Avaliação empírica da geração automatizada de testes de software sob a perspectiva de Test Smells

Tássio Guerreiro Antunes Virgínio[1], **Ivan Machado**[1]

[1]Instituto de Computação – Universidade Federal da Bahia (UFBA)
Av. Adhemar de Barros, s/nº – Ondina – 40170-110 – Salvador – BA – Brasil

tassiovirginio@gmail.com, ivan.machado@ufba.br

Resumo. *O teste de software é uma atividade-chave para o desenvolvimento de software de qualidade, sendo tão ou mais custoso do que o desenvolvimento do código de produção. De modo a reduzir os custos de projetos de software, o uso de ferramentas de geração automatizada de testes, tais como Randoop e Evosuite tem sido fortemente encorajado. No entanto, é necessário obter evidências sobre como o uso dessas ferramentas afeta a qualidade dos testes. Neste sentido, o presente estudo apresenta uma avaliação empírica da qualidade de testes gerados automaticamente, sob a perspectiva de test smells. O estudo contemplou o desenvolvimento de uma ferramenta open source de coleta e análise automatizada de test smells, a* `JNose Test`*, no desenvolvimento da ferramenta foi realizado estudo utilizando 11 projetos open source para verificar a relação entre test smells e a cobertura dos testes, e posteriormente no segundo estudo que avaliou a qualidade de testes gerados automaticamente para 21 projetos open source. No primeiro estudo encontramos relações fortes entre métricas de cobertura e test smells, e no segundo estudo os resultados indicam uma alta difusão dos test smells e co-ocorrências entre diferentes tipos de test smells nos projetos avaliados. Além disso, foi identificada uma alta difusão de test smells nos códigos de teste gerados pelas ferramentas Evosuite e Randoop que, frequentemente, estão correlacionados.*

Abstract. *Software testing is a crucial activity for quality software development, but it is usually more costly than production code development. To reduce software projects' costs, automated test generation tools such as Randoop and Evosuite have been strongly encouraged. However, evidence is needed on how the use of these tools affects test quality. In this sense, this study presents an empirical evaluation of the quality of automatically generated tests from the perspective of test smells. The study included developing an open-source automated collection and analysis of test smells tool, the* `JNose Test`*. In the development of the tool, a study was carried out using 11 open source projects to verify the relationship between test smells and test coverage, and later in the second study that assessed the quality of automatically generated tests for 21 open source projects. In the first study, we found strong relationships between coverage metrics and test smells, and in the second study, the results indicate a high diffusion of test smells and co-occurrences between different types of test smells in the evaluated projects. Furthermore, we identified a high diffusion of test smells in the test codes generated by the Evosuite and Randoop tools, which are often correlated.*

1. Introdução

Um dos principais problemas que afetam a qualidade de projetos de software diz respeito aos códigos mal elaborados. Apesar das boas intenções, códigos defeituosos são responsáveis por até 45% do tempo de inatividade dos sistemas computacionais. Estima-se em mais de 2 trilhões de dólares o impacto da baixa qualidade de software na economia dos EUA (dados de 2020) [CISQ 2021].

De modo a validar a qualidade dos projetos de software, a atividade de teste de software continua a ser amplamente utilizada para antecipar-se a problemas, identificando-os e corrigindo-os antes da entrega final do software, e assim reduzindo eventuais custos com reparos futuros. Com o intuito de reduzir o esforço e o custo na construção de testes, diversas ferramentas de geração automatizada tem sido propostas e amplamente utilizadas [Somé and Cheng 2008]. No entanto, a construção de testes também é passível de erros. [van Deursen et al. 2001] definiu o termo *test smells* para representar as más escolhas no *design* dos testes, que tem características sintomáticas e podem acarretar em diminuição na qualidade dos sistemas de software.

Diante da emergente necessidade de investigar o impacto da presença de *test smells* na qualidade de software, a presente investigação tem como objetivo avaliar a qualidade dos códigos de teste gerados com suporte ferramental, considerando a difusão dos *test smells*. A utilização dessas ferramentas é essencial para reduzir o esforço e garantir a eficácia dos testes [Somé and Cheng 2008]. Consequentemente, a verificação da qualidade dos códigos de testes gerados por essas ferramentas também deve ser considerada. A partir desse contexto definimos a seguinte pergunta de pesquisa:

De que forma os geradores de teste influenciam a qualidade dos testes de unidade na perspectiva dos *test smells*?

A pesquisa envolveu a realização de dois estudos empíricos. O primeiro, que consistiu no desenvolvimento da *JNose Test*, apresenta as correlações entre os *test smells* e as métricas de cobertura de código [Virgínio et al. 2019]. As métricas de cobertura são utilizadas para avaliar a qualidade dos testes por meio da mensuração da cobertura de elementos estruturais, como funções, instruções, ramificações e condicionais [Gopinath et al. 2014].

O segundo estudo apresenta uma investigação de caráter exploratório, cujo propósito é avaliar a qualidade dos códigos de testes gerados de forma automática e pré-existentes, a partir da seleção de projetos *open source* disponíveis no repositório GitHub, e que atendam aos seguintes critérios: projetos Maven, desenvolvidos na linguagem Java, e que tenham classes de testes que utilizem a *framework* JUnit. Foram utilizados vinte e um projetos selecionados de forma aleatória os quais atendem a esses critérios. Em seguida, foram gerados os testes de unidade a partir das ferramentas EvoSuite e Randoop, com base nos códigos fontes dos projetos selecionados.

A partir dos códigos de testes gerados foi executada a ferramenta de detecção de *test smells*, `JNose Test`[1] que foi proposta e desenvolvida neste estudo. A ferramenta proveu suporte à geração automatizada de métricas de qualidade, dentre elas, dados quan-

[1] `https://github.com/arieslab/jnose`

titativos de *test smells* encontrados. A partir dos dados coletados, verifica-se a qualidade dos testes de unidade com base nos *test smells*. Em contrapartida, a qualidade das ferramentas de geração de testes também é avaliada.

