nunes, lucasvegi, victor.cruz, figueiredo}@dcc.ufmg.br

RESUMO

Existem muitas ferramentas e técnicas diferentes para detecção de *code smells*. Alguns estudos indicam que as formas de detecção de *smells* podem variar significativamente mesmo quando aplicadas em um mesmo contexto. Visando avaliar e compreender se alguma forma de detecção automática de *code smells* provê resultados mais próximos da realidade, nesse artigo é realizada uma investigação quantitativa envolvendo três ferramentas de detecção de *code smells* e um sistema de votação que combina o uso dessas ferramentas. Todas essas formas de detecção de *code smells* foram comparadas com um oráculo proposto por especialistas para classificar uma classe como *God Class*. Por meio do uso de estatística descritiva, os resultados indicaram que a ferramenta JDeodorant apresenta melhor revocação na detecção de *God Class* quando usada individualmente e que o sistema de votação não gera ganhos significativos.

KEYWORDS

code smells, ferramentas de detecção, métricas de software

1 INTRODUÇÃO

Existem muitas pesquisas relacionadas à detecção de *code smells* [5, 18]. *Code smells* são estruturas sub-ótimas de código que podem comprometer a qualidade interna de um sistema [8]. Segundo Sobrinho et al. [18], entre 1990 e 2017, pelo menos 351 estudos diferentes sobre *code smells* foram realizados, sendo a detecção dessas estruturas o foco mais comum entre eles, refletindo 30% do total. Esses estudos propuseram em torno de 80 ferramentas e técnicas diferentes para detectar *code smells* e muitos deles as comparam [5].

Nem sempre ferramentas diferentes se comportam igual em um mesmo contexto. As ferramentas PMD [7] e JDeodorant [6], por exemplo, podem detectar instâncias diferentes de *God Classes* quando analisam o mesmo código [15]. Isso pode ocorrer devido ao uso de métricas e limiares diferentes nas estratégias de detecção. Existem ainda, soluções que realizam a detecção de *smells* via a interseção dos resultados de múltiplas ferramentas, tal que se pelo menos 50% delas acusarem um *smell*, o elemento avaliado é considerado problemático, semelhante à um sistema de votação [3, 9].

Visando compreender melhor o comportamento de diferentes formas de detecção do code smell *God Class*, este artigo compara os resultados de três ferramentas populares de detecção de *code smells* com as detecções feitas utilizando uma estratégia definida por especialistas. Além disso, é utilizado um sistema de detecção baseado no uso combinado de ferramentas de detecção de *code smells* e seus resultados são comparados com o uso isolado das ferramentas de detecção. Essas comparações são feitas utilizando estatística descritiva. Além da precisão, acurácia e revocação dessas formas de detecção, é avaliado o nível de concordância entre essas ferramentas. As análises foram feitas aplicando essas formas de detecção de *God Class* em uma base de dados composta por 1.942 classes de três sistemas de código aberto implementados em Java.

Este estudo possui duas contribuições principais: (1) foi identificada uma ferramenta específica para detecção de *God Class* que se destaca positivamente em relação às demais, sendo a mais indicada para quando o desenvolvedor optar por usar apenas uma ferramenta; e (2) o uso combinado de ferramentas para detectar *code smells* não proporciona resultados significativamente melhores que justifiquem o seu custo mais elevado.

O restante deste artigo está organizado como a seguir. A Seção 2 contextualiza o que são *code smells* e suas formas de detecção. Na Seção 3, são definidas as questões de pesquisa e os métodos usados no estudo. Os resultados são discutidos na Seção 4. A Seção 5 lista as ameaças à validade deste trabalho. Por fim, são apresentados alguns trabalhos relacionados na Seção 6 e conclusões na Seção 7.

2 DETECÇÃO DE CODE SMELLS

Code smells são estruturas sub-ótimas de código que podem prejudicar a manutenção e evolução de software. Este termo foi cunhado por Fowler [8] para nomear essas estruturas, tornando-se amplamente conhecidos entre os desenvolvedores e engenheiros de software. Além de definir o termo, Fowler [8] cataloga 22 *code smells* tradicionais. Embora tenham sido definidos há quase três décadas, *code smells* ainda são muito estudados atualmente [18]. Essas estruturas também são conhecidas por outros sinônimos [20], como anti-padrões, anomalias de código e *bad smells*.

