Evaluating Domain-Specific Metric Thresholds: An Empirical Study

Allan Mori[1], Gustavo Vale[2], Markos Viggiato[1], Johnatan Oliveira[3], Eduardo Figueiredo[1,5], Elder Cirilo[4], Pooyan Jamshidi[5] and Christian Kastner[5]

[1]Computer Science Department, Federal University of Minas Gerais (UFMG), Belo Horizonte, Brazil
[2]Department of Informatics and Mathematics, University of Passau, Passau, Germany
[3]Computer Science Department, Pontifical Catholic University of Minas Gerais (PUC-Minas), Belo Horizonte, Brazil
[4]Computer Science Department, Federal University of Sao Joao del-Rei (UFSJ), Sao Joao del-Rei, Brazil
[5]Institute for Software Research, Carnegie Mellon University (CMU), Pittsburgh, United States

{allanmori, markosviggiato, johnatan.si, figueiredo}@dcc.ufmg.br, vale@fim.uni-passau.de, elder@ufsj.edu.br, pjamshid@andrew.cmu.edu, kaestner@cs.cmu.edu

ABSTRACT

Software metrics and thresholds provide means to quantify several quality attributes of software systems. Indeed, they have been used in a wide variety of methods and tools for detecting different sorts of technical debts, such as code smells. Unfortunately, these methods and tools do not take into account characteristics of software domains, as the intrinsic complexity of geo-localization and scientific software systems or the simple protocols employed by messaging applications. Instead, they rely on generic thresholds that are derived from heterogeneous systems. Although derivation of reliable thresholds has long been a concern, we still lack empirical evidence about threshold variation across distinct software domains. To tackle this limitation, this paper investigates whether and how thresholds vary across domains by presenting a large-scale study on 3,107 software systems from 15 domains. We analyzed the derivation and distribution of thresholds based on 8 well-known source code metrics. As a result, we observed that software domain and size are relevant factors to be considered when building benchmarks for threshold derivation. Moreover, we also observed that domain-specific metric thresholds are more appropriated than generic ones for code smell detection.

CCS CONCEPTS

• **Software and its engineering → Software creation and management → Software post-development issues**

KEYWORDS

Software metrics, thresholds, software domains

TechDebt '18, May 27–28, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5713-5/18/05…$15.00
https://doi.org/10.1145/3194164.3194173

1 INTRODUCTION

Software metrics have been pragmatic means to detect technical debt and improve software products since the dawn of software engineering [1, 7]. Several studies have proposed and evaluated the usefulness of software metrics for practitioners [3, 15, 16, 18], for instance, to indicate technical debt in software systems [16]. As a result, developers may pay the debt by checking if there is something wrong in the system design or code and fixing anomalies to avoid future maintenance issues.

Nevertheless, the effective measurement of software systems is directly dependent on the definition of appropriate thresholds [1, 33]. Software metric thresholds allow us to objectively characterize or classify each entity (i.e., class or method) according to one of the software metrics. Different strategies and methods to derive thresholds have been proposed and, over the years, they have become more straightforward and systematic [33]. The current practice for deriving thresholds is to use a set of similar systems, named benchmarks, and follow a well-defined method [1]. The idea behind the use of benchmarks is to use information from similar systems (e.g., same programming language) to help deriving meaningful metric thresholds.

Unfortunately, existing methods and tools to derive thresholds either do not consider the intrinsic characteristics of software systems in each domain [1, 33] or provide a superficial analysis on thresholds for software domains [10]. That is, they ignore the fact that systems from different domains may have different degrees of complexity and size, for instance. As a result, even when a robust and pragmatic method is used, the derived thresholds can be inappropriate or meaningless.

This paper presents a large-scale empirical study to investigate whether and how thresholds vary among systems of different domains. We argue that the definition of appropriate metric thresholds needs to be tailored by each software domain. For instance, business systems require different thresholds of health systems, or the detection of technical debts might be imprecise.

Therefore, we defend the idea of domain-specific thresholds, given that systems from different domains may have distinct characteristics (e.g., localization systems can be more coupled than education systems) and, it may impair the derived metric thresholds.

This study relies on fifteen software domains composed of object-oriented Java systems. We apply to each system a set of eight well-known source code metrics [3, 16, 18]. After applying the measurements, we derived 90% and 95% thresholds for each metric per domain, compared, and analyzed them in different ways. For instance, we compared the thresholds among domains and analyzed the effectiveness of code smell detection between domain-specific and generic thresholds. Generic thresholds are derived from heterogeneous systems from several domains.

Our results confirm our intuition that metric thresholds vary across domains and most domain-specific thresholds differ from generic thresholds. However, we also found some similarities among thresholds from different domains. For instance, the thresholds from Business and Accounting domains are very similar, especially for size metrics. Furthermore, we saw that the size of the systems that compose each benchmark also affects the thresholds. In particular, we verified that all metrics have higher thresholds when larger systems are used to compose the benchmark. Finally, our analysis indicates that domain-specific thresholds are better than generic thresholds to detect code anomalies.

Overall, our contributions are:

- A large-size empirical study with more than 3 K systems to explore threshold derivation with the corresponding measurements for eight well-known software metrics [30].
- Empirical evidence that the domain and size are factors to consider when building benchmarks for threshold derivation.
- An observation that some domains have characteristics in common and, in essence, these domains can be grouped to promote more reliable benchmarks, or thresholds can be reused across similar domains.
- A reference list of code smells for 43 systems in 15 different domains [30].
- An empirical comprehension that domain-specific thresholds are better than generic thresholds for code smell detection.

2 BACKGROUND AND RELATED WORK

This section presents important concepts to understand this study, as well as, related work that explores software metrics, metric thresholds, and software domains.

Software metrics. One of the oldest cliché phrases in business is "You cannot manage what you cannot measure" [7]. It might not be always true, but measurements surely support project managers improving their products or processes [7]. In software engineering, researchers and practitioners are always looking for better ways to predict the number of faults, errors, and the effort to complete a task [15]. In this sense, many metrics have been proposed to measure technical debts in software products aiming at better controlling, assessing, and improving their quality. Examples of well-known metrics [15] to measure software products are Lines of Code [18], McCabe Cyclomatic Complexity [21], Coupling between Objects [3], and Depth of Inheritance Tree [3].

Metric thresholds. Despite the importance of software metrics, just their use is not enough because we need to know when a target metric starts to become an outlier or a problem in the software design or code. To overcome this problem, software engineers define or derive thresholds for their software metrics. Thresholds allow us to characterize objectively or to classify each component according to one of the quality metrics [33].