2. Referencial Teórico

2.1. Test Smells

Test smells são más práticas adotadas no design e implementação dos códigos de teste [van Deursen et al. 2001]. A presença de test smells pode servir como um indicador de software de baixa qualidade, uma vez que podem reduzir a compreensão, legibilidade e manutenibilidade do código de teste [Greiler et al. 2013]. Diversos tipos de test smells foram definidos na literatura [van Deursen et al. 2001, Meszaros et al. 2003, Peruma et al. 2019]. A presente investigação considerou 19 tipos de test smells:

- **Assertion Roulette (AR)** - Ocorre quando há muitas assertivas em um método de teste sem explicação para cada uma;
- **Conditional Test Logic (CTL)** - Ocorre quando um teste possui estruturas condicionais ou repetitivas;
- **Constructor Initialization (CI)** - Ocorre quando um método de teste implementa um construtor;
- **Default Test (DT)** - Isso ocorre quando um IDE cria classes de teste padrão como um exemplo para desenvolvedores que devem ser removidos ou renomeados.
- **Dependent Test (DpT)** - Ocorre quando um teste falha e vários outros falham sem motivo aparente. Isso indica que os testículos dependem dos efeitos colaterais de um teste realizado anteriormente.
- **Duplicate Assert (DA)** - Ocorre quando um método de teste testa a mesma condição várias vezes;
- **EagerTest (ET)** - Isso ocorre quando um método de teste verifica vários métodos do objeto a ser testado. Isso torna os testes dependentes uns dos outros e difíceis de manter;
- **Empty Test (EpT)** - Ocorre quando os métodos de teste não contêm instruções executáveis;
- **Exception Catching Throwing (ECT)** - Ocorre quando o resultado do método de teste depende explicitamente do método de produção que lança uma exceção;
- **General Fixture (GF)** - Ocorre quando um programador escreve um método *setUp():* para ser executado antes de cada método de teste para criar um ambiente para os testes serem executados;
- **Ignored Test (IgT)** - Ocorre quando os desenvolvedores ignoram os métodos de teste, resultando em sobrecarga em relação ao tempo de compilação e em um aumento na complexidade do código e no tempo de compreensão;
- **Lazy Test (LT)** - Isso ocorre quando vários métodos de teste verificam o mesmo método. Esses testes geralmente são significativos apenas quando são considerados em conjunto;
- **MagicNumber Test (MNT)** - Ocorre quando declarações assert em um método de teste contêm literais numéricos (ou seja, números mágicos) como parâmetros. Números mágicos não indicam o significado/propósito do número. Portanto, eles devem ser substituídos por constantes ou variáveis, fornecendo assim um nome descritivo para a entrada;

- **Mystery Guest (MG)** - Ocorre quando um teste autônomo deixa de ser autônomo porque usa recursos externos. Consequentemente, não há informações suficientes para entender o recurso em teste, o que dificulta o uso deste teste, principalmente quando os testes são considerados uma documentação chave do projeto;
- **Print Declaration (PS)** - Ocorre quando as instruções de impressão são redundantes porque os testes de unidade são executados como parte de um script automatizado. Portanto, pode consumir recursos ou aumentar o tempo de execução;
- **Redundant Assertion (RA)** - Isso ocorre quando os métodos de teste sempre contêm declarações de asserção verdadeiras ou falsas. Um teste destina-se a retornar um resultado binário, independentemente de o resultado pretendido estar correto ou não, e não deve retornar a mesma saída, independentemente da entrada;
- **Resource Optimism (RO)** - Ocorre quando o código de teste faz suposições otimistas sobre a existência ou ausência de um determinado recurso externo. O estado desses recursos externos pode causar um comportamento não determinístico nos resultados do teste;
- **Sensitive Equality (SE)** - Ocorre quando um método de teste contém uma verificação de igualdade usando o método *toString()*. Depende de detalhes como vírgulas e espaços. Portanto, quando o método é alterado, o teste pode falhar;
- **Sleepy Test (ST)** - Ocorre quando os desenvolvedores precisam pausar a execução de instruções em um método de teste por um determinado período e continuar sua execução logo em seguida;
- **Verbose Test (VT)** - Ocorre quando os testes possuem muitos códigos ou estruturas condicionais, o que torna difícil afirmar sua precisão;
- **Unknown Test (UT)** - Ocorre quando um método de teste não possui uma condição de asserção. O teste sempre será válido, sem resultar em uma exceção;

2.2. Ferramentas de geração de teste

As ferramentas de geração automatizada de testes possibilitam a repetibilidade, previsibilidade e eficiência do código de teste [Garousi and Kucuk 2018]. Dentre as ferramentas de código-fonte aberto, i.e., *open source*, que visam a geração automática de testes na linguagem Java, destacam-se a EvoSuite e a Randoop, utilizadas neste estudo. A ferramenta EvoSuite (*Automatic Test Suite Generation for Java*) implementa uma abordagem híbrida que combina testes baseados em busca e execução dinâmica simbólica para gerar as suítes de teste [Fraser and Arcuri 2014]. Por sua vez, o Randoop (*RANDom tester for Object-Oriented Programs*) utiliza o framework JUnit para gerar testes de unidade randomicamente. Ele toma como entrada o feedback das execuções anteriores para evitar redundância de dados [Pacheco and Ernst 2007].

3. JNose Test - Java Test Smell Detector

Neste estudo, desenvolvemos a `JNose Test`, ferramenta que visa detectar *test smells* e calcular métricas estruturais de testes. Ela foi utilizada para analisar a qualidade dos conjuntos de teste [Virgínio et al. 2019, Virgínio et al. 2020b, Virgínio et al. 2020a]. Ele constrói uma AST (*Abstract Syntax Tree*) a partir do teste que é analisado por cada componente de detecção, e aplica as regras de detecção de *test smells*.

