UNIVERSIDADE ESTADUAL DE FEIRA DE SANTANA
Autorizada pelo Decreto Federal n° 77.496 de 27/04/76
Recredenciamento pelo Decreto n°17.228 de 25/11/2016
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
COORDENAÇÃO DE INICIAÇÃO CIENTÍFICA

XXIV SEMINÁRIO DE INICIAÇÃO CIENTÍFICA DA UEFS
SEMANA NACIONAL DE CIÊNCIA E TECNOLOGIA - 2020

UM MAPEAMENTO SISTEMATICO SOBRE CATÁLOGOS DE ANOMALIAS DE CÓDIGO UTILIZADOS EM ESTUDOS EMPIRICOS NA ENGENHARIA DE SOFTWARE.

Ivanildo Gomes da Silva

1. Ivanildo Gomes da Silva PIBIC/CNPq, Graduando em Engenharia de Computação, Universidade Estadual de Feira de Santana, e-mail: ivanildo99gomes@gmail.com
2. José Amancio Macedo Santos, DETEC, Universidade Estadual de Feira de Santana, e-mail: zeamancio@gmail.com

PALAVRAS-CHAVE: Cheiro de códigos; Anti-padrões; Anomalia de códigos.

INTRODUÇÃO

Muitas ferramentas, processos, metodologias e afirmam melhorar ou avaliar a qualidade dos softwares. Entretanto há muitos desafios a serem superados no desenvolvimento de software. Fowler (1999), Beck (2001), contribuíram de forma significativa catalogando diversos problemas de código que foi denominado por eles como “maus cheiros” (Bad Smeel), além disso eles catalogaram possíveis soluções para tratar cheiros de código. Podemos entender “maus cheiros” como uma metáfora utilizada para descrever projetos de software ruins, falhas de design de software que podem afetar negativamente a qualidade do software e práticas ruins de programação. Onde alguns cheiros podem representar dois extremos dos mesmos atributos.

Diversos métodos e ferramentas foram desenvolvidas para identificar os anti-padrões e “smells band” em um software afim de medir a sua qualidade. A necessidade da ênfase em ferramentas e técnicas de detecção automáticas, que se baseiam na suposição de que estes arquétipos podem apontar partes particulares do programa que possuem problemas.
Os trabalhos de Riel (1996), Fowler (1999) e Lanza e Marinescu (2005) discutem anomalias de código a partir dos princípios dos paradigmas Orientado a Objetos (OO), como ocultação de informação ou polimorfismo, porém, há um conjunto de questões que dependendo do contexto vem afetar a forma como o conceito de anomalia e considerado na pratica do desenvolvimento de software. As questões incluem: a experiência dos desenvolvedores, a metodologia adotada para desenvolver o software, entre outros. O grande número de questões dependentes do contexto dificulta uma definição consistente, de quais partes do código podem ser consideradas anomalia.

É interessante observar que, alguns dos trabalhos apresentados apresentam resultados divergentes. Por exemplo, Sjøberg et al. (2013) investiga a relação entre anomalias de

código e o esforço na manutenção de software. Eles observaram que nenhuma das anomalias investigadas estava significativamente associada com aumento do esforço na manutenção de software. Macia et al. (2012) investiga a relação entre anomalias e problemas ligados à evolução da arquitetura de sistemas. Eles observaram que muitas anomalias detectadas não estavam relacionadas a problemas arquiteturais. Yamashita (2013) apresenta uma ampla análise baseada no mesmo experimento proposto por Sjøberg et al. (2013). Uma das conclusões dela é que o agrupamento de anomalias de código não é bom indicador do nível de manutenibilidade de sistemas. Há ainda outros estudos que permitem conjecturar sobre a ausência de evidências correlacionando anomalias de código com problemas decorrentes de más decisões de projeto, apesar da teoria indicar o contrário.

A compreensão de quais aspectos afetam a detecção de anomalias de código exige uma avaliação empírica. Segundo Mantyla (2006) “Precisamos de mais estudos empíricos com o objetivo de avaliar criticamente, validar e melhorar nossa compreensão de indicadores de qualidade dos projetos de software”.

