Comparando Técnicas de Extração de Valores Limiares para Métricas: Um Estudo Preliminar com Desenvolvedores Web

Raphael Lima[1], **Marcos Dósea**[1,2], **Claudio Sant'Anna**[1]

[1]Departamento de Ciência da Computação –
Universidade Federal da Bahia – Salvador, BA – Brasil

[2]Departamento de Sistemas de Informação –
Universidade Federal de Sergipe – Itabaiana, SE – Brasil

raphael.lima@ufba.br, santanna@dcc.ufba.br, dosea@ufs.br

Abstract. *The use of source code metrics still faces challenges. One of the main challenges is defining thresholds values that make sense. There are a number of techniques for defining these values. However, there is a lack of studies that evaluate the effectiveness of these techniques. This study aims to evaluate the accuracy of five techniques in the definition of threshold values for the detection of design problems. To do this, we conducted an empirical study to evaluate whether developers agree with design problems detected with the use of thresholds, defined by each of the five techniques, for four source code metrics.*

Resumo. *O uso de métricas de código-fonte ainda enfrenta desafios. Um dos principais desafios é a definição de valores limiares que façam sentido. Existe uma série de técnicas para definição de valores limiares. No entanto, existem poucos estudos que avaliem a efetividade destas técnicas. Este estudo tem por objetivo avaliar a acurácia de cinco técnicas usadas para definição dos valores limiares usados para detecção de problemas de design. Para isso, nós conduzimos um estudo empírico para avaliar a percepção dos desenvolvedores sobre problemas de design detectados com os valores limiares obtidos em cinco técnicas, para quatro métricas de código-fonte.*

1. Introdução

A utilização de métricas para a avaliação da qualidade do código-fonte ainda enfrenta alguns desafios [Kamei and Shihab 2016]. Uma das principais dificuldades é a definição dos valores limiares para indicar possíveis problemas de *design* no código-fonte. Por exemplo, Lanza & Marinescu [2006] propõem um valor limiar de 20 linhas de código para classificar um método como longo. Porém, esse valor pode não ser adequado para todos os sistemas. Fatores contextuais como a linguagem de programação e o domínio da aplicação podem influenciar nessa definição [Zhang et al. 2013].

Várias técnicas foram propostas para extrair valores limiares [Alves et al. 2010, Ferreira et al. 2012, Vale and Figueiredo 2015]. Entretanto, os valores limiares obtidos por essas técnicas são genéricos e não consideram informações do contexto da classe ou do sistema que será avaliado. Por exemplo, considerando um sistema desenvolvido em três camadas (*View, Business e Persistence*), será que classes/métodos da camada *Business*, que concentram a complexidade da aplicação, deveriam ser avaliados com os

mesmos valores limiares das classes das camadas *View* e *Persistence*? Trabalhos recentes vêm propondo técnicas para a identificação do contexto de *design* ou arquitetural da classe que permitiriam definir valores limiares que considerem essa informação de contexto [Aniche et al. 2016, Dósea et al. 2018].

Entretanto, apesar da variedade de técnicas, existem poucos estudos que avaliam a percepção dos desenvolvedores sobre os valores limiares. Também não há clareza sobre a percepção dos desenvolvedores sobre a utilização do contexto de *design* da classe na avaliação dos problemas de *design*. Neste trabalho conduzimos um estudo empírico para avaliar a percepção de desenvolvedores Web sobre valores limiares obtidos a partir de cinco técnicas de identificação de valores limiares. Os resultados preliminares indicam que a utilização de valores limiares que consideram alguma informação de contexto podem auxiliar a reduzir o número de falsos positivos para o desenvolvedor.

2. Trabalhos Relacionados

Poucos estudos avaliam a percepção dos desenvolvedores sobre valores limiares. Palomba et al. [2014] avaliam três sistemas Java buscando 12 *bad smells*. O estudo mostra que *bad smells* podem ou não representar um problema de acordo com o contexto do sistema. Oliveira et al. [2015] propõem uma técnica para extração de valores limiares baseada em *benchmark* de sistemas e validam os valores obitdos com desenvolvedores. Os resultados indicaram que aplicações de boa qualidade respeitam os valores limiares. Nosso estudo também avaliou problemas de *design* com desenvolvedores experientes, mas a avaliação não foi limitada a problemas sugeridos por uma única técnica, uma vez que comparamos cinco técnicas distintas de extração de valores limiares.

