Towards the Definition of Domain-Specific Thresholds

Allan Mori[1], Eduardo Figueiredo[1], Elder Cirilo[2]

[1]Computer Science Department, Federal University of Minas, Brazil

[2]Computer Science Department, Federal University of São João Del Rei, Brazil

{allanmori,figueiredo}@dcc.ufmg.br, elder@ufsj.edu.br

ABSTRACT

Software metrics provide basic means to quantify several quality aspects of information systems. However, the effectiveness of the measurement process is directly dependent on the definition of reliable thresholds. To define appropriate thresholds, we need to consider characteristics of the information systems, such as their size and domain. There are several studies to propose methods to derive thresholds and evaluate them. However, we still lack empirical knowledge about whether and how thresholds vary across different information system domains. To tackle this limitation, this paper investigates specific thresholds in four information system domains: accounting, e-commerce, health, and restaurant. Our study relies on 40 information systems to derive domain-specific thresholds for 9 well-known software metrics. Our results indicate that lower-bound thresholds (e.g., 15% smaller classes) usually do not significantly vary across domains. However, for all analyzed metrics, upper-bound thresholds (e.g., 5% largest classes) are different in some domains. Moreover, our study also suggests that domain-specific thresholds are more appropriated than generic ones. For instance, we observed in our analysis that the more appropriated threshold to select the 5% largest classes is 290 LOC in health systems and 147 LOC in accounting systems.

CCS Concepts

• Software and its engineering → Empirical software validation

Keywords

Software Metrics; Thresholds; Software Domain

1. INTRODUCTION

Software metrics are the pragmatic means for assessing different quality attributes of information systems, such as maintainability and changeability [5]. Certain metric values can help to reveal specific parts of the information system that should be closely monitored [8]. For instance, measures can indicate whether a critical anomaly is affecting the software structure of an information system. This way, developers may suspect that something is wrong in the system design or implementation. Typical examples of anomalies include cases of large classes, long methods, and long parameter lists [7].

Nevertheless, the effective measurement of information systems is directly dependent on the definition of appropriate thresholds [1][16]. Thresholds allow us to objectively characterize or classify each class (or method) according to one of the quality metrics. We argue in this paper that the definition of appropriate thresholds needs to be tailored to each metric and characteristics of the measured systems, such as their size and domain. For instance, a health system might be more complex and, therefore, it may have higher metric values, than an e-commence system.

In the past few years, thresholds were calculated based on experience of software engineers or using a single system as reference [5][11]. Recently, thresholds have been derived from a set of similar systems, named benchmarks, and calculated based on systematic methods [17]. A systematic method should provide the same thresholds for a metric if the same input is used. The idea behind the use of benchmarks is to use information from similar systems to help derive meaningful thresholds.

In this paper, similar systems mean that they were developed in the same programming language and that they belong to the same domain, even though they might have been developed by different people. The basic idea of using benchmark-based threshold derivation is to get common characteristics of the majority of components in a domain. Therefore, discrepant values might indicate a problem in that specific domain. For instance, if almost all classes in a set of e-commerce systems have less than 100 lines of code (LOC), the minority of classes with more than 100 LOC are outliers. On the other hand, this threshold might be 120 LOC for a set of health systems, for instance.

This paper presents an empirical study to investigate the hypothesis that thresholds vary among information systems of different domains since these systems have different degrees of complexity, cohesion, and size. To evaluate this hypothesis, we rely on four benchmarks composed by 40 object-oriented information systems. Each benchmark includes 10 information systems in one of the following domains: accounting, e-commerce, health, and restaurants. We apply a set of nine well-known metrics [5] [8] [9] to each system, namely Lines of Code (LOC), Number of Attributes (NOA), Number of Methods (NOM), Weighted Method per Class (WMC), Coupling between Objects (CBO), Lack of Cohesion in Methods (LCOM), McCabe Cyclomatic Complexity (McCabe), Depth of Inheritance Tree (DIT), and Number of Children (NOC). After aggregating metrics per domain, we derived thresholds for each metric by using a recently proposed systematic method [17].