Este trabalho foca na detecção de um *code smell*, definido por Riel [16], chamado *God Class*. Ele é similar à definição de Fowler [8] para *Large Class*. No geral, uma *God Class* é uma classe que concentra muitas responsabilidades, monopolizando o processamento do sistema. Por esse motivo, essa classe se torna grande, complexa e pouco coesa, sendo portanto de difícil legibilidade e manutenção.

Muitos estudos vêm sendo desenvolvidos nos últimos anos buscando detectar, prevenir, remover e entender a extensão dos impactos causados por essas estruturas na qualidade dos códigos. Sobrinho et al. [18] fizeram uma revisão sistemática da literatura sobre *code smells* ambrangendo os artigos publicados entre 1990 e 2017. Os autores encontraram 104 *code smells* documentados neste período. Eles observaram que 67% deles foram ainda pouco estudados. A detecção de *code smells* é o foco principal da maioria dos estudos nessa área [18]. Métricas de código-fonte [13] e estratégias baseadas em aprendizado de máquina [3, 4] são algumas das técnicas usadas para detectar essas estruturas sub-ótimas. Por exemplo, o uso combinado

de métricas de software que quantificam linhas de código, número de módulos, número de elementos, dados acoplados, etc. [19], pode ajudar a identificar *code smells* [8]. Chidamber e Kemerer [2] apresentam métricas comumente utilizadas para detecção de *smells*, como *Weighted Methods Per Class* (WMC), *Depth of Inheritance Tree* (DIT), *Number of Children* (NOC), *Coupling between Objects* (CBO) , *Response For Class* (RFC) e *Lack of Cohesion of Methods* (LCOM).

Por outro lado, existem mais de 80 ferramentas diferentes para detecção de *code smells* [5, 18]. Neste trabalho foram analisadas as seguintes ferramentas, todas elas são *plugins* de código aberto para a IDE Eclipse e analisam programas Java.

JDeodorant [6]. Este *plugin* detecta vários *code smells*, como *God Class*. A abordagem de detecção desta ferramenta é baseada em possibilidades de refatoração.

PMD [7]. Esta ferramenta possibilita quantificar métricas que servem como base para as formas de detecção *smells*.

JSpirit [21]. Este *plugin* auxilia na detecção e priorização de 10 *code smells*. A ferramenta também é capaz de identificar aglomerações [17] de *code smells*.

3 CONFIGURAÇÃO DO ESTUDO

Este estudo foi configurado seguindo as diretrizes propostas por Wohlin et al. [22] para estruturar e conduzir um experimento.

3.1 Escopo

O escopo desta pesquisa foi definido utilizando o *template* do GQM (*Goal Question Metric*), proposto por Basili e Rombach [1]:

> Analisar *ferramentas de detecção de smells* com o propósito de *avaliá-las* com relação à sua *precisão, acurácia e revocação* do ponto de vista de *engenheiros de software* no contexto de *detectar God Class em sistemas de código aberto implementados na linguagem Java.*

3.2 Planejamento e operação

O planejamento dessa pesquisa foi seguido na operação da mesma, conforme mostrado na Figura 1. Todas as etapas dessa instrumentação são detalhadas ao longo dessa seção.

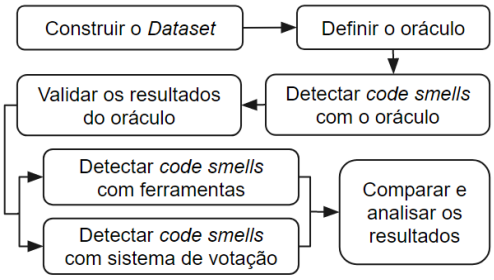

Figura 1: Instrumentação do planejamento e operação

Contexto

Esta pesquisa possui um contexto relativamente barato. Mais especificamente, ao definí-lo utilizando as quatro dimensões propostas por Wohlin et al. [22], temos que esta insvestigação é *offline*, pois não interage de forma direta com os seus sujeitos—classes Java. Além disso ela utiliza *pesquisadores em Engenharia de Software* como especialistas na definição de um oráculo, lida com *problemas reais* (códigos extraídos do GitHub) em um *domínio específico*—detecção do code smell *"God Class"* em 3 projetos Java de código aberto.