Vale *et al.* [2018] avaliam a utilização de limiares obtidos a partir de três técnicas de identificação de valores limiares para detectar *God Class* e *Lazy Class* em uma linha de produto de software. Os problemas no sistema foram previamente sugeridos por especialistas. Como resultado, a técnica proposta pelos próprios autores apresenta pequena vantagem em relação às outras técnicas avaliadas. Nosso estudo também avalia as mesmas técnicas avaliadas por Vale *et al.* [2018], exceto a técnica de Lanza & Marinescu [2006], por ela considerar que as métricas têm distribuição normal, o que raramente ocorre. Adicionalmente, o estudo proposto incluiu novas técnicas e utilizou sistemas Web reais com desenvolvedores experientes para avaliar os problemas de *design*.

3. Configuração do Estudo

O objetivo deste estudo foi investigar a percepção de desenvolvedores Web sobre problemas de *design* apontados a partir de valores limiares obtidos com a aplicação de cinco técnicas que extraem valores limiares a partir de *benchmark* de sistemas. Foram formuladas as seguintes questões de pesquisa:

QP1: *Qual técnica propõe valores limiares que melhor refletem a percepção dos desenvolvedores sobre os problemas de design?*

QP2: *Qual a percepção dos desenvolvedores sobre a utilização do papel de design da classe para identificar problemas de design?*

A primeira questão tem como objetivo avaliar de forma quantitativa a percepeção dos desenvolvedores sobre os problemas de *design* identificados, isto é, os métodos cujos valores de métricas ultrapassaram os valores limiares. A segunda questão de pesquisa

pretende avaliar de forma qualitativa a percepção dos desenvovedores sobre o papel de *design* da classe na identificação dos problemas de *design*.

Técnicas para Identificar Valores Limiares: Selecionamos técnicas que não consideram que a distribuição dos valores das métricas seguem distribuição normal, visto que é muito difícil que isso ocorra. Para evitar viés na configuração de parâmetros, também consideramos técnicas que podem ser completamente automatizadas. Baseados nesses critérios, foram selecionadas as técnicas de Alves *et al.* [2010] e Vale & Figueiredo [2015] que geram um valor limiar genérico para cada métrica. Também avaliamos uma variação da técnica de Alves *et al.* [2010], proposta por Dósea *et al.* [2016]. Esta variação consiste em usar sistemas de referência para compor o *benchmark*. Sistemas de referência são desenvolvidos com regras de *design* similares ao sistema que será avaliado e são considerados referência de qualidade de código. Por exemplo, para este estudo, os sistemas de referência foram selecionados por membros experientes da equipe.

A quarta técnica avaliada foi a proposta por Aniche *et al.* [2016] que também se baseia na técnica de Alves *et al.* [2010], mas define valores limiares específicos para as métricas de acordo com o papel arquitetural das classes definidos a partir de *frameworks* pré-definidos (Ex: Android, Spring). Por exemplo, uma classe associada ao papel arquitetural *Controller* deve ser avaliada por valores limiares específicos identificados para esse papel. Finalmente, avaliamos uma nova técnica sugerida por Dósea *et al.* [2018], que também usa Alves *et al.* [2010] e *benchmarks* compostos por sistemas de referência, mas identifica valores limiares específicos de acordo com o papel de *design* da classe. O papel de *design* é uma extensão do conceito de papel arquitetural e permite definir o papel de classes não vinculadas a um *framework* pré-definido [Dósea et al. 2018].