We finally compare the derived thresholds of each metric among the four domains. In short, the results indicate that lower-bound thresholds (e.g., 15% smaller classes) usually do not significantly

SBSI 2017, June 5th–8th, 2017, Lavras, Minas Gerais, Brazil.

vary across domains. However, for all analyzed metrics, upper-bound thresholds (e.g., 5% largest classes) are clear different in some domains. For instance, we observed in our analysis that the threshold to select the 5% largest classes is 290 LOC in health systems and 147 LOC in accounting systems.

Finally, we compared the derived metrics with two others benchmarks: (i) Qualitas Corpus [15] and SPL Benchmark [16]. The Qualitas Corpus is composed of more than a hundred real object-oriented Java systems. The SPL Benchmark, on the other hand, has 33 software product lines implemented in different languages and technologies, namely: AHEAD [4] and Feature House [3]. As a result, we observed that our derived thresholds seem to be more equalized to the reality of each domain (account, e-commerce, health, and restaurant) than to the general thresholds derived from the others two benchmarks.

The remainder of this paper is organized as follows. Section 2 presents a background on metrics and threshold derivation. Section 3 describes the study configuration and set up, including our goal, research question, and the selection of systems and domains. Section 4 presents and discusses the results of measurements per domain. Section 5 show visualizations for the thresholds found. Section 6 compares our results with other two papers on threshold derivation. Section 7 correlates our study with papers from the literature. Section 8 discusses threats to the study validity. Finally, Section 9 concludes this paper and points out directions for future work.

2. DERIVING THRESHOLDS FOR SOFTWARE METRICS

In this section, we present 9 well-known software metrics analyzed in the study (Section 2.1). We also provide some background about methods to derive thresholds (Section 2.2), including a short explanation of the Vale's method used in our work to derive domain-specific thresholds (Section 2.3).

2.1 Software Metrics

Software metrics are the pragmatic means for assessing different quality attributes of information systems, such as maintainability and changeability [5]. This study investigates domain-specific thresholds for nine metrics: Lines of Code (LOC), Number of Attributes (NOA), Number of Methods (NOM), Weighted Method per Class (WMC), Coupling between Objects (CBO), Lack of Cohesion in Methods (LCOM), McCabe Cyclomatic Complexity (McCabe), Depth of Inheritance Tree (DIT), and Number of Children (NOC). We choose these metrics because they capture different attributes of information systems, such as size, complexity, cohesion, and inheritance relationships. In addition, they are well-known object-oriented software metrics [5][8][9].

Size Metrics. *Lines of Code (LOC)* [9] measures the number of lines of code per class. It counts neither comment lines nor blank lines. The value of this metric indicates the size of a class. *Weighted Method per Class (WMC)* [5] counts the number of methods in a class weighting each method by its cyclomatic complexity. *Number of Methods (NOM)* and *Number of Attributes (NOA)* quantifies the number of methods/constructors and the number of fields/class variables, respectively. These metrics are mainly used to estimate the size of a class.

Coupling, Cohesion and Complexity Metrics. *Coupling between Objects (CBO)* [5] counts the number of classes called by a given class. CBO measures the degree of coupling among classes based on method calls and attribute accesses. *Lack of Cohesion in Methods (LCOM)* [5] computes the difference between (i) the pairs of methods in a class that do not access any attribute in common and (ii) the pairs of methods in a class that do access attributes in common. This metric measures the cohesion of methods of a class in terms of the frequency that they share attributes. *McCabe Cyclomatic Complexity (McCabe)* [11] counts the number of linearly independent paths through a program source code. It is used to indicate the complexity of a program.

Inheritance Metrics. *Depth of Inheritance Tree (DIT)* [5] counts the number of levels that a subclass inherits methods and attributes from superclasses in the inheritance tree of the system. This metric estimates the class complexity with respect to its inheritance relationships. *Number of Children (NOC)* [5] counts the number of direct subclasses of a given class. This metric indicates software reuse by means of inheritance.

2.2 Methods to Derive Thresholds

The effective use of software metrics is dependent on the definition of appropriate thresholds. Thresholds allow us to objectively characterize or to classify each component according to one of the quality metrics. In this paper, we aim to investigate if the definition of appropriate thresholds should be tailored to the characteristics of the measured system and its domain. In the past few years, thresholds were calculated based on experience of software engineers or using a single system as reference [5]. More recently, thresholds have been derived from benchmarks and calculated based on well-defined derivation methods.