Questões de pesquisa

Visando compreender melhor o problema definido pelo escopo e concluir a respeito das ferramentas de detecção de *code smells*, este estudo pretende responder as seguintes questões de pesquisa:

- *RQ1:* Alguma ferramenta é melhor individualmente na detecção de *code smells*?
- *RQ2:* O sistema de votação para detecção de *code smells* é melhor do que as ferramentas?

Seleção das classes a serem analisadas (sujeitos)

Para construir o conjunto de dados que foi utilizado no estudo deste trabalho, foram selecionados três sistemas de código aberto no GitHub. Os critérios de escolha desses sistemas foram: (I) pelo menos 80% da codificação em linguagem Java; (II) última atualização em pelo menos janeiro de 2020; (III) no mínimo 1.000 estrelas—recurso utilizado para avaliar a qualidade dos sistemas; (IV) sistemas com potencial para uso comercial—critério avaliado empiricamente a partir da descrição do projeto no GitHub. A Tabela 1 apresenta a lista dos sistemas selecionados. A primeira coluna refere-se ao nome do projeto, a segunda coluna é a versão do sistema, a terceira coluna mostra a quantidade de classes, a quarta é a quantidade de linhas de código e a quinta a quantidade de método.

Tabela 1: Projetos avaliados

Projetos	Versão	# Classes	LOC	Métodos
JSoup	1.14.2	325	26.809	2.611
JUnit4	4.13.2	1.473	32.973	4.307
NanoHttpd	2.3.1	144	7.106	715

Definição do Oráculo

A fim de avaliar a eficácia das ferramentas que detectam *code smells*, três engenheiros de software especialistas definiram, a partir das características conceituais de *God Class*, quais propriedades representam melhor este problema. Eles chegaram à um consenso de que alta complexidade, baixa coesão e alto acomplamento (principalmente chamadas de outras classes) estão relacionados a *God Classes*. Para representar essas propriedades, eles definiram que as métricas FANIN, HS-LCOM e LOC seriam usadas.

Em sequência foi identificado, para cada um dos três projetos avaliados, quais são os valores de média e de desvio padrão para as métricas selecionadas. As médias e desvios padrões foram calculados separadamente para cada projeto, pois eles possuem tamanhos e contextos distintos. Definidas as métricas e os limiares, a estratégia adotada para detecção de *God Class* foi baseada no *framework* sugerido por Marinescu [13]. Conforme ilustrado pela Figura 2, classes que possuem alta complexidade (medida por altos valores de LOC), alto acoplamento (medida por altos valores de FANIN) e baixa coesão (medida por altos valores de HS-LCOM) simultaneamente, são fortes candidatos a serem *God Classes*. Também foi definido pelos especialistas que o conceito de "Alto" para limiares seria o valor médio da métrica somado ao seu desvio padrão. Essa

estratégia foi utilizada para detectar automaticamente instâncias de *God Class* nos sujeitos selecionados nesse estudo. Os resultados dessa detecção foram posteriormente validados manualmente por três especialistas de forma democrática, em que dois especialistas analisaram individualmente classe apontada como candidata a *God Class*. Se ambos concordassem com essa classificação, a detecção automática foi considerada correta. Essa estratégia de detecção foi nomeada pelos especialistas como *Oráculo*.