Santos et al. (2018) identificaram desafios para a área. Segundo os autores, sem lidar com as questões identificadas não será possível compreender as causas das divergências de experimentos com a teoria relacionada ao conceito de anomalias. Uma das questões está relacionada ao excesso de catálogos com anomalias conceitualmente similares, mas com diferente nomenclatura. Sem agrupar e compreender as similaridades e diferenças entre as anomalias e os catálogos utilizados, não é possível sintetizar os resultados individuais dos experimentos. E sem uma síntese dos estudos realizados, muito esforço em pesquisa pode ser (e provavelmente tem sido) mal direcionado.

METODOLOGIA

Revisão sistemática de métodos mistos foi a metodologia selecionada para realização deste trabalho, descrevendo esforços iniciais de estudar empiricamente o uso do código de cheiros, analisando a uniformidade das avaliações olfácticas, catálogos de cheiros de código.

Segundo Sampaio RF(2002) e Mancini MC(2002) “Uma revisão sistemática, assim como outros tipos de estudo de revisão, é uma forma de pesquisa que utiliza como fonte de dados a literatura sobre determinado tema. Esse tipo de investigação disponibiliza um resumo das evidências relacionadas a uma estratégia de intervenção específica, mediante a aplicação de métodos explícitos e sistematizados de busca, apreciação crítica e síntese da informação selecionada ao possibilitar, de forma clara, um resumo de todos os estudos possíveis de determinada intervenção, a revisão sistemática nos permite incorporam um aspecto com resultados relevantes, ao invés de limitar as nossas conclusões á leitura de somente alguns artigos.”

Sampaio RF(2002) e Mancini MC(2002) concluem que “A revisão sistemática nos permite avaliar de uma forma consistente e generalizada dos resultados entre

populações ou grupos clínicos, bem como especificidade e variações de protocolos de tratamento". Dentro dos métodos de Revisão Sistemática, existe a modalidade de Métodos Mistos de Pesquisa. Que é uma uma revisão que utiliza os achados de estudos qualitativos e quantitativos, integrando métodos qualitativos e quantitativos de análise em uma mesma investigação. Desta formar a metodologia trabalha com uma análise mais abrangente, permitindo avaliar não somente a eficácia das intervenções, mas também viabilidade e a adequação da intervenção a um determinado contexto.

O grande número de questões dependentes do contexto dificulta uma definição consistente de quais partes do código podem ser consideradas como uma anomalia. Fontana et al afirmam que a detecção de anomalias de código "pode fornecer resultados incertos e inconsistentes", isso acontece pelo fato de que a maioria das anomalias são definidas de forma subjetiva e dependente do ser humano.

Diferentes abordagens para detecção de anti-padrões e cheiros de código foram propostas, afim de melhorar a manutenção do software, seu estado atual e possíveis esforços de melhoria devem ser mensuráveis, porém, avaliar a manutenibilidade dos softwares é uma tarefa difícil. Diante da dificuldade de catalogar e identificar os cheiros de código, no presente trabalho apresenta uma análise dos tipos de cheiros de código, suas citações e definições por alguns autores, buscando categoriza-los diante dessas definições, identificando semelhanças e diferenças.

Os dados das pesquisas a serem realizadas durante a execução do projeto foram extraídos de leituras a partir da aplicação do protocolo. Identificando um extenso conjunto de estudos que investigam o impacto de anomalias de código no desenvolvimento de softwares

A primeira fase foi feito um estudo sobre Revisão Sistemática Métodos Mistos (MS) e sua relevância para o desenvolvimento do trabalho. O protocolo MS e um tipo de revisão da literatura que o objetivo e identificar, selecionar e analisar estudos qualitativos quantitativos, onde oferece uma síntese de conhecimento produzido, facilitando a compreensão e a tomada de decisões dos profissionais se baseando nas evidências.

Na segunda fase foi realizada uma pesquisa sobre os mais comuns cheiros de código e extrair informações iniciais de cada tipo, assim foram selecionados 25 tipos. Como já citado cheiro de código pode ser definido como projetos de software ruins, falhas de design de software que podem afetar negativamente a qualidade do software e práticas ruins de programação. Ainda na segunda fase foi realizada uma seleção dos principais trabalhos sobre cheiros de código, foram escolhidos 12 trabalhos, onde foram retirados definições e citações dos autores sobre o tema, entre eles podemos citar os de Brow (1998), Fowler (1999), Demeyer (2002), Mantyla (2006).