As regras de detecção inicialmente foram utilizadas do `tsDetect` [Bavota et al. 2015, Grano et al. 2019, Spadini et al. 2020], a ferramenta identifica

vinte e um tipos de test smells. Ela retorna um valor booleano para indicar se um determinado test smell aparece no código de teste. Reutilizamos as regras de detecção *tsDetect*, pois apresenta uma pontuação de precisão que varia de 85% a 100% e uma pontuação de recall de 90% a 100% [Peruma et al. 2019]. No entanto, nós estendemos a ferramenta em vários aspectos. Dentre as funcionalidades implementadas na `JNose Test`, com destaque para as melhorias em relação ao *tsDetect*: Interface gráfica; Detecção de projetos em subdiretórios; Automação do fluxo de entrada e saída de dados em cada etapa; Coleta automatizada de métricas de código e de cobertura; Quantitativo de *test smells*; Resultado unificado dos projetos selecionados; Utilização de *threads* para cada detecção de classe de teste, projeto e processo de cobertura, com o intuito de melhorar a velocidade de processamento; Retorna a classe, o método e o número da linha onde ocorreu o test smells; Os códigos de detecção de test smells foram melhorados; Os códigos de detecção das classes de teste foram melhorados; Os códigos de detecção das classes de produção foram melhorados. Além disso, a ferramenta apoia a análise do conjunto de testes por meio de várias versões do projeto, minerando o repositório Git para fornecer informações sobre quando e por quem os test smells foram adicionados. A `JNose Test` evoluiu durante a elaboração da dissertação e após a defesa do mestrado. O artigo "JNose: Java Test Smell Detector", publicado na trilha de ferramentas do SBES 2020 recebeu o prêmio de melhor artigo [Virgínio et al. 2020b]. Uma versão estendida do estudo encontra-se em avaliação [Virgínio et al. 2021].

3.1. Arquitetura

A JNose Test foi desenvolvida na linguagem Java e seu funcionamento, como na ferramenta do estudo base, tem seu foco em código Java. Para gerenciar o projeto da ferramenta, foi utilizado o Apache Maven[2], responsável pelo gerenciamento das dependências de bibliotecas, compilação e execução do projeto. A implementação da JNose Test utiliza as seguintes bibliotecas:

- **JNose-Core**[3]: Biblioteca gerada com base no núcleo do JNose Test, o qual foi removido e refatorado em uma biblioteca independente, que pode ser utilizada por qualquer projeto Java, distribuído de forma *open source* e disponível no repositório do Github.
- **Apache Wicket**[4]: *Framework* para o desenvolvimento de aplicações Web, desenvolvido na linguagem Java. A versão 8 é a mais estável, as aplicações são em Java, HTML5 e CSS3. Na JNose Test, o Apache Wicket foi utilizado para o desenvolvimento da interface web;
- **JaCoCo**[5]: Biblioteca de cobertura de código aberto para Java, criada pela equipe do EclEmma com base nas lições aprendidas do uso e na integração de bibliotecas existentes por muitos anos. Neste projeto, a biblioteca JaCoCo foi utilizada para gerar a cobertura de testes na aplicação;
- **JavaParser**[6]: É um conjunto simples e leve de ferramentas para gerar, analisar e processar o código Java. Atualmente, é utilizado em diversos projetos comerciais

[2] `https://maven.apache.org/`
[3] `https://github.com/arieslab/jnose-core`
[4] `https://wicket.apache.org/`
[5] `https://www.eclemma.org/jacoco/`
[6] `https://javaparser.org/`

e de código aberto. Neste projeto, o JavaParser foi utilizada para a detecção dos *test smells*.

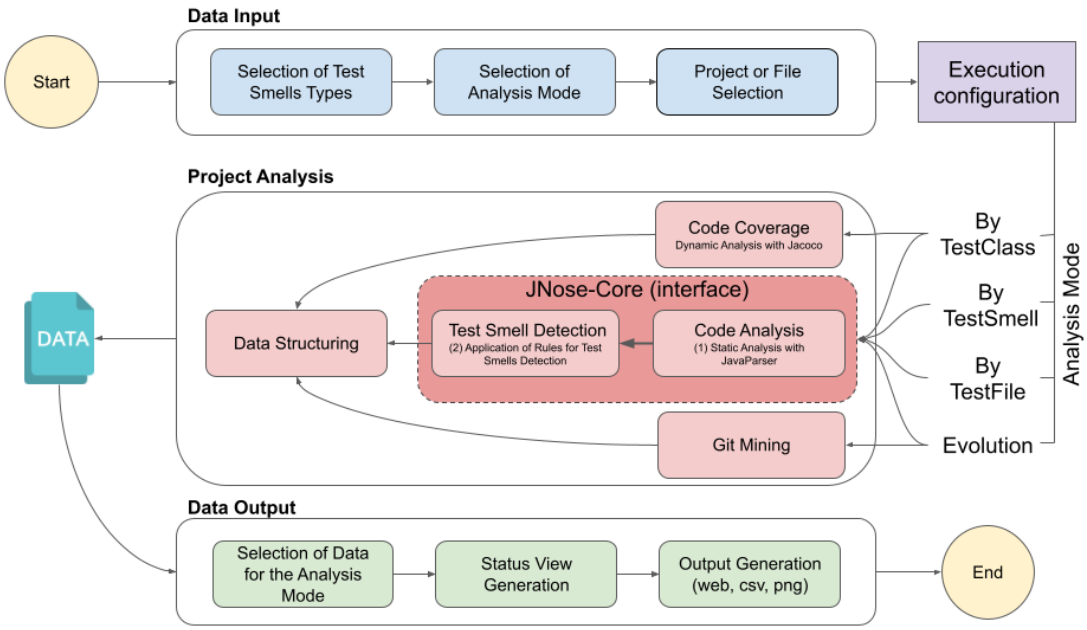


Figura 1. Visão geral esquemática da ferramenta de teste JNose e seus principais recursos

A operação `JNose Test` envolve três processos-chave (Figura 1): (i) **Data Input**, recebe as configurações para a execução da ferramenta, ou seja, a lista de tipos de *test smells*, modo de análise (*By TestClass*, *By TestSmell*, *By TestFile*, e *Evolution*), e o projeto a ser analisado; (ii) **Project Analysis**, chama o `JNose-Core`, uma API para realizar a análise do projeto de acordo com o modo de análise selecionado; e (iii) **Data Output**, mostra o status da execução e os resultados da análise.