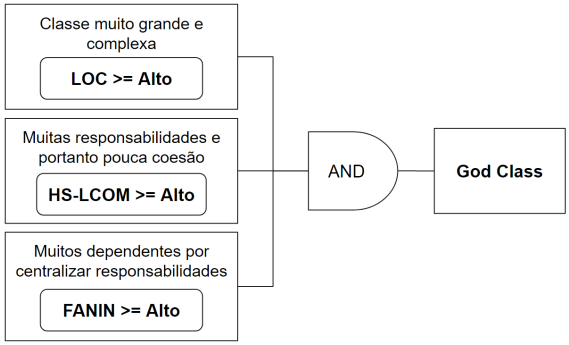

Figura 2: Estratégia para detecção de *God Class* (Oráculo)

Coleta das métricas

A fim de obter os valores das métricas do oráculo, foi necessário utilizar um extrator de métricas para a linguagem Java. As métricas foram extraídas usando o CK Metrics[1]. Essa ferramenta calcula 49 métricas para códigos Java, inclusive as utilizadas no oráculo. Após coletar as métricas, foi construido um *dataset* contendo os valores das três métricas, para cada classe dos projetos avaliados. O *dataset* pode ser consultado em https://doi.org/10.5281/zenodo.6862274.

Escolha das ferramentas de detecção (objetos)

Após extrair as métricas usadas no estudo, foram identificadas as classes consideradas problemáticas (*God Class*) pelas ferramentas de detecção. O estudo optou por selecionar e comparar três ferramentas para detectar *God Class*: PMD [7], JDeodorant [6] e JSpirit [21]. Essa estratégia garantiu que, para cada sujeito, três pontos de vista e estratégias diferentes fossem avaliadas.

Para cada projeto, foram executadas cada uma das três ferramentas de detecção. O resultado dessas detecções encontra-se no *dataset*, em forma de três colunas, uma para cada ferramenta, em que é definido para cada classe se a respectiva ferramenta detectou ou não uma possível *God Class*. Foi criada também uma outra coluna para indicar se uma classe é considerada *God Class* por um sistema de detecção baseado em votação. Para isso, se pelo menos duas ferramentas classificassem uma classe como problemática, democraticamente ela era considerada *God Class*. A interseção dos resultados de ferramentas de detecção de *code smells* foram usados em estudos recentes [3, 9]. O objetivo desse passo foi avaliar se a detecção de *smells* pelo sistema de votação é mais eficaz que de forma individualizada pelas ferramentas (*RQ2*).

[1]https://github.com/mauricioaniche/ck

3.3 Análise e interpretação

Para responder as duas questões de pesquisa deste trabalho, foram calculados e analisados os totais de verdadeiro positivos (VP), verdadeiros negativos (VN), falsos positivos (FP) e falsos negativos (FN) retornados por cada uma das ferramentas de detecção de *code smells* e pelo sistema de votação. Além disso, foram calculados a Precisão (P), Acurácia (A) e Revocação (R) de cada uma dessas formas de detecção de *smells*.

Também foram calculados a mediana e o desvio padrão dos valores VP, VN, FP, FN, P, A e R para as formas de detecção de *smells* utilizadas nessa pesquisa. Isso permitiu responder a *RQ1* e a *RQ2*. Foi calculado ainda o coeficiente Cohen's Kappa [10] por meio do software estatístico Stata[2], possibilitando assim a compreensão do grau de concordância entre as formas de detecção e o oráculo, e também entre cada ferramenta de detecção e o sistema de votação.

4 RESULTADOS

Conforme apresentado na Tabela 2, com exceção à ferramenta JDeodorant, no geral todas as outras formas de detecção de *code smells* utilizadas nesse trabalho detectaram um número de instâncias de *God Class* próximo à 1% do total de classes analisadas. Essses valores são recorrentes em experimentos de detecção de *code smells* [4], o que mostra que o comportamento geral das formas de detecção estão dentro de uma normalidade esperada.