Sistemas Alvo: Foram utilizados dois sistemas Web, desenvolvidos na linguagem Java, que pertencem a Superintendência de Tecnologia da Informação (STI) da Universidade Federal da Bahia (UFBA). Estudos empíricos com foco em avaliar percepção de problemas de *design* por desenvolvedores também não utilizam muitos sistemas [Palomba et al. 2014, Vale et al. 2018]. A seleção foi feita por conveniência, priorizando sistemas cujos desenvolvedores mais experientes estivessem disponíveis para a entrevista. Foram utilizados o Sistema Integrado de Serviços e Usuários (SIUS) e o Sistema de Avalização Docente/Discente (SIAV). O SIUS possui 76 classes, 844 métodos e 8545 linhas de código. O SIAV possui 53 classes, 525 métodos e 5277 linhas de código.

Métricas Analisadas: Foram utilizadas quatro métricas de método, selecionadas pela facilidade de cálculo manual pelos desenvolvedores sem o auxílio de ferramentas, característica importante para facilitar a percepção do problema de *design*. A primeira foi a métrica Complexidade Ciclomática de McCabe (CC) [McCabe 1976] que conta o número de desvios em cada método. A segunda foi a métrica de Número de Parâmetros do Método (NMP) [Fowler and Beck 1999] que conta o número de parâmetros de cada método. A terceira foi a métrica de Linhas de Código (LOC) [Lanza and Marinescu 2006] que conta o número de comandos executáveis em cada método, excluindo comentários e linhas em branco. A última foi a métrica que mede o Acoplamento Eferente (EC) [Martin and Lippman 2000] que conta o número de classes internas e externas ao sistema invocadas em cada método para chamar um método ou acessar um atributo.

Procedimentos do Estudo: A primeira etapa do estudo foi selecionar os sistemas

para compor os *benchmarks*. As técnicas de Alves *et al.* [2010], Vale *et al.* [2015] e Aniche *et al.* [2016] usaram o mesmo *benchmark* composto por 15 sistemas Web reais, desenvolvidos na linguagem Java e selecionados no repositório Github em 11/05/2018. Os sistemas possuem pelo menos 50 estrelas e no mínimo uma atualização desde 01/01/2018. A busca usando esses critérios retornou 272 projetos. Dessa lista, foram excluídas bibliotecas e *frameworks*, sistemas usados como exemplos de implementação, sistemas sem nenhuma *release* e com menos de 100 classes. Esses critérios tiveram como objetivo selecionar sistemas Web reais. Para as outras duas técnicas, baseadas em sistemas de referência [Dósea et al. 2016], foram criados dois *benchmarks*, um para cada sistema avaliado, devido as diferenças nas regras de *design*. Por exemplo, um dos sistemas usava Java Server Faces e o outro Struts para exibição dos dados. Os sistemas de cada *benchmarks* foram sugeridos por um grupo de três desenvolvedores experientes do próprio setor de TI. Os valores limiares foram gerados através da ferramenta DesignRoleMiner [Dósea et al. 2018] que disponibiliza a implementação das cinco técnicas. Adicionalmente, a ferramenta disponibiliza o cálculo de métricas de código, e gera uma lista de métodos com problemas de *design* indicados a partir dos valores limiares obtidos com as cinco técnicas.

Na segunda etapa, o objetivo foi selecionar os métodos, cujo código-fonte seria avaliado pelos desenvolvedores em busca de problemas de *design*. Em um estudo piloto, percebeu-se que avaliar todos os métodos de cada sistema inviabilizaria o estudo, devido ao tempo necessário e a fadiga demonstrada pelos desenvolvedores. Percebeu-se que métodos com todas as métricas abaixo dos valores limiares não eram identificados como problemáticos. Portanto, apenas métodos com uma das métricas acima de algum dos valores limiares definidos por uma das cinco técnicas foram considerados para avaliação. Também foi percebido que métodos associados ao mesmo papel de *design* e com valores de métricas similares obtinham as mesmas avaliações dos desenvolvedores. Por exemplo, se um método do papel de *design Persistence* com 50 linhas de código era avaliado como longo, outro método com 53 linhas de código também era considerado longo. Portanto, para evitar repetição de avaliações e diminuir a quantidade de métodos avaliados, foram selecionados, por papel de *design* e métrica, uma amostra de 10% dos métodos, de forma que não tivéssemos mais de um método com valores iguais ou muito próximos.