For instance, Alves et al. [1] proposed a method that weights 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. [6] 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. The groups are called good, regular, and bad measurements, respectively. In this paper, we rely on a recently proposed method, called Vale's Method (Section 2.3), to derive thresholds [17]. We choose Vale's Method because a software tool that supports this method is available, making the threshold derivation process easier.

2.3 Deriving Thresholds with Vale's Method

Vale's Method proposes the threshold derivation in five steps [17], as illustrated in Figure 1. First, metrics have to be extracted from a benchmark of software systems. In this step, we employed two tools to measure the source code we used, namely Metrics Plugin 1.3.6[1] and Code Pro Analytix 3.6[2]. For each entity, the method computes the weight percentage within the total number of entities in the second step. Then, Vale's Method sorts the metric values in ascending order and takes the maximum metric value that represents 1%, 2%, up to 100%, of the weight. In the fourth step, it aggregates all entities per metric value. Finally, the thresholds 3%, 15%, 90% and 95% are derived by choosing the percentage of the overall metric values we want to represent. Apart from the first step, all other steps are supported by a tool called TDTool [19].

[1] http://metrics.sourceforge.net/

[2] https://marketplace.eclipse.org/content/codepro-analytix

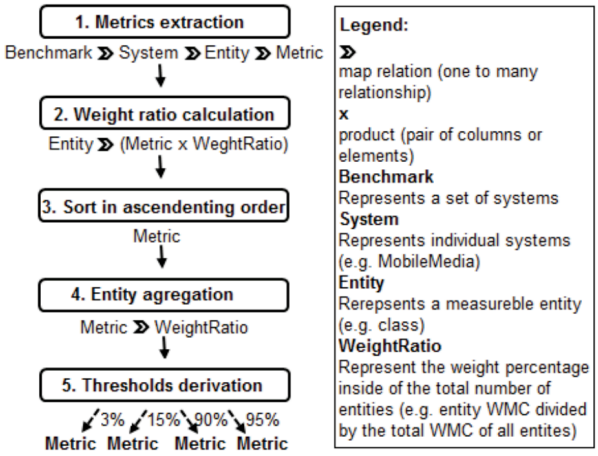

Figure 1. Vale's Threshold Derivation Method.

3. STUDY SETTINGS

In this section, we present the study settings. We present and motivate our research questions (Section 3.1). We then detail how we built the benchmark used in the study to derive domain-specific thresholds (Section 3.2). This section also discusses the main characteristics of each domain in terms of Lines of Code and Number of Classes.

3.1 Goal and Research Questions

This study aims to investigate whether and how metric thresholds vary across information systems of different domains. Therefore, we design and execute an empirical study with 40 information systems of four domains in order to answer the following research questions (RQ).

RQ1. What are the thresholds for metrics in each specific domain?

RQ2. Which software attributes (size, complexity and cohesion) benefit most from domain-specific thresholds?

RQ3 Is the Vale's method, when applied to the studied domains individually, able to generate more appropriate thresholds than when applied to general systems?

3.2 Selected Systems and Domains

This study relies on information systems from four domains: accounting, e-commerce, health, and restaurant. We choose such domains for the following reasons. First, information systems from these domains encompass several basic business requirements (e.g., user and product management). Second, there are a significant number of information systems in these domains available for download in GitHub. Third, the four chosen domains are well-defined in terms of requirements and, therefore, we believe that their differences might reflect in varying thresholds among systems of each domain.

We extracted information systems to compose our data set from GitHub[3] repositories. We performed the selection of systems in May 2016. We selected the information systems based on the ranking of the most starred systems and their length in terms of lines of code. In GitHub, stars are a meaningful measure for repository popularity among the platform users, and they may support the selection of relevant systems for study.

[3] https://github.com/

To minimize the risk of biasing our results we applied a strict set of criteria. First, we collected 400 Java systems from GitHub, 100 for each domain in the descending order of stars. Then, we discarded non-Java information systems, since GitHub does not automatically verify the main programming language of each result. For instance, we removed Java projects developed for Android, because these systems tend to have a different architectural design. Finally, we excluded systems with less than 300 lines of code (LOC) because we considered them simple toy examples. Finally, to balance our data set, we randomly select 10 information systems of each domain. Therefore, our final dataset includes 40 information systems; i.e., 10 systems of each domain.