Tabela 2: Instâncias de God Class detectadas por sistema

Sistema	# God Class (% total)				
	Oráculo	PMD	JDeodorant	JSpirit	Votação
JSoup	7 (2,2)	14 (4,3)	16 (4,9)	14 (4,3)	14 (4,3)
NanoHttpd	4 (2,8)	6 (4,2)	10 (6,9)	7 (4,9)	6 (4,2)
JUnit4	13 (0,9)	7 (0,5)	33 (2,2)	7 (0,5)	7 (0,5)
TOTAL	24 (1,2)	27 (1,4)	59 (3,0)	28 (1,4)	27 (1,4)

4.1 RQ1: Alguma ferramenta é melhor individualmente na detecção de *code smells*?

As Tabelas 3 e 4 apresentam respectivamente os desempenhos específicos e geral das ferramentas de detecção de *code smells* utilizadas neste trabalho. Como pode ser observado, tanto no desempenho das ferramentas de detecção em cada sistema analisado (Tabela 3), quanto no desempenho geral das ferramentas de detecção (Tabela 4), as acurácias (A) individuais dessas ferramentas são muito próximas de 100%. Isso pode ser confirmado pela mediana alta (0,98) e pelo desvio padrão baixo (0,1). Embora todas essas ferramentas possuam acurácia alta, elas possuem precisões (P) baixas, evidenciadas por uma mediana de (0,36) e desvio padrão de (0,08).

Considerando que todas essas ferramentas de detecção de *code smells* possuem acurácia alta e precisão baixa, pode-se concluir que elas acertam pouco no alvo (classificações do oráculo), mas possuem resultados que estão dispersos muito próximos desse alvo. Isso mostra que elas são confiáveis na detecção de *God Class*.

Para responder de fato a *RQ1* é importante considerar a revocação (R) dessas ferramentas. Nesse aspecto a ferramenta JDeodorant se destaca em relação às demais, tendo a maior revocação entre elas

[2]https://www.stata.com/

Tabela 3: Detecção de *code smells* por sistema

	Sistemas	VP	VN	FP	FN	P	A	R
PMD	JSoup	4	308	10	3	0,29	0,96	0,57
	NanoHttpd	3	137	3	1	0,50	0,97	0,75
	JUnit4	4	1457	3	9	0,57	0,99	0,31
JDeodorant	JSoup	4	306	12	3	0,25	0,95	0,57
	NanoHttpd	4	134	6	0	0,40	0,96	1,00
	JUnit4	7	1434	26	6	0,21	0,98	0,54
JSpirit	JSoup	3	307	11	4	0,21	0,95	0,43
	NanoHttpd	3	136	4	1	0,43	0,97	0,75
	JUnit4	4	1457	3	9	0,57	0,99	0,31

Tabela 4: Desempenho geral das ferramentas de detecção

Ferramentas	VP	VN	FP	FN	P	A	R
PMD	11	1902	16	13	0,41	0,99	0,46
JDeodorant	15	1874	44	9	0,25	0,97	0,63
JSpirit	10	1900	18	14	0,36	0,98	0,42
Mediana	11	1900	18	13	0,36	0,98	0,46
Desvio Padrão	2,65	15,62	15,62	2,65	0,08	0,1	0,11

(0,63), sendo a única com uma revocação geral acima da mediana (0,46). Além disso, o JDeodorant é superior nesse aspecto nas análises realizadas em todos os três sistemas (sujeitos) dessa investigação. Em especial, no NanoHttpd, a revocação do JDeodorant é de 100% (1,00), ou seja, todas as *God Classes* detectadas pelo oráculo nesse sistema, também foram detectadas pelo JDeodorant.

Embora tenha uma revocação maior, o JDeodorant possui precisão e acurácia menores que as demais ferramentas. Isso se dá devido ao número maior de falsos positivos (FP) detectados por ela. De toda forma, considerando o propósito de uma ferramenta de detecção automática de *code smells*, essa troca é vantajosa. Enquanto a revocação do JDeodorant está muito acima da mediana, os seus valores de precisão e acurácia estão apenas um pouco abaixo da mesma, por isso pode-se afirmar que esta ferramenta é, na prática, a melhor entre as três analisadas para detectar *God Class*.