A terceira fase foi feira uma correlação entre os tipos de cheiros de código e os trabalhos escolhidos, que pode ser vista na Tabela 1, que foi de extrema importância

para afunilar e direcionar os estudos nos trabalhos e cheiros de códigos mais significativos para esse trabalho, o resultado dessa seleção foi, 22 tipos de códigos de cheiros e 10 trabalhos de autores mais significativos. Que pode ser vista na Tabela 2. A partir desse ponto podemos comparar as definições, catálogos, ferramentas que descrevem esses cheiros de código comparando-os com as definições dos trabalhos já citados anteriormente, em busca de categorizar e identificar semelhanças entre os trabalhos escolhidos e cheiros de código para auxiliarem os desenvolvedores de software em suas pesquisas.

RESULTADOS E DISCUSSÃO

Nessa seção será apresentado os trabalhos que citam e definem os cheiros de códigos selecionados, além de uma breve descrição de alguns cheiros de códigos a partir de definições encontradas. Como citado anteriormente foram analisados 25 tipos de cheiros de códigos diferentes e 12 trabalhos de autores que é apresentado na Tabela 1.

Tabela 1 – Correlação Autores e Cheiros de Código (Smells)

SMELL	AUTOR										
	BROWN (1998)	DEMEYER (2002)	FOWLER (1999)	MANTYLA (2006)	MARINESCU (2002)	MARINESCU (2004)	MARINESCU (2006)	MARINESCU (2010)	MARINESCU (2012)	TRIFU E MARINESCU (2005)	ROBERT MARTIN (2002)
God class	X	X			x	x	x	x	x	x	
Feature envy			X	X	x	x	x	x		x	
Long Method		X	X	X			x				
Long Parameter List			X	X					x		
Data Class			X			x	x	x	x	x	
Message Chains			X							x	
Shotgun surgery			X			x	x				
Speculative Generality			X								
Complex Class							x				
Lazy Class			X								
Class Data Should Be Private (CDSBP)											
Large class			X	X			x			x	
Refused Bequest			X			x					
Spaghetti Code	X										
Refused Parent Bequest							x		x	x	
Blob	X										
Data Clump			X	X					x		
Duplicated code		X	X	X			x		x	x	
Swiss Army Knife	X						x				
Anti Singleton											
God Method						x					
Misplaced Class						x					
Schizophrenic Class									x		
Divergent Charge											
Tradition Breaker							x				

Tabela 1 - Correlação Autores e Cheiros de Código

A partir da Tabela 1 foi feita uma seleção de 22 dos 25 cheiros de códigos apresentados, e a seleção de 10 dos 12 trabalhos que se apresentavam maior relevância, que pode ser

apresentada na Tabela 2, buscando correlacionas as definições e citações, categorizando os cheiros de códigos de forma ordenada e de melhor compreensão.

Tabela 2 – Autores e cheiros de código (Smells)

SMELL	AUTOR									
	BROWN (1998)	DEMEYER (2002)	FOWLER (1999)	MANTYLA (2006)	MARINESCU (2002)	MARINESCU (2004)	MARINESCU (2006)	MARINESCU (2010)	MARINESCU (2012)	TRIFU E MARINESCU (2005)
God class	X	X			x	x	x	x	x	x
Feature envy			X	X	x	x	x	x		x
Long Method		X	X	X			x			
Long Parameter List			X	X					x	
Data Class			X			x	x	x	x	x
Message Chains			X							x
Shotgun surgery			X			x	x			
Speculative Generality			X							
Complex Class							x			
Lazy Class			X							
Large class			X	X			x			x
Refused Bequest			X			x				
Spaghetti Code	X									
Refused Parent Bequest							x		x	x
Blob	X									
Data Clump			X	X					x	
Duplicated code		X	X	X			x		x	x
Swiss Army Knife	X						x			
God Method						x				
Misplaced Class						x				
Schizophrenic Class									x	
Tradition Breaker							x			

Tabela 2 - Autores e Cheiros de Código

A partir da correlação dos cheiros de códigos e trabalhos, foi possível identificar semelhanças em citações e definições de alguns cheiros de código como também catálogos e ferramentas necessárias para identificar e tratar esses cheiros de código, abaixo foram selecionadas algumas citações dos autores em seus trabalhos referentes aos cheiros de código.