Table 1. Descriptive Statistics of Lines of Code per Domain

	Min	*Max*	*Mean*	*Std. Dev.*
Accounting	380	6,396	1,361.7	1,756.7
ECommerce	330	2,867	991.4	897.0
Health	636	12,046	2,825.2	3,526.6
Restaurant	354	3,967	2,103.0	1,439.4

Table 1 summarizes the descriptive statistics of the 40 selected information systems, in terms of lines of code. As we can see, the number of lines of code diverges largely across information systems. For example, the smallest one has only 330 LOC, while the biggest system has 12,046 LOC. In the same way, we can observe a considerable discrepancy in terms of lines of code among the studied domains. E-commerce information systems, on average, have near 70% less lines of code than health systems. This considerable difference corroborates with our hypothesis that health systems might be more complex and, therefore, it may present higher metric values. In summary, we consider that all means are representative of medium scale information systems.

We also investigated the distribution of the number of classes over the 40 selected information systems. Table 2 shows the obtained data. In terms of means, we can observe that there is no substantial difference among the studied domains. However, in the same way as LOC, the health information systems were the ones that presented the largest number of classes, followed by restaurant, accounting, and e-commerce. Therefore, it is possible to assume that there exists a uniform variation among the domains characteristics, which can be reflected in the other software metrics and their thresholds.

Table 2. Statistics of Number of Classes per Domain

	Min	Max	Mean	Std. Dev.
Accounting	2	101	24,6	27,9
E-Commerce	2	55	15,9	15,7
Health	9	133	32,7	34,4
Restaurant	5	62	29,7	17,0

4. RESULTS AND ANALYSIS

To answer our RQ1 and RQ2 research questions, we independently analyzed whether and how the derived thresholds vary in terms of size (Section 4.1), inheritance (Section 4.2) and other metrics (Section 4.3). We obtained the thresholds in conformance with Vale's Method by using the TDTool [19].

4.1 Thresholds for Size Metrics

Table 3 presents the thresholds derived for the size metrics: Lines of Code (LOC), Weighted Methods per Class (WMC), Number of Attributes (NOA), and Number of Methods (NOM). The first two

columns present the metrics name and the Vale's Method thresholds distribution (3%, 15%, 90%, and 95%). The others columns show the respective derived thresholds for each domain. For instance, 90% of classes in the accounting domain have no more than 121 lines of code.

Table 3. Thresholds for Size Metrics

		Account	ECom	Health	Restaurant
	3%	3	4	3	3
LOC	**15%**	4	4	5	5
	90%	121	164	231	172
	95%	147	264	290	238
	3%	0	0	0	0
WMC	**15%**	1	0	0	1
	90%	21	27	24	24
	95%	31	50	30	30
	3%	0	0	0	0
NOA	**15%**	0	0	0	0
	90%	9	9	23	9
	95%	15	11	30	15
	3%	0	0	0	0
NOM	**15%**	0	0	0	0
	90%	14	10	12	12
	95%	22	17	16	16

The results, as presented in Table 3, show no relevant difference across domains for size metrics in the lower-bound thresholds (i.e., 3% and 15%). This result is somehow expected since many classes of these systems are small. For instance, at least 15% of the classes have no more than five lines of code in all domains. One reason for this large number of small classes might be that some systems are under development. Therefore, their classes are empty or with just stub code.

On the other hand, we can observe a substantial variation on the 95% upper-bound thresholds. The classes in health information systems have highest thresholds for LOC and NOA than information systems from other domains. For example, comparing with the accounting information systems, the largest classes in health systems have about twice more lines of code (147 vs. 290) and attributes (15 vs. 30). Therefore, these results suggest that the largest classes in health systems are more complex than the largest one in other domains, such as accounting and restaurant. We can also observe in Table 3 that LOC and NOA are two metrics that might benefits from domain-specific metrics, while NOM seems to not vary largely across domains.

Indeed, accounting seems to contain a small number of classes with a large number of lines of code. That is, 95% of classes in this domain have no more than 147 lines of code. Despite of this low threshold value for LOC, large classes in the accounting domain have many methods compared to the other domains. Table 3 shows that for both 90% and 95% thresholds, accounting has the highest values for NOM (14 and 22 methods per class, respectively) among the analyzed domains.