4. Estudo exploratório: Correlação entre *Test Smells* e Cobertura

Durante o desenvolvimento da JNose Test foi realizado um estudo para analisar a correlações entre os *test smells* e as métricas de cobertura de código [Virgínio et al. 2019]. De modo a investigar se há correlação entre as métricas de cobertura e a presença de *test smells*, foi realizado um estudo quantitativo para responder a questão de pesquisa a seguir:

Existe correlação entre *test smells* e as métricas de cobertura de código?

A partir da questão de pesquisa definimos a seguinte hipótese:

- Hipótese nula (H_0): *Não há correlação entre as métricas de cobertura e os test smells.*
- Hipótese alternativa (H_1): *Há correlação entre as métricas de cobertura e os test smells.*

Para responder essa questão de pesquisa, foram analisados 11 projetos - listados da Tabela 1. No estudo, buscamos a ocorrência de 21 tipos de *test smells* e 10

métricas de cobertura. Foram utilizadas medidas de tendência central e medidas de dispersão para descrever e caracterizar as amostras dos dados. Em seguida, foi verificado se os dados da amostra seguem a distribuição normal, utilizando o teste de Kolmogorov-Smirnov [Wood and Altavela 1978]. O resultado da análise apresentou uma distribuição não-normal, portanto, o teste de correlação de Spearman foi escolhido para testar as hipóteses definidas.

Tabela 1. Resumo dos projetos de código aberto analisados

Projeto	Versão	LOC	JUnit LOC
Joda Time	2.9.7	29,272	55,554
Apache Commons Text	1.6	9,526	13,467
Apache Commons Lang	3.9	28,006	47,833
Apache Commons Email	1.5.1	2,947	3,989
Apache Commons CSV	1.6	1,787	4,783
Apache Commons Colletion	4.3	29,309	34,200
Apache Commons Codec	1.11	8,392	12,040
Apache Commons DBCP	2.6.0	14,458	16,846
Apache Commons Math	3.4.1	86,481	89,016
Apache Commons Pool	2.6.2	5,532	8,813
Apache Commons DbUtils	1.7	3,101	3,789

Tabela 2. Correlação entre *test smells* e as métricas de cobertura

Test Smell	BC	BM	CC	CM	IC	IM	LC	LM	MC	MM
AR	0,42★	0,29❖	0,55★	0,25❖	0,41★	0,18	0,46★	0,18	0,59★	0,06
CI	0,02	0,02	0,15	0,02	-0,01	-0,01	0,05	0,00	0,26❖	-0,02
CTL	0,25❖	0,27❖	0,20❖	0,26❖	0,29❖	0,26❖	0,26❖	0,24❖	0,15	0,17
DA	0,47★	0,35❖	0,52★	0,33❖	0,47★	0,26❖	0,50★	0,25❖	0,49★	0,14
ECT	0,24❖	0,21❖	0,34❖	0,19	0,29❖	0,19	0,32❖	0,19	0,38❖	0,09
ET	0,51★	0,32❖	0,67◆	0,29❖	0,51★	0,23❖	0,57★	0,22❖	0,70◆	0,11
GF	0,15	0,15	0,18	0,18	0,16	0,23❖	0,18	0,23❖	0,16	0,19
LT	0,56★	0,34❖	0,65◆	0,28❖	0,52★	0,23❖	0,57★	0,21❖	0,62◆	0,07
MNT	0,40❖	0,26❖	0,44★	0,21❖	0,40❖	0,17	0,41★	0,14	0,43★	0,06
RA	0,05	0,05	0,16	0,02	0,11	0,00	0,13	0,00	0,22❖	-0,04
SE	0,28❖	0,14	0,38❖	0,09	0,28❖	0,02	0,31❖	0,03	0,40❖	-0,06
UT	0,14	0,15	0,16	0,11	0,28❖	0,09	0,25❖	0,09	0,14	0,02

labels: (▲) muito forte, (◆) forte, (★) moderada, (❖) fraca, ( ) sem correlação.

Foi utilizado o teste de correlação de Spearman com 21 test smells e 10 métricas de cobertura, o que resultou em 210 correlações calculadas. Foram utilizadas as seguintes métricas de cobertura: *Branch Covarege (BC)*, *Branch Missed (BM)*, *Complexity Covarege (CC)*, *Complexity Missed (CM)*, *Instruction Covarege (IC)*, *Instruction Missed (IM)*, *Line Covarege (LC)*, *Line Missed (LM)* e *Method Covarege (MC) Method Missed (MM)*. A Tabela 2 apresenta a correlação entre os *test smells*, que varia de correlação fraca a forte, e a cobertura de teste. Os test smells *Conditional Test Logic* (CTL), *Execption Catching Throwing* (ECT), *Magic Number Test* (MNT), *Sensitive Test* (SE) e *Unknown Test* (UV) tiveram correlação fraca ou moderada com as métricas de cobertura, principalmente nas métricas *Branch Coverage*, *Instruction Coverage*, e *Line Coverage*. O *test smells*

Assertion Roulette (AR), *Duplicate Assert* (DA), *Eager Test* (ET) e *Lazy Test* (LT) tem correlações de moderada a forte com as métricas de cobertura *Branch Coverage*, *Complexity Coverage*, *Instruction Coverage*, *Line Coverage* e *Method Coverage*. Ou seja, no total obtivemos 4 correlações fortes, 17 correlações moderadas e 42 correlações fracas.