CLASSE DEUS:

TRIFU E MARINESCU (2005) “Classe Deus: é uma anomalia estrutural que se aplica a grandes, classes complexas de “monstros”, que tendem a fazer a maior parte do trabalho enquanto delega muito pouco. ”

RIEL (1996) “Identifica duas formas principais do Blob AntiPattern. Ele chama essas duas formas de Deus Classes: Formulário Comportamental e Formulário de Dados. A Forma Comportamental é um objeto que contém um processo centralizado que interage com a maioria das outras partes do sistema. Os dados Form é um objeto que contém

dados compartilhados usados pela maioria dos outros objetos no sistema. Riel apresenta uma série de heurísticas orientadas a objetos para detectar e refatorar a classe Deus designs. "

MARINESCU (2002) "Riel descreve informalmente a Classe de Deus falha de design da seguinte forma [27]: classes de nível superior em um o design deve compartilhar o trabalho de maneira uniforme [...]. Cuidado com classes que acessam diretamente dados de outras classes [...]. Cuidado com as classes com muito conteúdo não comunicativo comportamento."

FEATURE ENVY:

TRIFU E MARINESCU (2005) "É uma anomalia estrutural que se aplica a métodos que parecem mais interessados nos dados de outras classes que a de sua própria classe. "

FOWLER (1999). Diz que um dos principais pontos da FEATURE ENVY "É que eles são uma técnica para empacotar dados com os processos usados nesses dados. Um cheiro clássico é um método que parece mais interessado em uma aula diferente daquela realmente está dentro. O foco mais comum da inveja são os dados. Perdemos a conta das vezes vimos um método que invoca meia dúzia de métodos de obtenção em outro objeto para calcular algum valor. Felizmente a cura é óbvia, o método claramente quer estar em outro lugar, então você use Mover Método para chegar lá. Às vezes, apenas parte do método sofre de inveja; naquilo caso use Extrair Método na parte ciumenta e Mover Método para dar a ela uma casa de sonho. "

MANTYLA (2006) "O cheiro de Feature Envy significa um caso onde um método está fortemente acoplado a outras classes além daquela em que está. Cheiro de intimidade impróprio significa que duas classes estão fortemente acopladas a cada de outros. Message Chains é um cheiro onde a classe A precisa de dados da classe D. Para acessar estes dados, a classe A precisa recuperar o objeto C do objeto B, enquanto A e B têm uma referência. Quando a classe A obtém o objeto C, ela pede a C para obter o objeto D. Quando a classe A finalmente tem uma referência à classe D, A pede a D os dados de que precisa. O problema aqui é que se torna desnecessariamente acoplado às classes B, C e D, quando só precisa de alguns pedaços de dados da classe D. "

MARINESCU (2006) "Os objetos são um mecanismo para manter os dados juntos e as operações que processam esses dados. A desarmonia do design Feature Envy [FBB + 99] refere-se a métodos que parecem mais interessados nos dados de outras classes além da sua própria classe. Esses métodos acessam diretamente ou por meio de métodos de acesso muitos dados de outras classes. Este pode ser um sinal de que o método foi extraviado e que deveria ser mudou-se para outra classe. "

LONG METHOD

FOWLER (1999) "Métodos longos são problemáticos porque muitas vezes contêm muitas informações, que ficam ocultas pela lógica complexa que geralmente fica arrastado para dentro. A refatoração chave é Extrair Método, que pega um monte de código e o transforma em seu próprio método. O método embutido é essencialmente o oposto. Você pega uma chamada de método e a substitui com o corpo do código. Eu preciso do Método Inline quando tiver feito várias extrações e perceber alguns dos métodos resultantes não estão mais exercendo influência "

MARINESCU (2006) "Métodos longos - são indesejáveis porque afetam a compreensibilidade e testabilidade do código. Métodos longos tendem a fazem mais de uma funcionalidade e, portanto, estão usando muitas variáveis e parâmetros temporários, tornando-os mais sujeito a erros. "

LONG PARAMETER LIST

FOWLER (1999) "longas listas de parâmetros são difíceis de entender, porque se tornam inconsistentes e difíceis de usar, e porque você está sempre mudando conforme precisa de mais dados. A maioria das alterações são removidas pela passagem de objetos porque é muito mais provável que você precise para fazer apenas algumas solicitações para obter um novo dado [...]. E se a lista de parâmetros é muito longa ou muda com muita frequência, você precisa repensar sua dependência estrutura. "

MANTYLA (2006) "O cheiro Long Parameter List se refere a casos em que um método tem muitos parâmetros. Portanto, precisamos decidir quantos são demais [...]".