There are several methods and strategies to derive metric thresholds [1, 3, 5, 6, 13, 21, 23, 26, 28, 34]. In the past, thresholds were calculated based on experience of software engineers or using a single system as a reference [3, 21]. Nowadays, thresholds have been derived from benchmarks and calculated based on well-defined methods. For instance, Alves et al. [1] proposed a method that weight software metrics by lines of code. The method aims at labeling each entity of a system based on thresholds. Each label is based on a fix and predetermined percentage of entities. Similarly, Ferreira et al. [10] presented a simple method for calculating thresholds. The method consists in grouping the extracted metrics in a file and gets three groups, with high, medium, and low frequency. Oliveira et al. [23] propose a method based on the concept of relative thresholds. Their method consists in the formula called Compliance Rate and this formula considers the median of each system in the benchmark to derive thresholds. Vale et al. [34] derive thresholds based on lessons learned from a comparison of Alves', Ferreira's and Oliveira's methods and provide upper and lower-bound thresholds in four different labels. Unlike Alves' method, Vale's method does not correlate metrics for deriving thresholds [33].

Software domains. Some studies [17, 22, 27] have grouped software systems based on domains because somehow, they believe these systems share similar characteristics and differ from systems of other domains. For instance, Ray et al. [27] grouped systems in 7 different domains to check if language defect proneness depends on the software domain. Linares-Vásquez et al. [17] grouped systems in 13 domains to investigate the relationships between the presence of smells and quality-related metrics. Murphy-Hill et al. [22] compare the development of game (domain) with the development of other software systems (from other domains). These three studies have found that software domains matter in their analysis. For instance, Linares-Vásquez et al. observed that anti-patterns have different frequency in the different application domains and Murphy-Hill et al. [22] observed that the development of games differs from other software systems in several ways, such as the type of requirements, software design, and software quality. Like these studies, we share the idea that systems from the same software domain have similar characteristics. Unlike them, we use systems from similar domains to analyze metric thresholds.

Benchmark and threshold analysis. There are also studies analyzing benchmarks and thresholds, as we do in this work. The benchmark of previous studies [1, 10, 23, 33, 34] varies from 14 to 106 systems. For instance, Vale et al. [33] investigate three small benchmarks with 14, 22, and 33 configurable software systems. Instead, we used a larger benchmark in this work which is composed by more than 3K systems. Alves et al. [1] rely on a benchmark

composed by two programming languages (Java and C#) while Ferreira et al. [10] performed an analysis regarding domains, types, and size in their benchmark. Unlike these studies, we analyzed a larger number of systems (3K against 40), metrics (8 against 5) and domains (15 against 11).

3 STUDY SETTINGS

This section describes our research questions and experimental steps. We explain how we built our dataset and how we perform the measurement and threshold derivation.

3.1 Research Questions and Experimental Steps

This study aims to investigate (i) whether metric thresholds vary across systems of different software domains, (ii) if metric thresholds vary across systems of different sizes, and (ii) if domain-specific thresholds are better than generic thresholds for detecting code smells. We assume that some characteristics of each domain affect the systems and measures, so it may impair the derived thresholds. Given this assumption, we defined four research questions (RQs) as follows.

RQ1. Do thresholds for the same metric vary among different software domains?

RQ2. Are there metrics with the same thresholds regardless of the software domain?

RQ3. Does the system size impact on the derived thresholds?

RQ4. Are domain-specific thresholds better than general thresholds for detecting the God Class code smell?

We expect our findings to provide evidence that software domains and the system size should be considered when building benchmarks, for instance, to identify technical debts. For example, inaccurate thresholds may influence negatively the derived metric thresholds by providing meaningless values about singular software domain. Regarding the empirical steps, Fig. 1 presents an overview of this study described below.

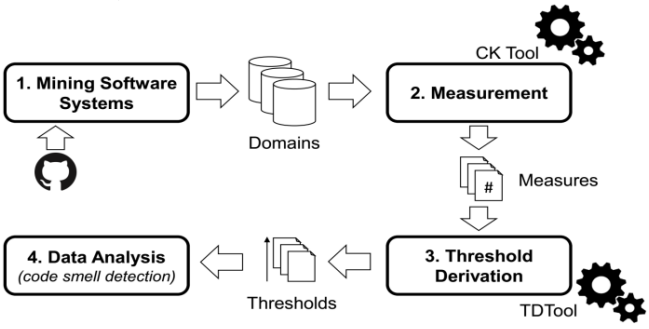


Figure 1: Experimental steps for the analysis of thresholds.

First, we built our dataset by mining open-source systems from GitHub (Step 1). We then measured the source code of each system using CK Tool [4] (Step 2). Once measured all systems, we derived metrics thresholds using TDTool [34] (Step 3). Finally, we analyzed the results and evaluated the effectiveness of domain-specific thresholds to detect code smells compared to generic thresholds (Step 4). The first three steps are described in the following sub-sections and the data analysis is described in Sections 4 to 6.

3.2 Selected Systems and Domain Classification

Table 1 presents and describes the 15 domains explored in this study with the number of systems per domain (last column). We choose these domains for the following reasons. First, they are well-defined in terms of requirements and, most of these domains have been used in previous studies [10, 17, 22], therewith, we believe that they are representative. Second, they encompass several types of systems (e.g., frameworks and tools. Third, there is a significant number of systems in these domains publicly available in GitHub.

Table 1: Description of Domains in Our Dataset

Domain	Description	#
Accounting (Acc)	Systems that record and process accounting transactions, such as accounts payable and receivable, payroll, and trial balance.	216
Business (Bus)	Systems that implement validation, calculation, and law regulations of business requirements, such as pricing, and inventory management.	368
Communication (Com)	Systems that manage connections between server and clients, using protocols to share information.	266
Development (Dev)	Software tools that support developers to implement projects in general.	528
Dictionaries (Dic)	Software tools used to translate a variety of languages.	130
E-commerce (EC)	Systems in charge of supporting the transactions of products buying and selling, as well as providing services to consumers.	34
Education (Edu)	Systems used by students to manage their study life and by managers to administrate their schools.	165
Free Time (FT)	Entertain systems or applications which provide information for joy, such as travel information.	28
Games (Gam)	Entertainment games that can be played alone or in collaboration.	452
Health (Hea)	Systems that offer health-related services to people in general.	279
Home (Hom)	Systems that control basic many home devices and services.	160
Localization (Loc)	Show local information normally based on GPS. Some of them display maps and location sensitive data for users.	70
Messaging (Mes)	Systems that allow the users to send messages from one client to another.	66
Restaurant (Res)	Systems that provide different services for both managers and users of food houses.	326
Science & Engineering (ScE)	Systems designed to aid users in several fields of science and engineering, such as 3D visualization and data analysis.	19

Mining and Classification. We mined systems from GitHub to compose our dataset in October 2017. For each domain, we collected up to 1,000 by automatically searching the project name and description for keywords that match the domain name. For instance, we collected 34 e-commerce systems by searching for "e-commerce" or "ecommerce". Whenever more systems have been returned by the GitHub search, we selected the first 1,000 systems in the descending order of stars. In GitHub, stars are a meaningful measure for repository popularity. We developed and iteratively tested a script that automatically searches relevant systems and applies a set of criteria. For instance, we excluded systems with less than 1,000 lines of code because we considered them toy examples or incipient software projects. In addition, we focus in Web and

Desktop-based Java systems and, therefore, we removed other projects (e.g., mobile apps) because they tend to have a different architectural design. Finally, we manually validated the collected systems by checking the project name and readme file and excluded the wrongly classified systems. As a result, the dataset used in this study includes 3,107 software systems in the 15 domains, with at least 19 systems per domain.