> ***Resposta RQ1:*** As ferramentas de detecção automática de *code smells* possuem o propósito de alertar o desenvolvedor sobre classes e métodos que podem ter qualidade baixa. Cabe a ele analisar esses códigos manualmente após o filtro da ferramenta. Nesse sentido, as estruturas que realmente prejudicam a qualidade do software não devem escapar desse filtro realizado pelas ferramentas, sendo portanto importante uma revocação alta, sem grandes perdas de precisão e acurácia. Considerando esses fatores, o JDeodorant é a melhor ferramenta individualmente para detectar *God Class*.

4.2 RQ2: O sistema de votação para detecção de *code smells* é melhor do que as ferramentas?

A Tabela 5 apresenta o desempenho do uso combinado das três ferramentas analisadas neste trabalho. Chamado de sistema de votação, essa estratégia classifica uma classe como *God Class* quando pelo menos duas ferramentas a classificam dessa forma. Como pode ser observado, percebemos que o sistema de votação tem um comportamento muito similar às ferramentas de detecção individualmente, possuindo precisão (P), acurácia (A) e revocação (R) similares à mediana desses valores apresentado na Tabela 4. Mais especificamente, o sistema de votação tem os mesmos valores de precisão, acurácia e revocação da ferramenta PMD.

Também foi analisado o grau de concordância das ferramentas e do sistema de votação com o oráculo e entre si. As concordâncias entre o sistema de votação e o oráculo, assim como entre o PMD e o oráculo, são idênticas segundo o coeficiente Cohen's Kappa calculado (0,42). Segundo Landis e Koch [10], essa concordância é moderada e está muito próxima do limite inferior do intervalo dessa classificação, que varia entre (0,41—0,60). A baixa precisão apresentada tanto pelo PMD, quanto pelo sistema de votação, explicam esse moderado grau de concordância com o oráculo.

Para concluir se o comportamento do PMD e do sistema de votação são equivalentes, a concordância entre essas duas formas de detecção de *smells* foi calculada. O coeficiente Cohen's Kappa entre ambas foi muito alto (0,96), portanto a concordância é classificada como quase perfeita e muito próxima do valor máximo (1,00).

Tabela 5: Desempenho do sistema de votação

Sistemas	VP	VN	FP	FN	P	A	R
JSoup	4	308	10	3	0,29	0,96	0,57
NanoHttpd	3	137	3	1	0,50	0,97	0,75
JUnit4	4	1457	3	9	0,57	0,99	0,31
TOTAL	11	1902	16	13	0,41	0,99	0,46

> ***Resposta RQ2:*** O sistema de votação tem um desempenho muito próximo da mediana das três ferramentas de detecção analisadas. Além disso, ele tem um comportamento praticamente idêntico ao da ferramenta PMD. Como o sistema de votação é composto por várias ferramentas de detecção combinadas, o seu uso não é recomendado para detectar *God Class*, pois, para este *smell*, existe uma ferramenta que individualmente proporciona resultados muito similares.

5 AMEAÇAS À VALIDADE

Validade da Conclusão: A principal ameaça desse tipo refere-se ao número de sujeitos e objetos utilizados nessa pesquisa. Apenas uma versão de três sistemas diferentes tiveram as suas classes analisadas. Além disso, somente três ferramentas de detecção de *smells* foram comparadas. Um estudo envolvendo mais sujeitos e objetos é sempre desejável. No entanto, devido às restrições de tempo, este estudo explorou um cenário mais restrito. Para minimizar essa ameaça, sistemas amplamente conhecidos, de tamanhos e finalidades variadas, foram analisados. Além disso, as ferramentas de detecção comparadas foram selecionadas de acordo com as suas relevâncias.

Validade da Construção: A principal ameaça dessa categoria refere-se à construção do oráculo. Como essa definição foi um processo manual e sujeito à influências de experiências pessoais, tem o risco de ser enviesada. Para minimizar essa ameaça, a definição do oráculo foi realizada e validada por três especialistas diferentes, sendo a estratégia final reflexo da concordância de todos eles.