DATA CLASS

DEMEYER (2002) "Essas são classes que têm campos, métodos de obtenção e configuração para os campos e nada mais. Essas classes são portadores de dados burros e quase certamente estão sendo manipulados em demasia detalhes por outras classes. [...]"

MARINESCU (2006) "Classes de dados são portadores de dados "burros" sem complexos funcionalidade, mas outras classes dependem fortemente deles. As faltas de métodos funcionalmente relevantes podem indicar que dados relacionados e comportamento não são mantidos em um só lugar; este é um sinal de um não orientado a objetos Projeto. As classes de dados são a manifestação de uma falta de encapsulamento de dados e de uma proximidade de funcionalidade de dados pobre. "

SHOTGUN SURGERY

FOWLER (1999) "A SHOTGUN SURGERY é semelhante à mudança divergente, mas é o oposto. Você sente isso sempre que você faz um tipo de mudança, tem que fazer várias pequenas mudanças em várias classes diferentes. Quando as mudanças estão em todo lugar, são difíceis de encontrar e é fácil perder uma importante mudança. Neste

caso, você deseja usar Mover Método e Mover Campo para colocar todas as alterações em uma única classe. Se nenhuma classe atual parecer um bom candidato, crie uma. Frequentemente, você pode usar a classe Inline para reunir um monte de comportamento. Você obtém uma pequena dose de mudança divergente, mas você pode lidar facilmente com isso. ”

MARINESCU (2006) “O cheiro da cirurgia de espingarda de Fowler também pode assumir a forma de um pedaço de código que é replicado continuamente em vários métodos, pertencendo a várias classes que poderiam não parecer acopladas a entre si. Por exemplo, quando uma classe é uma Classe de Dados (88), seus clientes muitas vezes duplica a funcionalidade que normalmente estaria sob a responsabilidade dessa classe. Assim, para tais casos, a Duplicação (102) desarmonia também pode ser considerada uma desarmonia de colaboração. ”

LARGE CLASS

FOWLER (1999) “Quando uma classe está tentando fazer muito, geralmente aparece como muitas variáveis de instância. Quando uma classe tem muitas variáveis de instância, o código duplicado não pode estar muito atrás.”

MANTYLA (2006) “[...] Fowler e Beck (2000) dizem que uma classe grande pode muitas vezes ser localizada olhando para o número de variáveis de instância. Portanto, o número de atributos é usado como uma de nossas medidas de tamanho de classe. O cheiro de grande classe também é reconhecido como um anti-padrão conhecido como Blob, Winnebago e God Class. [...]”

REFUSED PARENT BEQUEST

FOWLER (1999) "As subclasses herdam os métodos e dados de seus pais. Mas e se eles não quiserem ou precisa do que eles recebem? Eles recebem todos esses ótimos presentes e escolhem apenas alguns para brincar. O cheiro de legado recusado é muito mais forte se a subclasse está reutilizando comportamento, mas não deseja oferecer suporte à interface da superclasse. Não nos importamos em recusar implementações, mas recusar a interface nos coloca em nossos cavalos altos. Neste caso, no entanto, não mexa com a hierarquia; você deseja destruí-lo aplicando Substituir Herança por Delegação. "

MARINESCU (2006) "Herança é um mecanismo dedicado a suportar mudanças incrementais. Descrição consequentemente, a relação entre uma classe pai e seus filhos é pretendia ser íntimo, mais especial do que a colaboração entre duas classes não relacionadas. Esta colaboração especial é baseada em uma categoria de membros (métodos e dados) especialmente concebida pela classe base a ser usada por seus descendentes, ou seja, os membros protegidos. Mas se uma classe filha se recusar a usar este legado especial preparado por seu pai, então este é um sinal de que algo está errado dentro essa relação de classificação. "

DUPLICATED CODE

DEMEYER (2002) "O código duplicado surge naturalmente conforme o sistema evolui, conforme atalho para implementar código quase idêntico ou mesclar diferentes versões de sistemas de software. Se o código duplicado não for eliminado refatorando as partes comuns em abstrações adequadas, manutenção rapidamente se torna um pesadelo, pois o mesmo código tem que ser corrigido em muitos lugares. "

FOWLER (1999) "O número um na parada fedorenta é o código duplicado. Se você vir a mesma estrutura de código em mais de um lugar, você pode ter certeza de que seu programa será melhor se você encontrar uma maneira de unificá-los. O problema de código duplicado mais simples é quando você tem a mesma expressão em dois métodos de a mesma classe. Então, tudo que você precisa fazer é Extrair Método e invocar o código de ambos lugares."