Table 2 summarizes the descriptive statistics of our dataset focusing on the source lines of code per domain (excluding code of test cases). The software systems diverge largely in lines of code, while the smallest software system in every domain has about 1,000 LOC (due to the criteria described above), the largest system by domain vary from about 26,633 LOC in the E-Commerce domain to over 7 MLOC in the Business domain. We can also observe a considerable variance (i.e., standard deviation) and that data do not follow a normal distribution. In fact, all domains follow a right skewed distribution for lines of code. This considerable difference among domains corroborates with our assumption that systems in one domain might be more complex than systems in other domains, for instance. Therefore, metric thresholds should be tailored to each specific domain.

Table 2: Lines of Code per System Domain

Domain	Min	Max	Mean	Median	SD
Acc	1,003	1,938,805	83,358.40	4,984.00	308.704,89
Bus	1,011	7,822,498	130,576.84	6,402.50	775.923,96
Com	1,006	1,351,323	37,604.28	5,042.00	125.570,70
Dev	1,153	1,498,869	69,427.47	16,931.50	161.547,59
Dic	1,016	472,642	14,307.52	3,556.00	47.758,51
EC	1,104	26,633	6,298.76	3,133.50	7.340,34
Edu	1,025	1,170,720	26,440.87	5,204.00	102.033,91
FT	1,047	284,220	45,509.96	4,511.00	84.510,30
Gam	1,009	546,748	17,989.02	4,877.50	49.593,35
Hea	1,025	1,446,807	27,161.00	4,790.00	109.778,74
Hom	1,022	299,898	29,538.95	9,119.50	53.988,07
Loc	1,054	859,393	40,062.38	12,015.50	110.306,52
Mes	1,000	160,297	14,648.90	3,891.00	29.911,82
Res	1,063	151,814	7,639.50	2,951.50	18.216,42
ScE	1,028	439,747	36,939.84	8,649.00	100.667,62

Table 3: Number of Classes per System Domain

Domain	Min	Max	Mean	Median	SD
Acc	3	10,125	349.07	50.00	1061.97
Bus	1	12,470	471.92	65.50	1624.18
Com	3	8,379	249.99	49.00	753.12
Dev	3	8,735	472.20	167.50	962.62
Dic	4	2,296	87.76	30.00	234.74
EC	7	301	80.52	46.00	83.86
Edu	2	5,641	225.33	66.00	586.22
FT	10	2,128	296.67	42.00	564.69
Gam	1	2,439	116.40	45.00	221.61
Hea	3	5,322	186.50	51.00	592.33
Hom	4	2,798	238.14	83.50	426.75
Loc	4	4,041	222.97	57.50	540.42
Mes	1	980	100.00	44.50	192.06
Rest	4	1,263	62.31	35.00	114.14
ScE	12	1,528	184.42	78.00	358.14

Similar to Table 2, Table 3 shows descriptive statistics for the number of classes and interfaces in systems of each domain. In general, we can observe a considerable difference among systems inside a domain. For instance, while the median is 167.50 classes in the Development domain, the largest system in this domain has more than 8K classes. Considering the median per domain, Development has the largest systems in terms of classes and Dictionary has the smallest ones. On the other hand, the largest system, in terms of number of classes, belongs to the Business domain, containing more than 12K classes. Based on these observations, it is also expected that the variation among domains reflects in software metrics and, in their thresholds.

3.3 Measurement and Threshold Derivation

This study investigates domain-specific thresholds for eight metrics. We used a measurement tool, CK Tool [4], to compute the source code metrics. CK tool measures Java programs by means of static analysis and it supports all metrics used in this study. Table 4 presents the target metrics, showing their names and a brief description. We choose these metrics because: (i) they capture different attributes of software systems, such as size, complexity; (ii) they are well-known object-oriented software metrics [15]; and (iii) they have been often used by researchers and practitioners to measure several technical debts and quality attributes [3, 15].

Table 4: Software Metrics for Technical Debt Detection

	Metric	Description
Size	Lines of Code (LOC) [18]	Measures the number of lines of code per class. It counts neither comment lines nor blank lines.
	Number of Attributes (NOA) [18]	Quantifies the number of fields and constants in a class.
	Number of Methods (NOM) [18]	Quantifies the number of methods and constructors in a class.
Complexity	Weighted Method per Class (WMC) [3]	Counts the number of methods in a class weighting each method by its cyclomatic complexity [21].
	Lack of Cohesion in Methods (LCOM) [3]	Divides (i) the pairs of methods in a class that do not access any attribute by (ii) the pairs of methods in a class that do access attributes in common.
	Coupling Between Objects (CBO) [3]	Counts the number of classes that are coupled to a class, by calling methods or accessing attributes of the other classes.
Inheritance	Depth of Inheritance Tree (DIT) [3]	Counts the number of levels that a subclass inherits methods and attributes from a superclass in the inheritance tree.
	Number of Children (NOC) [3]	Counts the number of direct subclasses of a given class. This metric indicates software reuse by means of inheritance.

We present now the method and tool used in this paper to derive metric thresholds. The selected method, supported by TDTool [34], proposes the threshold derivation in five steps. First, metrics are extracted from each benchmark of software systems presented in Table 2. After the measurement, we computed the weight percentage for each entity within the total number of entities (2nd step). We then sorted the metric values in ascending order and used the maximum metric values to represent 1%, 2%, up to 100%, of the weight (3rd step). In the 4th step, we aggregated all entities per metric value. Finally, we selected thresholds in the 5th step to represent the labels high (90%) and very high (95%). In this paper, we focus on these two top-threshold values (i.e., 90% and 95%) because the

differences are higher for these labels making the analysis more applicable. We choose this method because it is supported by a software tool, making the threshold derivation process easier. In addition, the method was proposed based on lessons learned from a comparison of other methods [33].

4 RESULTS

This section presents the derived thresholds for each domain on the dataset presented in Section 3.2. Table 5 shows the threshold values for 90% and 95% for each software metric and domain analyzed. We classify domains with similar characteristics into four categories to make our discussions more direct and to support us answering RQ1 and RQ2 in Section 5. The following topics describe the four categories.