Nos projetos analisados, não identificamos correlação entre sete *test smells*: *Empty Test* (EpT), *Ignore Test* (IgT), *Mystery Guest* (MG), *Print Declaration* (PS), *Resource Optimism* (RO), *Sleep Test* (ST) e *Verbose Test* (VT). Além disso, os *test smells Dependent Test* (DpT) e *Default Test* (DT) não estão presentes nos projetos analisados.

Esse primeiro estudo teve um caráter exploratório, a partir do qual foi possível verificar as correlações existentes entre *test smells* e as métricas de cobertura. Como resultado, foi identificada a necessidade de prover ferramental de apoio para a coleta e análise de dados. Neste sentido, propusemos o desenvolvimento da `JNose Test`, com objetivo de automatizar e facilitar o levantamento dos dados. Com a `JNose Test` foi possível levantar novos *gaps* na área, que serviram de base para a concepção e execução do estudo principal apresentado nesse trabalho.

5. Avaliação da qualidade de testes gerados automaticamente

O estudo contemplou a avaliação empírica da qualidade do código de teste gerado pelas ferramentas Randoop e Evosuite, comparando-as com os códigos de teste existentes nos projetos *open source* avaliados. O estudo avaliou 21 projetos (Tabela 3) desenvolvidos na linguagem Java, selecionados de forma aleatória a partir do repositório do GitHub.

Tabela 3. Projetos Selecionados

#	Projeto	nº classes	nº teste	URL (https://github.com)
1	Algorithms	86	80	/pedrovgs/Algorithms.git
2	sling-org-apache-sling-api	116	15	/apache/sling-org-apache-sling-api.git
3	blade	191	84	/lets-blade/blade.git
4	checkstyle	377	1395	/checkstyle/checkstyle.git
5	commons-collections	326	217	/apache/commons-collections.git
6	commons-jexl	102	63	/apache/commons-jexl.git
7	DiskLruCache	3	2	/JakeWharton/DiskLruCache.git
8	failsafe	56	47	/jhalterman/failsafe.git
9	fastjson	178	2642	/alibaba/fastjson.git
10	GCViewer	155	67	/chewiebug/GCViewer.git
11	HanLP	472	155	/hankcs/HanLP.git
12	mybatis-generator-gui	31	2	/zouzg/mybatis-generator-gui.git
13	Mybatis-PageHelper	48	66	/pagehelper/Mybatis-PageHelper.git
14	lettuce-core	643	349	/lettuce-io/lettuce-core.git
15	reflections	36	12	/ronmamo/reflections.git
16	spark	99	81	/perwendel/spark.git
17	uid-generator	22	2	/baidu/uid-generator.git
18	xmlgraphics-commons	314	73	/apache/xmlgraphics-commons.git
19	ysoserial	59	18	/frohoff/ysoserial.git
20	commons-csv	12	22	/apache/commons-csv.git
21	commons-text	101	80	/apache/commons-text.git

As seguintes questões de pesquisa foram investigadas:

QP1 Existe diferença na qualidade do código de teste gerado pelas ferramentas Randoop e Evosuite?

QP2 Existe diferença na qualidade do código de teste gerado pela ferramenta Randoop e os códigos de testes pré-existentes?

QP3 Existe diferença na qualidade do código de teste gerado pela ferramenta Evosuite e os códigos de testes pré-existentes?

As hipóteses definidas para o estudo estão listadas a seguir:

- **Hipótese nula** (H_0): Não há diferença entre a quantidade de *test smells* detectados no códigos gerados pelas ferramentas Randoop, Evosuite e os códigos existentes.
- **Hipótese alternativa** (H_1): Há diferença entre a quantidade de *test smells* detectados no códigos gerados pelas ferramentas Randoop, Evosuite e os códigos existentes.

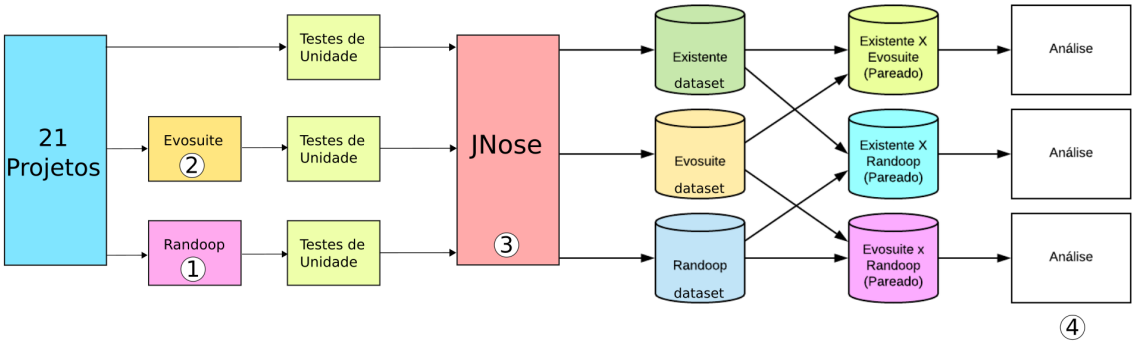


Figura 2. Fluxo do Experimento

A Figura 2 apresenta uma visão geral do estudo. Nas etapas 1 e 2 foram executados os geradores de testes. Logo após, com os 3 pacotes de testes foi executada a `JNose Test` (etapa 3), gerando 3 pacotes de dados (test smells) que foram pareados e utilizados para realizar a Análise Estatística (Etapa 4).

O *dataset*[7] final gerou 5.445 linhas, que correspondem a 4.542.303 linhas de código de teste. O Teste de Wilcoxon foi utilizado para validar as hipóteses. Com um *p-value* menor que 0.05, foi possível refutar a hipótese nula. Além de testar as hipóteses, o estudo considerou a análise de co-ocorrências dos *test smells* em cada pacote de teste. Para tal, aplicamos o Teste de Spearman.