Validade Interna e Externa: Esses riscos dizem respeito à impossibilidade de generalização dos resultados obtidos, uma vez que

o estudo analisou apenas um *smell* (*God Class*), em sistemas de código aberto desenvolvidos com apenas uma linguagem (Java) e as métricas foram coletadas usando apenas uma ferramenta (CK Metrics). Esse risco foi parcialmente mitigado, uma vez que os sistemas analisados possuem diferentes tamanhos, propósitos, domínios e popularidades. Para mitigar esse risco de forma mais ampla, seria necessária uma análise maior, envolvendo mais de tipos de *code smells*, de preferência *smells* também relacionados à métodos e não apenas à classes. Além disso, seria importante explorar sistemas desenvolvidos em outras linguagens de programação.

6 TRABALHOS RELACIONADOS

Olbrich et al. [14] apresentaram uma discussão sobre como os *code smells* podem ser detectados de maneira manual ou automatizada. Como os sistemas explorados por eles eram muito extensos para detecção humana, os autores fizeram uso de heurísticas automatizadas, baseadas em métricas orientadas a objetos, para identificar os *smells God Class* e *Brain Class*. Os sistemas avaliados são bem conhecidos e de código aberto, contendo em média até 400 classes.

Mäntylä [11] e Marinescu [12, 13] também identificaram desvantagens da detecção manual de *smells*, como o fato de ser demorada e não poder ser repetida, apesar de ser considerada mais confiável. Nesses estudos, a eficácia dos métodos de detecção manual foi contrastada com heurísticas de detecção automática. Os resultados indicam que a detecção automática é uma boa alternativa em relação à detecção manual feita via revisão de código, pois as técnicas de detecção automática são tão precisas quanto a detecção manual, têm o mesmo nível de recuperação, mas são mais escaláveis.

Estudos mais recentes, ainda propõem a utilização de um sistema de votação para detecção de *code smells*. Cruz et. al. [3] utilizam cinco ferramentas, em que pelo menos três são capazes de detectar cada um dos *smells* avaliados, a fim de obter dados para construção de um *dataset* de classes e métodos afetados por *smells*. Ichtsis et. al. [9] propõem um *plugin* que automaticamente identifica a interseção de resultados de seis ferramentas de detecção de *smells*.

7 CONCLUSÕES

Este trabalho realizou a detecção de *God Class* em uma amostra de 1.942 classes de três sistemas de código aberto Java. Segundo o oráculo definido pelos especialistas, 24 classes da amostra (1,2%) são *God Class*. Foi conduzida uma investigação quantitativa visando compreender e avaliar o comportamento de diferentes formas de detecção de *code smells*. Três ferramentas de detecção e uma estratégia baseada um sistema de votação foram comparadas com o oráculo. Considerando a precisão, acurácia e revocação dessas ferramentas, conclui-se que o JDeodorant é a melhor ferramenta entre as três para ser usada individualmente na detecção de *God Class*, pois a sua revocação é maior, sem perda significativa de precisão e acurácia.

Com relação sistema de votação, conclui-se que ele não gera ganhos significativos, pois a sua precisão, acurácia e revocação são muito próximos da mediana das três ferramentas de detecção. Além disso, o nível de concordância do PMD, segundo o coeficiente Cohen's Kappa [10], é quase perfeito em relação ao sistema de votação. Dessa forma, o custo maior gerado pelo sistema de votação não se justifica nesse caso.

Como trabalhos futuros, será realizado um estudo mais amplo, possibilitando generalizar os resultados. Para isso uma amostragem maior e mais heterogênea será utilizada, composta por mais sistemas, implementados em linguagens diferentes. Além disso, um escopo mais amplo será utilizado, detectando outros tipos de *smells*.

8 AGRADECIMENTOS

Esta pesquisa foi parcialmente apoiada por agências de fomento brasileiras: CNPq, CAPES e FAPEMIG.