All High thresholds. As the name suggests, this category present high thresholds for all analyzed metrics when comparing the derived thresholds of all domains studied. Columns 3 to 5 of Table 5 present the thresholds for the three domains (Accounting, Business and Science & Engineering) that composes this category. These data show that the 5% largest classes (i.e., 95% threshold) in Accounting systems have 981 or more lines of code. Interestingly, the 95% threshold for Business is very similar to Accounting; that is, 928 lines of code. We speculate that the similarity of Accounting and Business systems might be because these domains involve heterogeneous systems in broader trade fields. In addition, systems in this category tend to be large and complex. For instance, 10% classes in Accounting systems with the highest lack of cohesion (i.e., 90% threshold) present LCOM equals or higher than 317. This value is the highest 90% threshold for LCOM among all domains. The reason for this lack of cohesion in Account systems might be due to the several unrelated functionalities controlled by this kind of systems. For instance, Account systems deal with several changes of tax and rules. We believe that these types of functionalities are loosely coupled, which might result in low cohesive systems.

In the Science & Engineering domain, thresholds are not the highest ones for any metric, compared to the other domains of this category. However, these values are still high for LOC, NOM and WMC compared to domains in other categories (see Table 5). In fact, Science & Engineering systems usually involves complex or extensive computations increasing lines of code and complexity. Therefore, these systems naturally belong to this category.

High size thresholds. This topic discusses thresholds derived for domains with large classes (i.e., many Lines of Code and Number of Methods), but low complexity (i.e., few Weighted Method per Class and Coupling Between Objects) when compared to the systems of our dataset. This category includes four domains: Games, Health, Messaging, and Restaurant. Table 5 also shows the threshold values for these four domains in columns labeled *High Size Thresholds*. Comparing thresholds of the previous category with the ones in this category, we observe that thresholds do not vary much for the size metrics (LOC, NOA, and NOM). For instance, all eight domains in these categories have 18 attributes or more for the 95% NOA metric threshold; the exception is Health systems with a 16-attribute cut for 95% threshold. In general, systems in these categories also have more lines of code than systems in the other two categories (see next topics). For instance, apart from Restaurant, the systems from this category show more than 382 LOC for the top-5% largest classes. Classes in Restaurant systems are not much large in terms of LOC, but they have high threshold values for the other size metrics (i.e., NOA and NOM). Hence, we decide to classify this domain in this category instead of the last one with low thresholds.

Systems in this category are not highly complex in terms of coupling (CBO), cohesion (LCOM), and weighted methods by cyclomatic complexity (WMC) when compared to the first category. For instance, while the 95% thresholds of LCOM are above 600 for three out of four domains in the previous category, they are below 400 for four domains in this category. These results can be explained by the fact that systems in this category usually involve simple, yet large, functionalities. In addition, some of these domains, such as Messaging and Restaurant, usually involve a clear set of simple requirements; i.e., message and media exchange for messaging systems or order registration and inventory management for restaurant systems.

Health and Games systems have some particularities because they present high coupling and high cyclomatic complexity, respectively, compared to the other domains in this category. We speculate that Health systems present higher CBO thresholds because they might involve several cases (e.g., many symptoms) to reach a conclusion (e.g., a disease). Therefore, these systems seem highly coupled to other classes, such as domain-specific API classes, using an initial code to provide the basic structure and requirements. In the Games domain case, the result is expected since games usually are computationally extensive involving long if-then-else statements (or worst, long switch-case statements) [22]. In fact, we verified that methods in games are commonly large and complex, although each class has few methods. Therefore, this common practice for gaming development contributes to higher WMC values, but lower values for LCOM and NOM.

High Complexity. The current category includes domains with small classes, yet high complexity. Four domains fall into this category: Development, Free Time, Dictionary, and Localization. Columns labeled *High Complexity* in Table 5 present the thresholds derived for systems of this category. Software systems in this category usually have classes with high thresholds for complexity metrics, although thresholds are not very high for size metrics. For instance, the 5% largest classes in systems of this and the previous category have about 450 or more lines of code. Hence, they have similar size in terms of LOC. On the other hand, systems in this category are more complex than systems of the previous category at least in CBO metrics. If we observe the 95% thresholds for CBO, for instance, we see that these values are always higher than 21 in this category and lower than 21 in the previous category.

It is interesting to observe, however, that the Dictionary and Localization domains have some commonalities since they present similar variation of thresholds for all metrics in general. For example, the 10% and 5% classes with the highest complexity in Dictionary domain present WMC very similar to the ones in Localization domain; that is, around 63 and 111, respectively. In contrast, despite being in the same category, Development systems often have lower thresholds than Dictionary and Localization systems for all metrics. We decided to classify Development in the

Table 5: Metric Threshold per Domain Group

		All High Thresholds			High Size Thresholds				High Complexity Thresholds				All Low Thresholds			
Metric	**%**	**Acc**	**Bus**	**ScE**	**Gam**	**Hea**	**Mes**	**Rest**	**Dev**	**FT**	**Dic**	**Loc**	**Com**	**EC**	**Edu**	**Hom**
LOC	90	541	527	456	310	289	296	255	287	314	337	345	303	160	238	248
	95	981	928	718	491	442	447	382	450	464	575	589	467	232	378	403
NOA	90	14	12	11	11	10	11	12	8	9	11	10	10	8	9	8
	95	21	21	19	18	16	18	19	13	14	17	17	16	11	14	13
NOM	90	34	30	23	20	19	18	18	19	19	21	21	18	17	19	17
	95	46	53	36	31	29	27	25	29	29	33	31	26	24	29	27
WMC	90	123	98	88	59	43	52	41	51	49	63	75	52	30	43	44
	95	236	232	148	100	73	81	62	86	80	111	132	92	49	73	76
LCOM	90	317	253	148	120	136	126	134	117	104	136	159	85	91	120	91
	95	741	742	392	330	351	344	256	302	276	405	378	219	218	311	248
CBO	90	15	16	16	13	15	12	11	15	16	15	15	14	13	14	13
	95	23	23	23	18	20	18	15	22	21	22	21	20	17	20	19
DIT	90	3	4	4	4	3	5	6	4	4	5	4	4	2	3	3
	95	5	5	5	5	5	6	6	5	5	6	5	4	2	4	4
NOC	90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	95	0	0	1	1	0	0	0	1	1	1	1	0	0	0	0

current category, instead of the last one, because systems in this domain seem more complex than systems in the last category for at least two out of three complexity metrics (WMC and CBO).

All Low Thresholds. This category consists of domains with low thresholds for most of the eight metrics compared to the other domains/categories. It is composed by four domains (Communication, E-Commerce, Education, and Home) presented in the last four columns of Table 5. In particular, size metrics have lower thresholds with respect to the first and second categories, while complexity metrics have lower thresholds compared to the first and third categories. Communication and Home domains have larger systems in terms of LOC than E-Commerce and Education, the other two domains in this category. For instance, the first two domains have more than 403 LOC defined for the 95% threshold, while the last two have 232 LOC and 378 LOC, respectively. This observation suggests that E-Commerce and Education share the common characteristics of holding the smallest classes in terms of LOC considering all 15 analyzed domains.