Na terceira etapa, foram realizadas entrevistas com um desenvolvedor de cada sistema. Cada desenvolvedor possuía pelo menos cinco anos trabalhando no projeto. Sem saber que os métodos avaliados foram indicados como problemáticos por uma das cinco técnicas estudadas, foi solicitado que o desenvolvedor avaliasse os métodos em busca dos quatro problemas de *design*: método longo, método com muitos parâmetros, método com muitos desvios em seu código e método muito acoplado a outras classes. Dessa forma, para cada método avaliado, o desenvolvedor poderia dizer que não haveria problema ou indicar um ou mais dos quatro problemas de *design*. Para analisar os dados quantitativos obitdos para as cinco técnicas foi realizada o cálculo das medidas de *recall*, precisão e medida-F.

4. Ameaças à Validade

Esta seção apresenta ameaças à validade do estudo e ações tomadas para minimizá-las.

Validade do Construto: Uma possível ameaça é que os *benchmark*s utilizados gerem valores limiares que prejudiquem os resultados de algum método avaliado. Para diminuir esse viés, aplicamos critérios bem definidos para selecionar sistemas Web reais

do repositório Github e montamos uma equipe de especialistas para decidir quais sistemas iriam compor os *benchmark*s usados pelos métodos baseados em sistemas referência de *design* e qualidade. Sobre a medição da percepção dos desenvolvedores, uma possível ameaça era que os desenvolvedores fossem sugestionados para os problemas de *design*. Para evitar esse problema, o protocolo[1] define que o desenvolvedor deve ser informado que os métodos avaliados poderiam ou não ter alguns dos quatro problemas de *design*.

Validade Interna: A principal ameaça é o número de desenvolvedores entrevistados. Mas o estudo é preliminar e também não é comum em outros estudos similares uma grande quantidade de entrevistados. Para minimizar essa ameaça selecionamos o desenvolvedor mais experiente de cada equipe, mas pretendemos em trabalhos futuros entrevistar outros desenvolvedores da mesma equipe e incluir novos projetos.

Validade Externa: Os resultados obtidos são válidos para os dois sistemas avaliados e para as quatro métricas estudadas. Não sugerimos que sejam generalizados.

5. Resultados

A Tabela 1 apresenta os valores limiares obtidos pelas cinco técnicas para os dois sistemas avaliados. Os valores definidos pelas técnicas de Alves *et al*. [2010], Vale & Figueiredo [2015] e Aniche *et al*. [2016] são iguais para os dois sistemas, uma vez que essas técnicas não recomendam a utilização de *benchmarks* específicos para sistemas diferentes. Para as técnicas de Dósea *et al*. [2016] e Dósea *et al*. [2018] foram criados *benchmarks* distintos, detalhados na Seção 3, para avaliação dos dois sistemas. Adicionalmente, as técnicas de Aniche *et al*. [2016] e Dósea *et al*. [2018] definem valores limiares específicos de acordo com o papel arquitetural e papel de *design* da classe respectivamente.

Tabela 1. Valores limiares propostos pelas técnicas

	SIUS					SIAV				
Técnicas	Papel de Design	LOC	CC	Eferente	NOP	Papel de Design	LOC	CC	Eferente	NOP
Alves [2010]	Geral	85	16	10	3	Geral	85	16	10	3
Aniche [2016]	Entity	8	1	2	0	Entity	8	1	2	0
	Controller	28	4	8	0	Controller	28	4	8	0
	Persistence	41	4	7	3	Persistence	41	4	7	3
	Service	39	7	12	4	Service	39	7	12	4
Vale [2015]	Geral	18	3	6	1	Geral	18	3	6	1
Dósea [2016]	Geral	50	11	13	1	Geral	98	16	16	3
Dósea [2018]	Authenticate	43	9	12	2	Authenticator	43	5	11	3
	Adapter	15	2	5	2	Action	92	14	23	3
	Validate	47	18	6	2	Exception	43	5	11	3
	Controller	45	8	14	3	Persistence	43	6	11	3
	Entity	47	13	12	2	Entity	43	5	11	3
	Service	31	8	12	6	Decorator	43	5	11	3
	Model	15	2	5	2	AbstractBO	55	18	11	3
	View	15	2	5	2	ValidatorForm	43	5	11	3