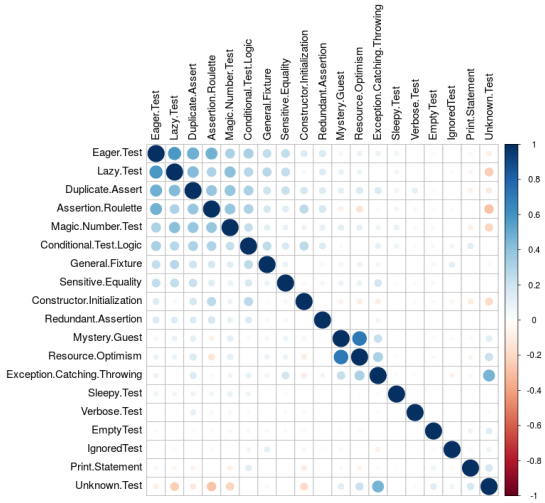
6. Discussão, Resultados e Implicações

Há uma diferença significativa entre as métricas estruturais, onde os testes da ferramenta Randoop apresentaram aproximadamente cinco vezes o tamanho dos testes da ferramenta Evosuite. Como mencionado no estudo de [Palomba et al. 2016], os *test smells* têm forte correlação com características estruturais. Os resultados do estudo empírico realizado no contexto dessa investigação possibilitou identificar uma maior dispersão do que aquelas apresentadas em estudos anteriores [Bavota et al. 2015, Palomba et al. 2016].

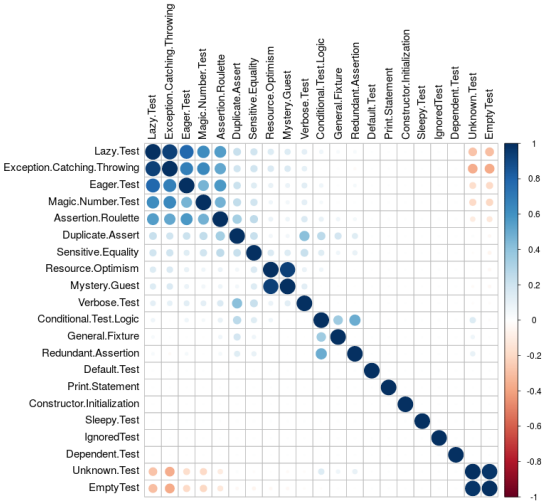
Observamos pelo menos um *test smell* em 97% dos testes existentes, um valor maior em comparação ao estudo de [Bavota et al. 2015], que encontrou uma dispersão de 85%. No estudo de [Palomba et al. 2016], os códigos de testes foram gerados pela Evosuite, alcançando uma dispersão de 83%. No presente estudo a dispersão encontrada foi de 99.95% para a Evosuite.

Como no estudo de [Palomba et al. 2016], foi possível identificar uma correlação entre os *test smells*. Entretanto, observa-se diferentes características dessas correlações entre os pacotes. A partir dos resultados das análises de co-ocorrência e a distribuição dos

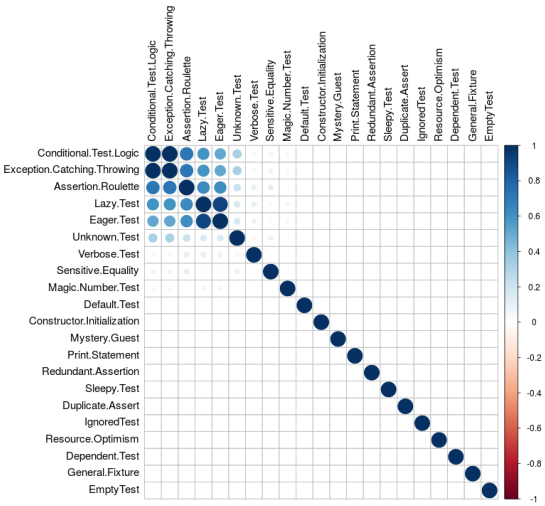
[7] `https://doi.org/10.5281/zenodo.3715424`



(a) Testes pré-existentes



(b) Evosuite



(c) Randoop

Figura 3. Co-ocorrência dos *test smells*

test smells, constata-se que os pacotes de testes diferem de forma significativa. As Figuras 3a, 3b e 3c apresentam o resultado da análise de co-ocorrência [Virgínio et al. 2020a].

Os itens a seguir destacam os principais achados em cada uma das perguntas de pesquisa definidas para essa investigação:

- **Existe diferença na qualidade do código de teste gerado pelas ferramentas Randoop e Evosuite?** Sim. Encontramos aproximadamente seis vezes mais *test smells* do que nos testes da Evosuite e a distribuição de *test smells* ficou em 99,53% para a Randoop e 99,95% para a Evosuite, mostrando diferença de 0,42%. Assim, é possível inferir que os testes da Evosuite tem melhor qualidade do que os testes gerados pela Randoop. Porém, a distribuição de *test smells* foi similar, e a quantidade de tipos de *test smells* foi maior na Evosuite;

- **Existe diferença na qualidade do código de teste gerado pela ferramenta Randoop e os códigos de testes pré-existentes?** Sim. Encontramos trinta vezes mais *test smells* do que nos testes existentes, e a distribuição de *test smells* ficou em 99,53%, valor acima dos testes existentes. É possível inferir que os testes existentes tem melhor qualidade do que os testes gerados pela Randoop;

- **Existe diferença na qualidade do código de teste gerado pela ferramenta Evosuite e os códigos de testes pré-existentes?** Sim. Observamos aproximadamente quatro vezes mais *test smells* do que nos testes existentes e a distribuição de *test smells* ficou em 99,95%, valor acima dos testes existentes. É possível inferir que os testes existentes tem melhor qualidade do que os testes gerados pela Evosuite.

7. Trabalhos Relacionados

Nesta seção, os trabalhos relacionados são apresentados conforme as temáticas abordadas neste estudo.