Although not as small and simple as E-Commerce and Education, the Communication and Home domains also have small and simple classes. Classes in these domains have a particularly small number of methods and attributes, when compared to other domains. For instance, focusing on the 90% thresholds in Table 5, we observed that Communication and Home domains have classes with up to 18 and 17 methods (NOM), respectively. Their 90% threshold values are 10 and 8 attributes (NOA), respectively. These two domains also have simple classes in terms of WMC and LCOM. With respect to coupling, these domains are in the middle; i.e., neither the lowest nor the highest thresholds for CBO. This result is somehow expected for Communication because systems in this domain manage relationship about different sub-systems, such as in a Client-Server architecture [2].

5 DISCUSSIONS

This section discusses the first three research questions.

5.1 RQ1: Threshold Variation across Domains

First, we investigate if thresholds for the same metrics vary on different software domains. If so, it supports our assumption that software domains should be considered when building benchmarks for metrics-based quality assessment. To support our discussion, we compare the highest to the lowest thresholds of each metric. Table 6 presents for each metric the lowest to the highest thresholds among the 15 domains. It also shows, between parentheses, how many times the highest threshold is greater than the lowest threshold. For instance, the lowest 95% threshold for the LOC metric is 232 (see E-Commerce in Table 5), while the highest 95% threshold for the LOC metric is 981 (see Accounting in Table 5). The difference is that the Accounting threshold is 4.2 times higher than the E-Commerce threshold in this case.

The results in Table 6 show that the investigated thresholds largely vary for all metrics; except Number of Children (NOC), which we discuss in Section 5.2. Apart from NOC, all metrics have a significant variation in the 90% and 95% thresholds. The metrics with higher threshold variations are LCOM, WMC, and LOC. The rate between the highest and the lowest thresholds in a domain is about 3.4 times or higher for these 3 metrics. This large difference between the highest and lowest thresholds suggests that LCOM, WMC, and LOC are highly sensitive to the software domain. Therefore, developers should be aware of this variation when using thresholds for these metrics in software quality evaluation.

Table 6: Difference between Highest and Lowest Thresholds

%	LOC	NOA	NOM	WMC	LCOM	CBO	DIT	NOC
90	160-541	8-14	17-34	30-123	85-317	12-16	2-6	0-0
	(3.4)	(1.7)	(2.0)	(4.1)	(3.7)	(1.3)	(3.0)	(n/a)
95	232-981	11-21	24-53	49-236	218-742	15-23	2-6	0-1
	(4.2)	(1.9)	(2.2)	(4.8)	(3.4)	(1.5)	(3.0)	(n/a)

** Numbers between parentheses mean how many times the highest value is greater than the lowest value*

The rate between the highest and the lowest thresholds for NOA, NOM, and CBO is about 2 times. This difference suggests that these metrics are not as sensitive to the software domain as LCOM, WMC,

and LOC. Yet, a difference of twice can be considered large, depending on the evaluation goal. The DIT metric is an interesting case. The threshold of DIT did not present large variation among domains. For instance, the 95% thresholds of DIT varied between 3 and 6 (2 times) for 14 out of 15 domains. The only exception was E-commerce with both 90% and 95% thresholds of 2 for DIT (see Table 5). Therefore, we consider the variation of DIT moderate.

Answer to RQ1: Thresholds for the same metric may vary from 1.3x (CBO) to 4.1x (WMC) for the 90% cut, and from 1.5x (CBO) to 4.8x (WMC) for the 95% cut. We have not observed high variation of thresholds only for Number of Children (NOC). Therefore, we conclude that the metric thresholds are typically sensitive to the software domain.

5.2 RQ2: Similar Thresholds across Domains

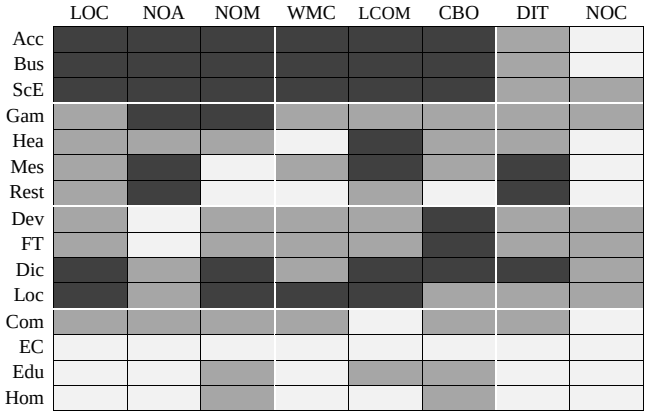
We analyze similar thresholds across different domains for each metric. The main reason for this analysis is to identify whether and which domains can be grouped to promote more reliable benchmarks. In addition, if different domains have similar thresholds for the same metric, it means that some metric thresholds can be reused across domains.

For this analysis, we cluster thresholds for each metric into three levels according to how high these values are. Table 7 uses grey scale to indicate domains (rows) with low, medium, and high thresholds for each metric (columns). In Table 7, light grey boxes mean lower thresholds, grey boxes mean medium thresholds, and dark grey boxes mean high thresholds. Based on Table 5, we jointly analyzed both the 90% and 95% thresholds to determine if the metric threshold is low, medium, or high for a domain. For instance, Accounting domain has high thresholds for 6 metrics (LOC, NOA, NOM, WMC, LCOM and CBO), medium thresholds for DIT, and low thresholds only for NOC.

It is interesting to observe that Accounting and Business domains have the same levels of thresholds for all eight metrics (i.e., same colors in corresponding boxes). This result means that these two domains are very similar in terms of measurement. Therefore, if someone derives thresholds for a set of Accounting systems, these thresholds are expected to be reliable to be used for detecting technical debts in Business systems, for instance. As discussed before, one reason for this similarity is because both domains involve broader trade fields. In addition, they are usually large and complex systems as indicated by high threshold for most metrics.

Other pairs of similar domains in terms of metric thresholds are: Education and Home, Free Time and Development, Dictionary and Science & Engineering, and Dictionary and Localization. The explanation for similar thresholds vary in a case by case basis. For instance, Education and Home in our dataset have some of the smallest and simplest systems. Therefore, these domains share low thresholds for most metrics. On the other hand, Free Time and Development tend to be heterogeneous, but both are used in many situations. For instance, Free Time is often used for entertainment and leisure purposes, while development must support a variety of scenarios for completing development projects. An interesting similarity is between Dictionary and Localization systems, although these domains deal with completely different requirements, we observed that they share some similar coding styles. For instance, Dictionary and Localization systems have large methods coupled to several API classes. This characteristic makes them similar in terms of metrics and thresholds. This explanation can also be given to the similarity between Dictionary and Science & Engineering.