Para calcular a precisão e o *recall* nós usamos os seguintes conceitos: Verdadeiros positivos (TP) são os métodos apontados como problemáticos por uma técnica (valor da métrica maior que o valor limiar) e o desenvolvedor concordou que deveria haver uma refatoração devido ao mesmo problema; Falsos positivos (FP) são os métodos apontados como problemáticos por uma técnica e o desenvolvedor discordou que deveriam ser refatorados; Verdadeiros negativos (TN) são os métodos que não foram apontados como problemáticos por uma técnica e o desenvolvedor concordou em não refatorar; Falsos negativos (FN) são os métodos que não foram apontados como problemáticos por uma

[1]Protocolo e dados dos resultados disponíveis no site: https://sites.google.com/view/vem-2018

técnica e o desenvolvedor concordou que deveriam ser refatorados. Para calcular *Recall* foi usada a fórmula TP / (TP + FN). Para a precisão a fórmula utilizada foi TP / (TP + FP). Para calcular a medida-F foi usada a fórmula (2*precisão**recall*)/(precisão+*recall*).

QP1: *Qual método propõe valores limiares que melhor refletem a percepção dos desenvolvedores sobre os problemas de design?*

As Tabelas 2 e 3 apresentam os resultados das medidas de Precisão, *Recall* e medida-F de cada técnica em cada métrica analisada sobre os dois sistemas avaliados. Para o cálculo destas medidas, foram analisadas as respostas dos desenvolvedores acerca dos métodos apontados com possíveis problemas de *design*. Inicialmente, percebe-se que para métrica de número de linhas de código (LOC), a técnica que apresenta uma maior precisão, em ambos os sistemas, foi a de Alves *et al.* [2010] seguida pelas técnicas de Dósea *et al.* [2016] e Dósea *et al.* [2018], que apresentaram valores maiores de *recall*. A técnica de Vale & Figueiredo [2015] apresentou valores altos de *recall*, mas baixa precisão, por ter definido valores limiares muito baixos que acabam incrementando os falsos positivos. Se analisarmos os valores da medida-F, podemos perceber que as três técnicas Alves *et al.* [2010], Dósea *et al.* [2016] e Dósea *et al.* [2018] obtiveram os melhores resultados para a métrica LOC.

Tabela 2. Resultados das Análises do Sistema SIUS

Técnicas	LOC			CC			Eferente			NOP		
	Precisão	Recall	F	Precisão	Recall	F	Precisão	Recall	F	Precisão	Recall	F
Alves [2010]	1,00	0,33	0,50	0,00	0,00	—	0,03	1,00	0,05	0,07	1,00	0,13
Aniche [2016]	0,17	0,33	0,22	0,40	0,33	0,13	0,02	1,00	0,04	0,02	1,00	0,04
Vale [2015]	0,08	1,00	0,15	0,30	1,00	0,46	0,01	1,00	0,03	0,06	1,00	0,04
Dósea [2016]	1,00	0,33	0,50	0,00	0,00	—	0,17	1,00	0,29	0,07	1,00	0,13
Dósea [2018]	0,50	0,33	0,40	0,75	0,50	0,60	1,00	1,00	1,00	0,07	1,00	0,13

Tabela 3. Resultados das Análises do Sistema SIAV

Técnicas	LOC			CC			Eferente			NOP		
	Precisão	Recall	F	Precisão	Recall	F	Precisão	Recall	F	Precisão	Recall	F
Alves [2010]	1,00	0,50	0,67	1,00	0,33	0,21	0,67	0,50	0,57	—	—	—
Aniche [2016]	0,25	1,00	0,40	0,13	0,67	0,67	0,44	1,00	0,62	0,00	—	—
Vale [2015]	0,10	1,00	0,17	0,14	1,00	0,25	0,36	1,00	0,53	0,00	—	—
Dósea [2016]	0,50	1,00	0,67	0,50	0,33	0,40	0,67	0,50	0,57	0,00	—	—
Dósea [2018]	0,50	1,00	0,67	0,67	0,67	0,67	0,67	0,50	0,57	—	—	—

Para a métrica de complexidade ciclomática (CC), percebe-se que a técnica com os melhores resultados para precisão, *recall* e medida-F nos dois sistemas foi a de Dósea *et al.* [2018]. As outras técnicas apresentaram variações entre os sistemas. As técnicas de Alves *et al.* [2010] e Dósea *et al.* [2016] não obtiveram resultados no sistema SIUS e tiveram resultados regulares no SIAV. As técnicas de Aniche *et al.* [2016] e Vale & Figueiredo [2015] apresentaram baixa precisão, principalmente no SIAV.