7.1. Avaliação da qualidade de código de teste gerado automaticamente

O Evosuite e o Randoop são ferramentas bastante utilizadas na indústria para a geração automatizada de testes de software [Almasi et al. 2017, Silva et al. 2017, Fraser and Arcuri 2014]. Em [Almasi et al. 2017], a eficácia das duas ferramentas são analisadas e comparadas. Um dos *insights* levantados é sobre a melhoria da legibilidade dos testes gerados, questão intrinsecamente ligada ao estudo com os *test smells*.

Em um outro estudo, [Silva et al. 2017] investigaram a eficiência de suítes geradas automaticamente por ferramentas de automação de testes, quanto à capacidade de detecção de falhas de refatoração. Assim como o estudo anterior (cuja estratégia também foi seguida em nossa investigação), foram utilizadas as ferramentas Randoop e Evosuite, que resultaram em uma perda na detecção de cerca de 50% de todas as falhas. Isso mostra um *gap* para a melhoria dos códigos gerados.

A ferramenta Evosuite é também mencionada no estudo de [Fraser and Arcuri 2014], que discute um experimento realizado com 100 projetos projetos de software de código-fonte aberto. O estudo confirma o bom alcance de níveis de cobertura da Evosuite. O estudo também exemplifica como a escolha de sistemas de software para um estudo empírico pode influenciar os resultados dos experimentos.

O presente estudo realizou um comparativo entre as ferramentas Evosuite e Randoop. Entretanto, ao invés de utilizar as métricas de cobertura como dados comparativos, foi utilizado os *test smells*, e ainda, utilizou no comparativo os códigos de testes existentes nos projetos. Isso possibilitou um estudo aprofundado sobre a existência dos *test smells* nos códigos de testes.

7.2. Detecção automatizada de *Test Smells*

[Palomba et al. 2016] conduziram uma análise em larga escala da difusão de *test smells* em testes de unidade gerados automaticamente. A análise mostrou alta difusão dos *test smells* e frequente co-ocorrência de diferentes tipos nos projetos. O estudo verificou forte correlação positiva com características estruturais (tamanho e quantidade de classes). Na presente investigação, estendeu-se o estudo de [Palomba et al. 2016] nas técnicas de geração de testes de unidade e sua influência nas ocorrências dos *test smells* encontrados. Para tal, foi acrescentado mais um gerador de teste, a Randoop, e foram utilizados os códigos de testes pré-existentes.

[Bavota et al. 2015] apresentaram duas investigações empíricas sobre a prevalência e o impacto dos *test smells* nos projetos. A primeira investigação mostrou que há grande difusão de *test smells* em sistema de software de código aberto e industriais, com 85% dos testes de unidade contendo pelo menos um *test smell*. A segunda investigação fornece evidências do impacto negativo na compreensão e na manutenção dos códigos de teste do sistema, dificultando em 30% a compreensão daqueles que tinham *test smells*. Como uma contribuição do estudo, foi desenvolvido o *tsDetect*, a ferramenta utilizada para o levantamento de dados do estudo. Essa ferramenta foi utilizada como base para o desenvolvimento da JNose Test, a qual foi desenvolvida a partir da necessidade de automatizar o processo de execução e facilitar o acesso a outros desenvolvedores, além de incluir novas métricas e requisitos os quais não estavam presentes na *tsDetect*, por exemplo, o quantitativo dos *test smells* encontrados e as métricas de cobertura de código.

8. Contribuições

A principal contribuição desse trabalho para a área de Engenharia de Software consistiu em investigar as diferenças entre a qualidade dos códigos de teste gerados de forma automática (ou não) em relação a presença de *test smells*. Em resumo, as contribuições do trabalho são:

- Um estudo quantitativo para investigar a correlação entre a presença de *test smells* e a cobertura do código de teste [Virgínio et al. 2019];
- Um estudo exploratório para identificar se há diferença significativa entre a quantidade de *test smells* inseridos em códigos de teste gerados a partir de ferramentas automatizadas e de códigos pré-existentes [Virgínio et al. 2020a].
- O desenvolvimento da ferramenta `JNose Test`, desenvolvida utilizando a linguagem Java, que tem como funcionalidade a detecção de *test smells* em códigos de testes de unidade desenvolvidos com o *framework* JUnit. Além de trazer métricas de cobertura dos testes, facilita as etapas de seleção e de processamento para o usuário final a partir de uma interface Web amigável [Virgínio et al. 2020b];

9. Considerações Finais

"Quem testa os testes?" Diante de tal questionamento, neste trabalho buscou avaliar a qualidade dos testes de software, sob a perspectiva da existência de *test smells* no código de testes. A investigação consistiu em avaliar empiricamente a qualidade de testes gerados manual e automaticamente. Como dados observados, considerou um conjunto de 21 projetos de software de código-fonte aberto, disponíveis na Plataforma Github. Esses projetos foram executados em duas ferramentas de geração automatizada de testes: Randoop e Evosuite.

A avaliação empírica levou em consideração três hipóteses primárias. Como resultado da avaliação, todas as hipóteses foram refutadas. Uma outra observação bastante significativa do estudo foi a correlação entre os *test smells*, onde cada suíte apresentou pares de correlações de *test smells* distintas. Com isso, concluiu que as características dos *test smells* não são generalizáveis para as metodologias de geração de códigos de teste.

Os resultados aferidos com o estudo destacam a importância do processo de qualidade dos códigos de teste, gerados ou não de forma automatizada, com base nos *test smells* detectados. Observou que os *test smells* são distribuídos em mais que 97% das classes de teste, independentemente do pacote analisado. As ferramentas de geração de teste usam metodologias diferentes e, portanto, apresentaram características diferentes de *test smells*, tais como: quantidade total de *test smells*, quantidade por tipos, quantidade de tipos e correlações entre os *test smells*.