Table 7: Low, Medium and High Levels for each Metric



Number of Children (NOC) has similar – and low – thresholds for all domains. In fact, the 90% thresholds are always 0 for all domains while the 95% threshold is either 0 or 1. We observed in the distribution of NOC that its values only vary largely if we select the 3% to 1% classes with more children (i.e., the 97% to 99% thresholds). This result means that only 1% to 3% of classes in a system usually have more than one subclass. Therefore, when defining threshold for NOC, someone needs to consider this particularity of this specific metric.

Answer to RQ2: Some domains have similar thresholds for the same metrics. This result implies that these domains can be grouped to promote more reliable benchmark-based threshold derivation. Accounting and Business are examples of domains with similar thresholds for all metrics.

5.3 RQ3: Impact of System Size on Thresholds

This section discusses if the size of the systems which compose a benchmark affects the metric thresholds. It so, in addition to domains, the system size should be considered when building benchmarks for quality assessment. For instance, metrics-based code smell detection tools should consider the size of the systems in this case. Otherwise, they might use overly high thresholds for small systems, and vice versa.

To investigate RQ3, we build four balanced benchmarks with different system sizes and compare the derived thresholds. The first one is Qualitas Corpus [32] which is a benchmark composed of 111 large Java systems, such as Eclipse, Netbeans, and Ant. The second one is the AllSystems benchmark which includes all 3,107 heterogeneous systems of this study that we mined from GitHub. The third benchmark, named MediumSystems, is a subset of AllSystems comprising 20 systems per domain and, making 300 systems in total. Our aim was to exclude outlier (too large or too small) systems in this benchmark of medium-sized systems. Therefore, we only selected 10 systems immediately above and 10

systems immediately below the median of each domain in terms of LOC. The fourth benchmark, SmallSystems, is also a subset of AllSystems with 20 systems per domain (i.e., 300 systems in total). However, in this case we randomly selected systems below the median LOC for each domain. For instance, all 20 Health systems in SmallSystems have between 1,000 LOC (selection quality criterion) and 4,786 LOC. Since three domains (E-Commerce, Free Time, and Science & Engineering) have less than 40 systems and we have the criterion of 20 systems per domain, SmallSystems has exactly the 20 smallest systems for these three domains.

Table 8 presents the results of the 95% thresholds for the four benchmarks described in the previous paragraph. This table focuses on 7 metrics because we have not observed difference for NOC; i.e., its 95% thresholds are either 0 (SmallSystems and MediumSystems) or 1 (Qualitas and AllSystems). The results in Table 8 shows that larger systems have higher thresholds for all metrics. For instance, in the case of LOC, the 95% thresholds are 602, 599, 315, and 286 for Qualitas Corpus, AllSystems, MediumSystems, and SmallSystems, respectively.

Table 8: Thresholds for Different Benchmarks

	LOC	NOA	NOM	WMC	LCOM	CBO	DIT
Qualitas	602	16	34	111	406	20	6
AllSystems	599	18	37	125	475	22	5
MediumSyst.	315	13	23	59	168	17	4
SmallSystems	286	12	20	52	129	15	4

In fact, we already expected that larger systems have higher thresholds for size metrics, such as LOC, NOA, and NOM. However, it is interesting to observe that the system size has also affected the thresholds of complexity and inheritance metrics. This result contradicts previous studies [10, 33] that claim metrics like CBO and DIT do not have a high correlation with LOC. In fact, we observe that larger systems have both more complex classes and denser use of inheritance relationships. In addition, we observed in Table 8 that Qualitas Corpus and AllSystems have similar thresholds, although they do not have any system in common. This observation suggests that if the benchmark is composed of a high enough number of heterogeneous systems (i.e., from different domains and sizes), the metrics thresholds tend to be comparable.

Answer to RQ3: The size of the systems that compose the benchmark influences the metric thresholds. Benchmarks with larger systems yield higher thresholds for all metrics. Furthermore, benchmarks composed of many heterogeneous systems in terms of size and domains tend to have similar thresholds.

6 CODE SMELL DETECTION EVALUATION

Code smells describe a technical debt where there are hints that suggest a flaw in the source code [31]. For instance, one of the most well-known code smell, God Class, is defined as a class that knows or does too much in the software system [9, 12]. God Class is a strong indicator of technical debt because this component is aggregating functionality that should be distributed among several components. In fact, previous work has found that this code smell is related to maintenance problems, such as bugs [11], design flaws [24, 25], and instability [8].

This section evaluates the God Class detection by comparing the thresholds derived from each domain with the thresholds derived from AllSystems benchmark. The first set we call domain-specific thresholds and the second set we call generic thresholds. The evaluation consists of comparing precision and recall of both sets in God Class detection. To perform this evaluation, we followed four steps. First, we randomly selected 43 systems from all 15 domains explored in this study and built an oracle of God Class instances for these systems (Section 6.1). Second, we used the metric-based strategy to identify God Classes and, then, to compute precision and recall (Section 6.2). Finally, we analyzed the effectiveness results for detecting code smells to answer RQ4 (Section 6.3).

6.1 Dataset for Code Smell Analysis

To build the Smell Dataset, we randomly selected 43 systems using two criteria: (i) at least two systems per domain; (ii) systems that compiles in Eclipse IDE. The first criterion is to have a sample that covers all domains and the second one is to address tool support limitations. This dataset is then composed by 3 systems per domain, except two domains (E-Commerce and Free Time) which we only found 2 systems that match the second criterion.

Oracle Creation. For each system of our Smell Dataset, we built an oracle of true positive instances. The oracle can be understood as the reference list of the actual smells found in a system. This oracle is the basis for determining whether the derived thresholds are effective in the identification of God Classes. In order to provide a reliable oracle, we run three well-known code smell detection tools (JDeodorant [20], JSpirit [35], and PMD [11]) for all target systems and build a list with possible anomalies pointed by these tools. Then, at least a pair of authors analyzed each class pointed as God Class to validate our oracle. This manual validation consisted of the answer of four questions (Does the class have more than one responsibility? Does the class have functionality that would fit better into other classes? Do you have problems summarizing the class responsibility in one sentence? Would splitting up the class improve the overall design?) with a confidence rate varying from 1 to 5. These four questions were based on questions of a previous study [29]. In the cases we had a disagreement between the two evaluators or an average confidence score small than 3, a third evaluator checked the class and the three evaluators discussed to reach a consensus. The oracle validation questions and the complete reference list can be found in our support website [30].

6.2 Metric-based Detection Strategy and Measurement of Effectiveness

Metrics are often too fine-grained to comprehensively quantify technical debt [16]. To overcome this limitation, metric-based detection strategies have been proposed [19]. This work selected and adapted a detection strategy from the literature to identify God Class [24], presented in Fig. 2. There are two main reasons to use such strategy. First, it has been evaluated in other studies and presented good results for the detection of God Class [24]. Second, this detection strategy defines a straightforward way for identifying

instances of God Class by combining four different metrics. In fact, we adapted its original definition based on the metrics we investigate in this paper, but the adapted detection strategy captures the same quality characteristics from the original work [16, 19].