Para a métrica de Acoplamento Eferente, verificamos que para o sistema SIUS, todas as técnicas apresentaram 100% de *recall*, mas em termos de precisão a técnica de Dósea *et al.* [2018] foi a que apresentou os melhores resultados. Para o SIAV, as técnicas de Alves *et al.* [2010], Dósea *et al.* [2016] e Dósea *et al.* [2018] novamente obtiveram bons resultados. Aniche *et al.* [2016] e Vale & Figueiredo [2015] continuaram apresentando valores altos de *recall*, mas baixa precisão. Analisando as medidas-F, pode-se perceber que a técnica de Dósea *et al.* [2018] foi a que obteve os melhores resultados.

Para a métrica de número de parâmetros (NOP), todas as técnicas apresentaram resultados ruins. Para o sistema SIUS, todas apresentaram um *recall* de 100%, mas taxas

muito baixas de precisão. Os piores valores foram os da técnica de Aniche *et al*. [2016]. Para o SIAV, não foi possível gerar os resultados devido à falta de verdadeiros positivos (TP) e falsos negativos (FN), isto é, não existiram métodos com número de parâmetros acima dos limiares para que os desenvolvedores concordassem ou discordassem. Como as técnicas praticamente não apontaram métodos para serem avaliados no SIAV, a análise foi prejudicada para esta métrica.

Após as análises individuais dos resultados em cada métrica, podemos definir que o método que propôs os valores limiares que mais se aproximaram da percepção dos desenvolvedores sobre os problemas de *design* foi a técnica de Dósea *et al*. [2018]. As técnicas de Dósea *et al*. [2016] e Alves *et al.* [2010] também obtiveram alguns bons resultados, principalmente para a métrica LOC. Os piores resultados ficaram com as técnicas de Aniche *et al*. [2016] e Vale & Figueiredo [2015] que apresentaram valores baixos de precisão para praticamente todas as métricas analisadas, apesar dos valores altos de *recall*.

QP2: *Qual a percepção dos desenvolvedores sobre a utilização do papel de design da classe para identificar problemas de design?*

Para fazer uma análise qualitativa dos resultados, foram realizadas algumas perguntas (disponíveis no protocolo) aos desenvolvedores com o objetivo de capturar a percepção dos desenvolvedores sobre a influência do papel de *design* das classes na avaliação dos problemas de *design*. No sistema SIAV, percebemos na Tabela 1, que os valores limiares obtidos com a técnica de Dósea *et al*. [2018] para o papel de *design Action* nas métricas LOC, CC e Eferente foram mais do que o dobro dos limiares encontrados para os outros papéis de *design*. Para a métrica de acoplamento, o desenvolvedor do SIAV explica que *"É comum que sejam encontrados métodos acoplados em camadas Business e nas Actions. Na Action, geralmente são feitas muitas chamadas a business ou a services, no qual são chamadas algumas regras do sistema"*. Em relação a métrica de linhas de código, ele explica que *"É comum que as Actions sejam longas, pois a Action é responsável por receber dados da interface, fazer consultas em outras camadas (como service ou business) para então tratar o que for necessário e depois retornar um direcionamento (foward) ao sistema."* Os relatos exemplificam situações nas quais o desenvolvedor parece considerar o papel de *design* da classe para fazer a avaliação da qualidade do método. Ou seja, o papel de *design* da classe é utilizado pelos desenvolvedores como informação de contexto para definir o valor limiar a ser considerado na avaliação.