Diante das possibilidades trazidas por este estudo, a seguir apresentamos oportunidades para investigações futuras:

- Conduzir estudos que repliquem o *design* experimental apresentado nesta investigação, no sentido de incluir amostras de tamanho maior (maior quantidade de projetos avaliados e/ou projetos de maior tamanho que aqueles observados no presente estudo); ou ainda, considerar projetos industriais, cujo código-fonte não esteja disponível em repositórios abertos. Isso possibilitará coletar mais evidências, o que acarretará em oportunidades de generalização dos resultados encontrados;
- Realizar a análise de dados de outras ferramentas automatizadas para a geração de testes, de modo a observar se os resultados são semelhantes aos alcançados com as ferramentas Evosuite e Randoop e com os códigos de testes pré-existentes;
- Criação de um *plugin* do `IntelliJ IDEA` de detecção de *test smells* na etapa de codificação dos testes, para a realização de uma análise da qualidade dos testes e apresentação de *bugs* a partir de sua utilização;
- Realizar um estudo sobre o impacto da implantação do uso das refatorações baseadas nos *test smells* em projetos industriais, com acompanhamento da equipe de desenvolvimento (através do método de *action research*), e qual o impacto dessa implementação na qualidade do projeto;
- Realizar uma análise sobre a correlação de métricas de dívidas técnicas e os *test smells* ao longo do desenvolvimento do software;
- Realizar um estudo empírico que verifique a correlação entre a experiência do desenvolvedor com a inserção de *test smells* ao longo do desenvolvimento de um sistema, utilizando o histórico de projetos no GitHub;

- Melhoria da ferramenta JNose Test para funcionar como *plugin* em sistemas de integração contínua, tais como, o Jenkins[8] e Hudson[9]. Isso possibilitará a integração da `JNose Test` com projetos que utilizam essas ferramentas.

Referências

Almasi, M. M., Hemmati, H., Fraser, G., Arcuri, A., and Benefelds, J. (2017). An industrial evaluation of unit test generation: Finding real faults in a financial application. In *International Conference on Software Engineering (ICSE-SEIP)*. IEEE CS.

Bavota, G., Qusef, A., Oliveto, R., Lucia, A. D., and Binkley, D. W. (2015). Are test smells really harmful? an empirical study. *Empir. Softw. Eng.*, 20(4):1052–1094.

CISQ (2021). The Cost of Poor Software Quality in the US: A 2020 Report. `https://www.it-cisq.org/pdf/CPSQ-2020-report.pdf`. Acessed: March 1st, 2021.

Fraser, G. and Arcuri, A. (2014). A large-scale evaluation of automated unit test generation using evosuite. *ACM Trans. on Software Engineering and Methodology*, 24:1–42.

Garousi, V. and Kucuk, B. (2018). Smells in software test code: A survey of knowledge in industry and academia. *Journal of Systems and Software*, 138:52–81.

Gopinath, R., Jensen, C., and Groce, A. (2014). Code coverage for suite evaluation by developers. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE.

Grano, G., Palomba, F., Nucci, D. D., Lucia, A. D., and Gall, H. C. (2019). Scented since the beginning: On the diffuseness of test smells in automatically generated test code. *Journal of Systems and Software*, 156:312 – 327.

Greiler, M., van Deursen, A., and Storey, M. D. (2013). Automated detection of test fixture strategies and smells. In *6th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE Computer Society.

Meszaros, G., Smith, S., and Andrea, J. (2003). The test automation manifesto. In Maurer, F. and Wells, D., editors, *Third XP and Second Agile Universe Conference*, volume 2753 of *LNCS*. Springer.

Pacheco, C. and Ernst, M. (2007). Randoop: Feedback-directed random testing for java. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA*, pages 815–816.

Palomba, F., Nucci, D. D., Panichella, A., Oliveto, R., and Lucia, A. D. (2016). On the diffusion of test smells in automatically generated test code: An empirical study. In *9th International Workshop on Search-Based Software Testing (SBST)*. ACM.

Peruma, A., Almalki, K., Newman, C. D., Mkaouer, M. W., Ouni, A., and Palomba, F. (2019). On the distribution of test smells in open source android applications: An exploratory study. In *29th Annual International Conference on Computer Science and Software Engineering (CASCON)*. ACM.

[8] `https://jenkins.io/`

[9] `https://www.eclipse.org/hudson/`

Silva, I. P. S. C., Alves, E. L. G., and Andrade, W. L. (2017). Analyzing automatic test generation tools for refactoring validation. pages 38–44.

Somé, S. S. and Cheng, X. (2008). An approach for supporting system-level test scenarios generation from textual use cases. In *ACM Symposium on Applied Computing*. ACM.

Spadini, D., Schvarcbacher, M., Oprescu, A.-M., Bruntink, M., and Bacchelli, A. (2020). Investigating severity thresholds for test smells. In *17th International Conference on Mining Software Repositories (MSR)*. ACM.

van Deursen, A., Moonen, L., van den Bergh, A., and Kok, G. (2001). Refactoring test code. In *2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP)*.

Virgínio, T., Martins, L., Rocha, L., Santana, R., Costa, H., and Machado, I. (2020a). An empirical study of automatically-generated tests from the perspective of test smells. In *34th Brazilian Symposium on Software Engineering (SBES)*. ACM.

Virgínio, T., Martins, L., Rocha, L., Santana, R., Cruz, A., Costa, H., and Machado, I. (2020b). JNose: Java Test Smell Detector. In *34th Brazilian Symposium on Software Engineering (SBES)*. ACM.

Virgínio, T., Martins, L., Rocha, L., Santana, R., Cruz, A., Costa, H., and Machado, I. (2021). On the test smells detection: an empirical study on the jnose test accuracy. *Journal of Software Engineering Research and Development (JSERD). Em avaliação*.

Virgínio, T., Railana, S., Martins, L. A., Soares, L. R., Costa, H., and Machado, I. (2019). On the influence of test smells on test coverage. In *33rd Brazilian Symposium on Software Engineering (SBES)*. ACM.

Wood, C. L. and Altavela, M. M. (1978). Large-sample results for Kolmogorov-Smirnov statistics for discrete distributions. *Biometrika*, 65(1):235–239.