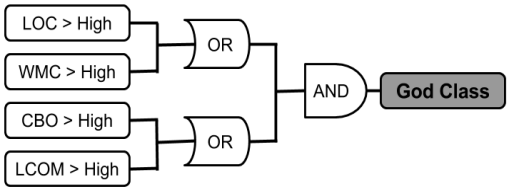


Figure 2: God Class Detection Strategy.

We use precision and recall as a proxy for effectiveness in code smell detection. Recall measures the fraction of relevant classes listed by the detection strategy using a set of thresholds. Relevant classes are classes that appear in the oracle. Precision measures the ratio of correctly detected code smells by the total classes listed. To compute precision and recall we need to know the values of true positives (TP), false positives (FP), and false negatives (FN). TP and FP quantify the number of correctly and wrongly identified code smells by the detection strategy compared to the oracle. FN, on the other hand, quantifies the number of code smells the detection strategy missed out from the oracle. The computation of recall and precision is: Recall = TP / (TP + FN) and Precision = TP / (TP + FP). Recall and precision vary from 0 to 1 and higher values are related to better effectiveness.

6.3 RQ4: Effectiveness of Thresholds for Code Smell Detection

In this section, we answer and discuss RQ4 (Are domain-specific thresholds better than general thresholds for code smell detection?). To do that, we computed precision and recall as just described. Analyzing the results presented in Table 9, domain-specific thresholds did not always get the best results in such evaluation. In terms of precision, domain-specific thresholds were better in 3 cases, and technically equal in other 5 cases. We considered technically equal, cases that the difference is smaller than 5%. In terms of recall, the results are more exciting since domain-specific thresholds are better in 8 cases and technically equal in other 5 cases. We use bold to show which is better in the pairwise comparison.

High precision means that the detection strategy indicated more relevant than irrelevant code smells. High recall, on the other hand, means that the detection strategy was able to identify most code smells in the system. Hence, a large number of false positives (high recall) are preferred over a large number of false negatives (high precision) by software engineers, because manual inspection, which is inevitable, tends to uncover false positives. Therefore, considering that recall is higher for domain-specific thresholds in 8 domains, we conclude that domain-specific thresholds fared better in detecting code smells than generic thresholds.

Still talking about Table 9 and considering the four categories presented in Section 4, higher thresholds got higher precision and lower recall than generic thresholds. Exactly, the opposite result for low thresholds category. For the intermediate categories is harder to conclude something, but it tends to follow low thresholds category where generic thresholds achieved higher precision and domain-specific higher recall. Therefore, the classification of domains presented in Section 4 should be considered for selecting the most appropriate thresholds for software quality evaluation.

Table 9: Precision and Recall using Domain-Specific and Generic Thresholds

		Precision		Recall	
Cat.	**Dom.**	**DS**	**G**	**DS**	**G**
All High	Acc	**1.00**	0.75	0.20	**0.60**
	Bus	0.00	0.00	0.00	0.00
	ScEng	**0.88**	0.82	0.58	**0.75**
High Size	Gam	0.05	0.09	0.67	0.67
	Hea	0.50	**0.75**	**0.36**	0.27
	Mes	0.40	**1.00**	**0.40**	0.20
	Res	**0.25**	0.00	**0.25**	0.00
High Complextiy	Dev	0.67	**0.76**	0.94	0.94
	FT	0.43	**0.75**	0.50	0.50
	Dic	0.00	0.00	0.00	0.00
	Loc	0.64	0.67	**0.69**	0.46
All Low	Com	0.58	**0.68**	**0.83**	0.72
	EC	0.03	0.00	**0.50**	0.00
	Edu	0.54	**0.80**	**0.54**	0.31
	Hom	0.79	**0.89**	**0.85**	0.62

Answer to RQ4: In terms of recall, domain-specific thresholds are usually better than generic thresholds for most domains.

7 THREATS TO VALIDITY

Our empirical study, like others, has some potential threats to validity. Hence, we discuss the main actions we have taken to mitigate their impact on the research results.

Internal and Conclusion Validities. There are two main threats to internal validity: the selected domains and measurements. Regarding the first threat, we might have not selected the best or more representative software domains. We selected these domains because they are consolidated and we expect to get high quality and frequently used systems. In addition, we have also filtered systems with more than 1,000 lines of code. For measurements threat, we may get false or wrong measures for the target software systems. We quantify eight metrics for all systems that compose this study and we also derive thresholds for these metrics. Aiming to make these measurement processes easier, we used a specific tool presented in another study [34]. In addition, we also made some tests to check if the results of this tool were as expected.

External and Construction Validities. There are four external threats to validity of our study. First, it is not possible to ensure that the select systems reflect the best samples of the recurrent practices. To reduce this risk, we filter the total amount of systems by number of stars and lines of code. Second, the systems that compose our dataset are developed in Java. However, for other programming languages or technologies similar results can be found. Third, our

results are restricted for the set of metrics we selected, but we believe that similar results can be also found for metrics that quantify similar quality attributes or characteristics. Finally, even though we have presented a large-size study, additional replications are necessary to determine if our findings can be generalized to other domains and dataset of systems. We described the main actions to minimize possible threats regarding the oracle creation, detection strategies, and how we built our benchmarks in Sections 6.1 and 6.2.

8 CONCLUSIONS AND FUTURE WORK

This paper presented an empirical study on domain-specific thresholds. We conducted the study by selecting and measuring 3,107 software systems from 15 software domains. Once we have the measurements, we derived 90% and 95% thresholds for each metric per domain and analyzed them in different ways. For instance, we compared the thresholds among domains and investigated the effectiveness of code smell detection between domain-specific and generic thresholds. The results indicate that metric thresholds are sensitive to software domain. For example, some metrics may vary across domains from 1.5x to 4.8x for the 95%. Moreover, we observed that not only the domains, but also the size of the systems that compose the benchmark is a factor that affect the metric thresholds. That is, the results corroborate with the claim that benchmarks composed of heterogeneous systems tend to have similar thresholds. Finally, in terms of recall, we collect evidence indicating that domain-specific thresholds are better than generic thresholds on average for code smell detection.

For future work, we plan further investigation with additional technical debts and additional replications of this study to determine whether our findings can be generalized to other domains and systems. In addition, we are also considering commercial software systems to confirm whether and how thresholds vary across domains and their impact on technical debts.

ACKNOWLEDGEMENTS

This research was partially supported by Brazilian funding agencies: CNPq (Grant 290136/2015-6 and 424340/2016-0), CAPES, and FAPEMIG (Grant PPM-00651-17).

REFERENCES

[1] T. Alves, C. Ypma, J. Visser. "Deriving Metric Thresholds from Benchmark Data". In Proceedings of 26th International Conference on Software Maintenance (ICSM), pp. 1–10, 2010.