4.2 Thresholds for Inheritance Metrics

Table 4 presents the derived thresholds for two inheritance metrics, named Depth of Inheritance Tree (DIT) and Number of Children (NOC). The results reveal no considerable difference across domains considering the lower-bound thresholds (3% and 15%). Therefore, in addition to be small, at least 15% of classes in all analyzed domains do not use inheritance relationships.

Table 4. Thresholds for Complexity and Inheritance Metrics

		Account	ECom	Health	Restaurant
	3%	1	1	1	1
DIT	**15%**	1	1	1	1
	90%	8	4	13	8
	95%	10	4	21	12
	3%	0	0	0	0
NOC	**15%**	0	0	0	0
	90%	1	1	0	1
	95%	2	4	1	1

It is worth to note that the e-commerce and health systems have opposite results considering inheritance metrics (DIT and NOC). The health systems have deeper inheritance trees and a low number of children, while classes in e-commerce systems have many subclasses with flat inheritance tree. Accounting and restaurant information systems have very similar thresholds for DIT and NOC. Therefore, we can conclude that domain-specific thresholds might corroborate to more precise analysis of DIT and NOC only in some domains, such as e-commerce and health systems.

4.3 Thresholds for Coupling, Cohesion and Complexity Metrics

Table 5 presents the derived thresholds for three metrics: Coupling between Objects (CBO), Lack of Cohesion of Methods (LCOM), and McCabe Cyclomatic Complexity (McCabe). Considering the CBO metric, we can observe that the accounting domain has the lowest thresholds for both 95% and 90% percentiles. Also, the e-commerce and health domains present the highest thresholds for the 95% percentile which corroborates with our claim that health is a more complex domain. In the same way, as we can observe in Table 5, accounting and health are the two domains with the more discrepant values for LCOM. That is, they have more non-cohesive classes than the other domains. On the other hand, the McCabe complexity metric does not vary largely across domains. Therefore, CBO and LCOM, in contrast to McCabe, seem to be metrics that could benefits from domain specific thresholds. In general, considering the LCOM and McCabe metrics, we can hypothesize that the restaurant domain has, in general, more cohesive and simple classes than the other domains, which illustrate the need for specific thresholds for classes across domains.

Table 5. Thresholds for Coupling, Cohesion, and Complexity

		Account	ECom	Health	Restaurant
CBO	**3%**	0	0	0	0
	15%	0	0	1	0
	90%	6	10	11	9
	95%	8	14	14	10
LCOM	**3%**	0	0	0	0
	15%	0	0	0	1
	90%	55	28	36	35
	95%	207	71	72	63
McCabe	**3%**	0	0	0	0
	15%	1	0	0	1
	90%	7	12	8	7
	95%	11	20	10	11

5. ANALYSIS OF THRESHOLDS

After visualizing the calculated results using the Vale method, in this section some figures will help to better understand the significance of the obtained values for three metrics, CBO, DIT LOC. To conduct this visualization, the following figures show the values obtained for the metrics and their percentage of representativeness, for each domain and also for the combined domains. In Figure 1 the domains are represented by colors, as in the next figures. It is worth mentioning that the analysis is performed on the classes present in the systems within a domain, and there may be equal values, which in the graph would occupy the same space.

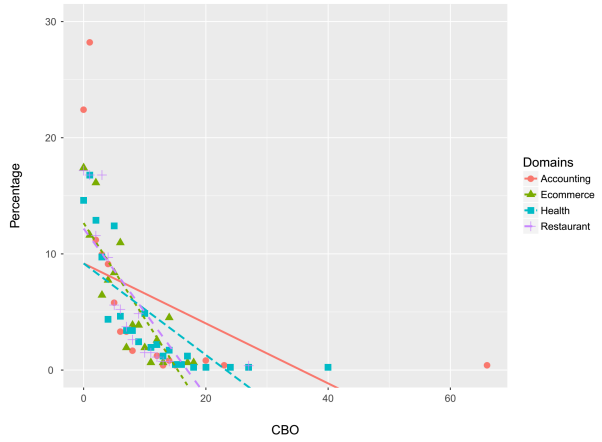

Figure 1. Thresholds for Coupling between Objects

In view of the previous figure, we note that the trend line is the need for different values of thresholds, being a unique value far from ideal, as represented by the orange color in accounting. An interesting point is that the lines point to a similar distribution in the accounting and ecommerce domains, but very different for health and restaurant.