Outra descoberta foi a possibilidade de problemas de *design* identificados em métodos associados ao mesmo papel de *design* poderem ser estendidos para outros métodos associados ao mesmo papel de *design* e com métricas similares. O desenvolvedor do SIUS disse que *"Se um método de MBEAN é longo ou complexo, e ele faz o que precisa ser feito para realizar suas tarefas, então outros métodos de MBEAN também terão esses problemas."*.

6. Conclusão e Trabalhos Futuros

Neste trabalho foi conduzido um estudo empírico para avaliar a percepção dos desenvolvedores Web sobre valores limiares obtidos a partir de cinco técnicas de identificação de valores limiares baseados em *benchmark* de sistemas. Os resultados preliminares indicam que a utilização de valores limiares que consideram alguma informação de contexto podem auxiliar a reduzir o número de falsos positivos para o desenvolvedor. Como trabalhos

futuros, pretende-se estender o estudo para outros sistemas maiores, com a participação de outros desenvolvedores e empresas diferentes.

Agradecimentos. Esse trabalho é apoiado pelo CNPq (projeto 312153/2016-3).

Referências

Alves, T. L., Ypma, C., and Visser, J. (2010). Deriving metric thresholds from benchmark data. In *Int'l. Conf. on Sofware Maintenance. (ICSM)*, pages 1–10.

Aniche, M., Treude, C., Zaidman, A., Deursen, A. V., and Gerosa, M. A. (2016). Satt: Tailoring code metric thresholds for different software architectures. In *Int. Working Conf. on Source Code Analysis and Manipulation (SCAM)*, pages 41–50.

Dósea, M., Sant'Anna, C., and da Silva, B. C. (2018). How do design decisions affect the distribution of software metrics? In *Proceedings of the 26th Conference on Program Comprehension (ICPC)*, pages 74–85.

Dósea, M., Sant'Anna, C., and Santos, C. (2016). Towards an Approach to Prevent Long Methods Based on Architecture-Sensitive Recommendations. In *Work. on Softare Visualization, Evolution and Maintenance (VEM)*, Maringá.

Ferreira, K. A., Bigonha, M. A., Bigonha, R. S., Mendes, L. F., and Almeida, H. C. (2012). Identifying thresholds for object-oriented software metrics. *Journal of Systems and Software.*, 85(2):244–257.

Fowler, M. and Beck, K. (1999). *Refactoring: improving the design of existing code*. Addison-Wesley Professional, Boston, MA, USA.

Kamei, Y. and Shihab, E. (2016). Defect prediction: Accomplishments and future challenges. In *Int'l. Conf. on Soft. Analysis, Evolution and Reengineering*, pages 33–45.

Lanza, M. and Marinescu, R. (2006). *Object-Oriented Metrics in Practice*. Springer, Berlin, Heidelberg.

Martin, R. and Lippman, S. (2000). *More C++ Gems*. SIGS Reference Library. Cambridge University Press.

McCabe, T. J. (1976). A complexity measure. In *Proc. of the 2nd Int't Conf. on Software Engineering*, Los Alamitos, CA, USA. IEEE Computer Society Press.

Oliveira, P., Valente, M. T., Bergel, A., and Serebrenik, A. (2015). Validating metric thresholds with developers: An early result. In *Int'l. Conf. on Software Maintenance and Evolution (ICSME), Bremen, Germany*, pages 546–550.

Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., and De Lucia, A. (2014). Do they really smell bad? a study on developers' perception of bad code smells. In *Int'l Conf. on Software Maintenance and Evolution (ICSME)*, pages 101–110. IEEE.

Vale, G., Fernandes, E., and Figueiredo, E. (2018). On the proposal and evaluation of a benchmark-based threshold derivation method. *Software Quality Journal*, pages 1–32.

Vale, G. A. D. and Figueiredo, E. M. L. (2015). A method to derive metric thresholds for software product lines. In *2015 29th Braz. Symp. on Soft. Eng.*, pages 110–119.

Zhang, F., Mockus, A., Zou, Y., Khomh, F., and Hassan, A. E. (2013). How does context affect the distribution of software maintainability metrics? In *IEEE Int'l. Conf. on Software Maintenance (ICSM)*, pages 350–359.