[2] F. Buschmann, R. Meunier, Ha. Rohnert, P. Sommerlad, M. Stal. "Pattern-Oriented Software Architecture: A System of Patterns". Volume 1, Wiley, 1996.

[3] S. Chidamber, C. Kemerer. "A Metrics Suite for Object Oriented Design". IEEE Transactions on Software Engineering, vol. 20, Issue 6, 476–493, Jun., 1994.

[4] CK Tool, available at https://github.com/mauricioaniche/ck

[5] D. Coleman, B. Lowther, and P. Oman. "The Application of Software Maintainability Models in Industrial Software Systems". Journal on System Software, vol. 29, Issue 1, pp. 3–16, Feb., 1995.

[6] G. Concas, M. Marchesi, S. Pinna, and N. Serra. "Power-Laws in a Large Object-Oriented Software System". IEEE Transactions on Software Engineering, vol. 33, Issue 10, pp. 687–708, Oct., 2007.

[7] T. DeMarco. "Controlling Software Projects: Management, Measurement, and Estimates". Prentice Hall, 1986.

[8] E. Fernandes, G. Vale, L. Sousa, E. Figueiredo, A. Garcia, and J. Lee. "No Code Anomaly is an Island Anomaly Agglomeration as Sign of Product Line Instabilities". In proceedings of the 16th International Conference on Software Reuse (ICSR), pp. 48-64, 2017.

[9] E. Fernandes, J. Oliveira, G. Vale, T. Paiva and E. Figueiredo. "A Review-based Comparative Study of Bad Smell Detection Tools". In proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE), 2016.

[10] K. Ferreira, M. Bigonha, R. Bigonha, L. Mendes, H. Almeida. "Identifying Thresholds for Object-Oriented Software Metrics". Journal of Systems and Software, vol. 85, Issue 2, pp. 244–257, Feb., 2012.

[11] F. Fontana, M. Zanoni, A. Marino, and M. Mantyla. "Code Smell Detection: Towards a Machine Learning-based Approach". In Proceedings of the 29th International Conference on Software Maintenance (ICSM), 2013.

[12] M. Fowler. "Refactoring: Improving the Design of Existing Code". In Addison-Wesley Professional, 1999.

[13] S. Herbold, H. Grabowski and S. Waack. "Calculation and optimization of thresholds for sets of software metrics". Empirical Software Engineering, vol. 16, Issue 6, pp. 812–841, Dec. 2011.

[14] D. Hubbard. "How to Measure Anything: Finding the Value of "Intangibles" in Business". John Wiley & Sons, 2014.

[15] B. Kitchenham, "What's up with software metrics? – A preliminary mapping study". The Journal of Systems and Software, vol. 83, Issue 1, pp. 37–51, 2010.

[16] M. Lanza, R. Marinescu. "Object-Oriented Metrics in Practice". Springer, 2006.

[17] M. Linares-Vásquez, S. Klock, C. McMillan, A. Sabané, D. Poshyvanyk, and Y. Guéhéneuc. "Domain matters: bringing further evidence of the relationships among anti-patterns, application domains, and quality-related metrics in Java mobile apps". In Proceedings of the 22nd International Conference on Program Comprehension (ICPC), pp. 232-243, 2014.

[18] M. Lorenz and J. Kidd. "Object-Oriented Software Metrics: A Practical Guide". Prentice Hall, 1994.

[19] R. Marinescu. "Detection strategies: metrics-based rules for detecting design flaws". In Proceedings of the 20th International Conference on Software Maintenance (ICSM), pp. 350–359, 2004.

[20] D. Mazinanian, N. Tsantalis, R. Stein, and Z. Valenta. "JDeodorant: clone refactoring". In Proceedings of the 38th International Conference on Software Engineering Companion (ICSE). 2016.

[21] T. McCabe. "A complexity measure". IEEE Transactions on Software Engineering, vol. 2, Issue 4, pp. 308–320, 1976.

[22] E. Murphy-Hill, T. Zimmermann, and N. Nagappan. "Cowboys, ankle sprains, and keepers of quality: how is video game development different from software development?" In Proceedings of the 36th International Conference on Software Engineering (ICSE), pp.1-11, 2014.

[23] P. Oliveira, M. Valente, and F. Lima. "Extracting relative thresholds for source code metrics". In Proceedings of the 18th International Conference on Software Maintenance and Reengineering (CSMR), pp 254–263, 2014.

[24] W. Oizumi, A. Garcia, L. Sousa, B. Cafeo, and Y. Zhao. "Code anomalies flock together: exploring code anomaly agglomerations for locating design problems". In Proceedings of the 38th International Conference on Software Engineering (ICSE), 2016.

[25] J. Padilha, J. Pereira, E. Figueiredo, J. Almeida, A. Garcia, and C. Sant'Anna. "On the Effectiveness of Concern Metrics to Detect Code Smells: An Empirical Study". In proceedings of the 26th International Conference on Advanced Information Systems Engineering (CAiSE), 2014.

[26] P. Perkusich, A. Medeiros, L. Silva, K. Gorgônio, H. Almeida, and A. Perkusich. "A Bayesian network approach to assist on the interpretation of software metrics". In Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC), 2015.

[27] B. Ray, D. Posnett, V. Filkov, and P. Devanbu. "A large scale study of programming languages and code quality in GitHub". In Proceedings of the International Symposium on Foundations of Software Engineering (FSE), 2014.

[28] D. Spinellis. "A tale of four kernels. In Proceedings of the 30th International Conference on Software Engineering (ICSE), pp. 381–390, 2008.

[29] J. Schumacher, N. Zazworka, F. Shull, C. Seaman, and M. Shaw. "Building empirical support for automated code smell detection". In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2010.

[30] Support Website, Available at https://labsoft-ufmg.github.io/techdebt-2018/.

[31] TechDebt 2018 - International Conference on Technical Debt Website, Available at https://2018.techdebtconf.org/track/TechDebt-2018-papers.

[32] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. "Qualitas corpus: a curated collection of java code for empirical studies". In Proceedings of the 17th Asia Pacific Software Engineering Conference (APSEC), pp. 336–345, 2010.

[33] G. Vale, D. Albuquerque, E. Figueiredo, and A. Garcia. "Defining metric thresholds for software product lines: a comparative study". In Proc. of the International Software Product Line Conference. (SPLC), pp. 176–185, 2015.

[34] L. Veado, G. Vale, E. Fernandes, and E. Figueiredo. "TDTool: threshold derivation tool". In proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE), Tools Session, 2016.

[35] S. Vidal, H. Vazquez, J. A. Diaz-Pace, C. Marcos, A. Garcia and W. Oizumi, "JSpIRIT: a flexible tool for the analysis of code smells". In Proc. of the International Conference of Chilean Computer Science Society (SCCC), 2015.