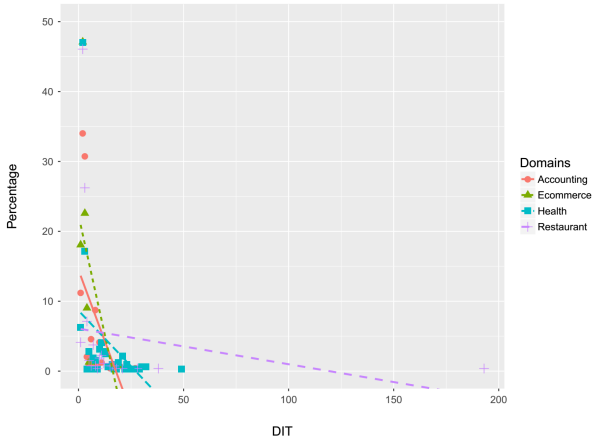

Figure 2. Thresholds for Depth Inheritance Tree

In Figure 2 it is possible to see the existence of a cluster of points in the initial region of the graph, showing that the inheritance trees are concentrated in smaller numbers, but also a different distribution in restaurant.

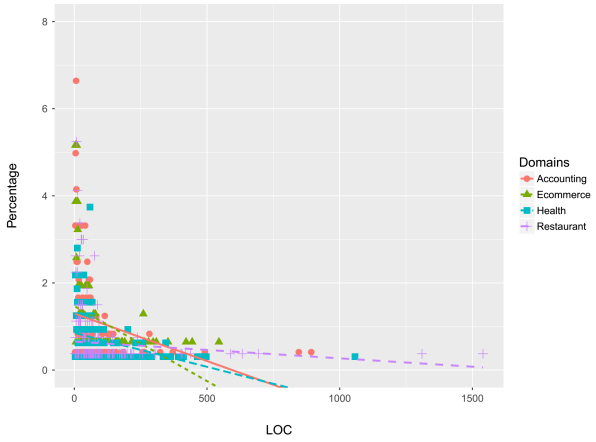

Figure 3. Thresholds for Lines of Code

For Figure 3, what stands out is the distribution of the points showing that there are many classes with a smaller size of lines of code, but the distribution helps to visualize that the distributions for each domain follow different, which justifies the different representative thresholds presented In the previous tables.

Considering the presented figures it is observed that when Vale method is applied on software systems domains the value of thresholds obtained are more accurate, varying for different metrics and systems modalities, being more appropriate for uses in other contexts, as in the search for Anomalies in codes, answering the third research question.

6. ANALYSIS OF OTHER BENCHMARKS

The previous section analyzed thresholds taking into account four benchmarks of systems that we mined from GitHub; one for each selected domain. In this section, we analyzed and compared our results with two other benchmarks from the literature, namely Qualitas Corpus [15] and Software Product Lines [16] (SPL Benchmark, for short). Section 5.1 analyzes thresholds for Qualitas Corpus, while Section 5.2 discusses the SPL Benchmark.

6.1 Comparison with Thresholds for Qualitas Corpus

In previous work [18], Vale's Method has been applied to the Qualitas Corpus benchmark. Since in this paper we used the same method to derive domain-specific thresholds, it is expected a comparison between ours and Vale's results [18]. The Qualitas Corpus benchmark is composed of industry-strengths information systems. It has more than a hundred systems and most of these systems are larger and more complex than the ones we mined from GitHub (used in Section 4). Similar to our benchmarks, all systems in Qualitas Corpus were developed in Java programming language.

Data in this section rely on the 20101126 release of Qualitas Corpus, composed by 106 open source Java software systems. For each system, the corpus presents a set of 21 software metrics. However, three systems, namely Eclipse 3.7.1, JRE 1.6.0, and Netbeans 7.3, do not have all metrics computed. Vale [18] then derived thresholds for a subset of seven metrics for 103 systems, but only four of these metrics (LOC, WMC, DIT, and NOC) are the same in this study.