MANTYLA (2006) “De acordo com Fowler e Beck (2000), o Duplicate Code cheira número um em a parada do fedor. “A remoção da duplicação torna os programas mais fáceis de entender, manter e desenvolver ainda mais. Quando medimos o cheiro do Código Duplicado, deve decidir o tamanho dos fragmentos duplicados que desejamos identificar. Não é muito sábio remover fragmentos de código duplicados que consistem em apenas algumas linhas de código, já que o esforço despendido para os remover superará os benefícios. Não estamos cientes de qualquer recomendação de quantas linhas de código duplicadas são demais. ”

Analisando as definições dos autores referentes aos tipos de cheiros de códigos, podemos perceber que alguns trabalhos apresentam definições semelhantes, entretanto algumas definições de autores podem apresentar contradições da outra, independente da questão temporal da publicação dos trabalhos. Foram apresentadas apenas algumas das definições de cheiros de código que apresentavam contradições, de forma mais estruturada, afim de facilitar a compreensão de cada tipo de cheiro de código. O trabalho atual tem o intuito de analisar os trabalhos dos autores já citados, criando uma categorização das definições dos cheiros de código, visando ajudar aos desenvolvedores na interpretação e entendimento do que são o s cheiros de código e quais são as ferramentas, métodos e catálogos.

CONSIDERAÇÕES FINAIS

A detecção de cheiros de códigos, catálogos e padrões não é uma tarefa fácil, entretanto dependem da construção e classificação atribuído a uma determinada entidade, com base nas definições e citações dos autores apresentados nesse trabalho, é demonstrado que o estudo empírico é de extrema importância na tentativa de catalogar e categorizar os cheiros de código, auxiliando no entendimento, resolução e aplicação de metodologias visando evitar à ocorrência de cheiros de código. Partindo do pressuposto

que as definições devem conter informações coesivas entre ideias, percebe-se que nem todos os trabalhos dos autores apresentados estão em acordo entre si.

O resultado deste trabalho vai ampliar o volume de estudos empíricos sobre o conceito de anomalias de código e contribuirá para uma maior compreensão do efeito da adoção do conceito na prática do desenvolvimento de software. A contribuição resultante deste trabalho suportará a realização de sínteses sobre o tema, auxiliando pesquisadores a direcionar melhor seus estudos, seja na experimentação, seja na construção de ferramentas de suporte. Trabalhos futuros incluem a implementação dessa categorização no desenvolvimento de ferramentas de extração de informação sobre repositórios de software e evolução dos cheiros de código.

REFERÊNCIAS

A. Trifu, R. Marinescu. Diagnosing Design Problems in Object Oriented Systems, Proceedings of the 12th Working Conference on Reverse Engineering (WCRE'05) 2005 IEEE.

BECK K, Beedle M, van Bennekum A, Cockburn A, Cunningham W, Fowler M, Grenning J et al (2001) Manifesto for agile software development. [cited 8/21 2003]. Available from http://agilemanifesto.org/.

CARNEIRO, G. et al.: Identifying code smells with multiple concern views. In: Proc. of 1th Brazilian Conference on Software: Theory and Practice, CBSOFT 2010, Salvador, Bahia, Brazil.

FONTANA, F.A. et al.: An experience report on using code smells detection tools. In: Proc. Of 4th ICSTW 2011, Berlim, Germany.

FORTANA et al.: Anti-pattern and Code Smell False Positives:Preliminary Conceptualisation and Classification, 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering.

FOWLER M, Beck K (2000) Bad smells in code. In: Refactoring: improving the design of existing code, 1st edn. Addison-Wesley, Boston, pp 75–88.

FOWLER, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1999).

M. Lanza., Marinescu, R., Ducasse, S.: Object-Oriented Metrics in Practice. Springer, Secaucus, NJ, USA (2005).

M. Lanza e R. Marinescu, Metrics in Practice. Springer, 2006.

MACIA, I. et al. On the relevance of code anomalies for identifying architecture degradation symptoms. In: *Proc. of the 16th European Conference on SoftwareMaintenance and Reengineering (CSMR)* . [S.l.: s.n.], 2012. p. 277–286.

MANTYLA et al.: Bad smells - humans as code critics. In: 20th IEEE International Conference on Software MaintenanceICSM 2004, ICSM 2004, Chicago Illinois, USA.