REFERÊNCIAS

[1] V. Basili and H. Rombach. 1988. The TAME project: towards improvement-oriented software environments. *IEEE Trans. on Software Engineering* 14, 6 (1988), 758–773.
[2] S. Chidamber and C. Kemerer. 1994. A metrics suite for object oriented design. *IEEE Trans. on Software Engineering* 20, 6 (1994), 476–493.
[3] D. Cruz, A. Santana, and E. Figueiredo. 2020. Detecting bad smells with machine learning algorithms: an empirical study. In *3rd Int. Conf. on Technical Debt*. 31–40.
[4] D. Di Nucci, F. Palomba, D. Tamburri, A. Serebrenik, and A. De Lucia. 2018. Detecting code smells using machine learning techniques: are we there yet?. In *IEEE 25th Int. Conf. on Software Analysis, Evolution and Reengineering*. 612–621.
[5] E. Fernandes, J. Oliveira, G. Vale, T. Paiva, and E. Figueiredo. 2016. A review-based comparative study of bad smell detection tools. In *20th Int. Conf. on Evaluation and Assessment in Software Engineering (EASE)*. 1–12.
[6] M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou. 2011. JDeodorant: identification and application of extract class refactorings. In *2011 33rd Int. Conf. on Software Engineering (ICSE)*. 1037–1039.
[7] F. Fontana, M. Zanoni, A. Marino, and M. Mäntylä. 2013. Code smell detection: towards a machine learning-based approach. In *IEEE 29th Int. Conf. on Software Maintenance (ICSM)*. 396–399.
[8] M. Fowler and K. Beck. 1999. *Refactoring: improving the design of existing code* (1 ed.). Addison-Wesley.
[9] A. Ichtsis, N. Mittas, A. Ampatzoglou, and A. Chatzigeorgiou. 2022. Merging Smell Detectors: Evidence on the Agreement of Multiple Tools. In *5th Int. Conf. on Technical Debt*. 61–65.
[10] J. Landis and G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (1977), 159–174.
[11] M.V. Mantyla, J. Vanhanen, and C. Lassenius. 2004. Bad smells - humans as code critics. In *20th IEEE Int. Conf. on Software Maintenance*. 399–408.
[12] R. Marinescu. 2001. Detecting design flaws via metrics in object-oriented systems. In *39th Int. Conf. and Exhibition on Technology of Object-Oriented Languages and Systems*. 173–182.
[13] R. Marinescu. 2005. Measurement and quality in object-oriented design. In *21st IEEE Int. Conf. on Software Maintenance (ICSM)*. 701–704.
[14] S. Olbrich, D. Cruzes, and D. Sjøberg. 2010. Are all code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems. In *26th IEEE Int. Conf. on Software Maintenance (ICSM)*. 1–10.
[15] G. Rasool and Z. Arshad. 2015. A review of code smell mining techniques. *Journal of Software: Evolution and Process* 27, 11 (2015), 867–895.
[16] A. Riel. 1996. *Object-Oriented Design Heuristics*. Addison-Wesley.
[17] A. Santana, D. Cruz, and E. Figueiredo. 2021. An exploratory study on the identification and evaluation of bad smell agglomerations. In *36th Annual ACM Symposium on Applied Computing (SAC '21)*. 1289–1297.
[18] E. Sobrinho, A. De Lucia, and M. Maia. 2021. A systematic literature review on bad smells–5 w's: which, when, what, who, where. *IEEE Trans. on Software Engineering* 47, 1 (2021), 17–66.
[19] T. A Soliman, A. El-Swesy, and S. Ahmed. 2010. Utilizing CK metrics suite to UML models: A case study of microarray midas software. In *7th Int. Conf. on Informatics and Systems*. 1–6.
[20] L. F. M. Vegi and M. T. Valente. 2022. Code smells in Elixir: early results from a grey literature review. In *30th Int. Conf. on Program Comprehension (ICPC)*. 1–5.
[21] S. Vidal, H. Vazquez, J. Diaz-Pace, C. Marcos, A. Garcia, and W. Oizumi. 2015. JSpIRIT: a flexible tool for the analysis of code smells. In *34th Int. Conf. of the Chilean Computer Science Society*. 1–6.
[22] C. Wohlin, P. Runeson, M. Hst, M. Ohlsson, B. Regnell, and A. Wessln. 2012. *Experimentation in Software Engineering*. Springer.