MANTYLA, M., Lassenius, C.: Subjective evaluation of software evolvability using code smells: An empirical study. Empirical Software Engineering 11(3), 395{431 (2006).

MANTYLA, M.: An experiment on subjective evolvability evaluation of object-oriented software: explaining factors and interrater agreement. In: Proc. of the 4th International Syimposium on Empirical Software Engineering, ISESE 2005, Noosa Heads, Australia.

MANTYLA, M.V., Lassenius, C.: Drivers for software refactoring decisions. In: Proceedings of the International Symposium on Empirical Software Engineering, ISESE 2006, Rio de Janeiro, Brazil.

MEYER, B.: Object-Oriented Software Construction, 1st edn. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1988).

PARNIN, C. et al.: A catalogue of lightweight visualizations to support code smell inspection. In: Proc. of the 4th Software Visualization, SOFTVIS 2008, Herrsching am Ammersee, Germany. Proc. of the 5th ACM Symposium on Software Visualization, SOFTVIS 2010, Salt Lake City, Utah, USA.

R. C. Martin, Agile Software Development: Principles, Patterns, and Practices, Prentice Hall, 2002.

R. Marinescu, Assessing technical debt by identifying design flaws in software systems, IBM J. RES. & DEV. VOL. 56 NO. 5 PAPER 9 SEPTEMBER/OCTOBER 2012.

R. Marinescu, Detection Strategies:Metrics-Based Rules for Detecting Design Flaws, Department of Computer Science ”Politehnica” University of Timisoara 2004.

R. Marinescu, G. Ganca, e I. Verebi, inCode: Continuous Quality Assessment and Improvement, European Conference on Software Maintenance and Reengineering, 2010.

R. Marinescu, Measurement and Quality in Object-Oriented Design, University of Timisoara, 2002.

RAPU, D. et al.: Using history information to improve design aws detection. In: Proc. Of 8th European CSMR 2004, Tampere, Finland.

RIEL, A. J. *Object-Oriented Design Heuristics*. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1996.

S. Demeyer, S. Ducasse, O. Nierteasz, Object-Oriented Reengineering Patterns. 2002.
Sampaio RF, Mancini MC. Estudos de Revisão Sistemática: Um guia para síntese criteriosa da evidência cientifica,clínica do fisioterapeuta e do terapeuta ocupacional. Rev. Bras. Fisioter. 2007;v. 11, n. 1, p. 83-89,.

SANTOS, José Amancio M. et al.: An exploratory study to investigate the impact of conceptualization in god class detection. In: Proc of 17th International Conference on Evaluation and Assessment in Software Engineering, EASE 2013, Porto de Galinhas, Brazil.

SANTOS, José Amancio M. et al. A systematic review on the code smell effect. **Journal Of Systems And Software** , [s.l.], v. 144, p.450-477, out. 2018. Elsevier BV.

SJØBERG, D. et al. Quantifying the effect of code smells on maintenance effort. *IEEE Transactions on Software Engineering* , v. 39, n. 8, p. 1144–1156, 2013.

SCHUMACHER et al.: Building empirical support for automated code smell detection. In: Proc. of the International Symposium on Empirical Software Engineering and Measurement, ESEM 2010, Bolzano-Bozen, Italy.

SIMON, F. et al.: Metrics based refactoring. In: Proc. of 5th European Conference on Software Maintenance and Reengineering, CSMR 2001, Lisbon, Portugal.

SJOBERG, D.I.K. et al.: Quantifying the effect of code smells onmaintenance e ort. IEEE Transactions on Software Engineering 39(8), 1144{1156 (2013).

W. Brown, R. Malveau, T. Mowbray, e J. Wiley, AntiPatterns: refactoring roftware, arquitecturas e projectos em crise. Wiley, 1998, vol. 3, não. 4.

YAMASHITA, A. How good are code smells for evaluating software maintainability? results from a comparative case study. In: *29th IEEE International Conference on Software Maintenance (ICSM)* . [S.l.: s.n.], 2013. p. 566–571. ISSN 1063-6773.

ZHANG, M.; HALL, T.; BADDOO, N. Code bad smells: A review of current knowledge. *Software Maintenance and Evolution: Research and Practice,* John Wiley & Sons, Inc. New York, NY, USA, v. 23, n. 3, p. 179–202, abr. 2011. ISSN 1532-60X.