Table 6 shows the obtained threshold values for these four metrics in Qualitas Corpus. For example, for LOC the derived thresholds are 3, 11, 308, and 510 for 3%, 5%, 90%, and 95%, respectively. Taking into account the inheritance metrics (DIT and NOC), we could not see significant difference between thresholds for specific domains and for general systems. That is, thresholds in Tables 4 and 5 for DIT and NOC are similar.

On the other hand, we observe that domain-specific thresholds for size metrics (Tables 3) are lower when compared with general thresholds (Table 5). With respect to LOC and WMC, thresholds for Qualitas Corpus are always higher than for any of our four benchmarks. For instance, in the 95% label, the highest domain-specific threshold is 290 LOC (health), against 510 LOC for Qualitas Corpus. A possible explanation is that, in general, systems in Qualitas Corpus are larger than systems in our benchmarks. Therefore, our conclusion is that, in addition to domain, thresholds also vary depending on the size of the systems in the benchmark.

Table 6. Thresholds of Qualitas Corpus

	LOC	*WMC*	*DIT*	*NOC*
3%	3	1	1	0
15%	11	2	1	0
90%	308	42	4	1
95%	510	70	5	2

6.2 Comparison with Thresholds for Software Product Lines

This section presents and discusses thresholds derived for a benchmark of 33 Software Product Lines (SPL Benchmark) [16]. An SPL is a configurable set of systems that shares a common, managed set of features in a particular market segment [13]. Features can be defined as modules of an application with consistent, well-defined, independent, and combinable functions [2]. The SPL Benchmark was built in previous work [2] because SPLs have been increasingly adopted in software industry to support coarse-grained reuse of software assets. This benchmark only includes software product lines developed using feature-oriented programming (FOP). In this study, we only considered software product lines implemented in AHEAD [4] and Feature House [3] because these programming languages extends Java.

Table 7 shows the threshold values for two metrics (LOC and WMC) for the SPL Benchmark. We focus only on these two metrics because they are the same in both studies, Vale et al. [16] and ours. Data in Table 3 shows that domain-specific thresholds for size metrics are usually higher than thresholds for software product lines (Table 7). For instance, considering the 95% percentage, the highest values are 139 and 32 for LOC and WMC, respectively, derived from the SPL Benchmark. For domain-specific thresholds, however, the lowest values are 147 for LOC (accounting) and 30 for WMC (health and restaurant). That is, the lowest domain-specific thresholds are higher than thresholds for software product lines, considering the 95% upper limit. This result confirms that thresholds depend on the used implementation technology (OOP vs. FOP) and programming languages (Java vs. AHEAD and FeatureHouse).

Table 7. Thresholds of Software Product Lines Benchmark

Benchmark	*%*	*LOC*	*WMC*
	3%	3	1
1	***15%***	5	2
	90%	78	18
	95%	139	32

7. RELATED WORK

Alves et. al. [1] proposed a threshold derivation methodology designed according to three requirements. First, it should respect the statistical properties of the metric, such as scale and distribution. Second, it should be based on data analysis from a representative set of systems (benchmark). Third, it should be repeatable, transparent and straightforward to execute. The authors applied the proposed method to a benchmark of 100 object-oriented information systems (C# and Java), both proprietary and open-source, to derive thresholds for metrics included in the SIG maintainability model (Unit Complexity, Unit Size, Module Inward Coupling, Module Interface Size). As a result, they observe that the derived thresholds are representative of the selected quantiles: low risk (between 0 – 70%), moderate risk (70 – 80%), high risk (80 – 90%) and very-high risk (> 90%).

Ferreira et. al. [6] presents the results of a study based on a collection of open-source software programs written in Java. The study aimed to establish thresholds for a set of metrics (LCOM, DIT, coupling factor, afferent couplings, number of public methods, and number of public fields). Based on the most commonly values, they were able to derive general thresholds for object-oriented software metrics, and thresholds by size and type (tool, library and framework). As Ferreira et. al. [6] did not find relevant difference among thresholds, they conclude that the general thresholds can be applied to object-oriented software in general.

Oliveira et. al. [12] propose the concept of relative thresholds for evaluating metrics data following heavy-tailed distributions. The proposed thresholds are relative because they assume that metric thresholds should be followed by most source code entities, but that it is also natural to have a number of entities in the "long-tail" that do not follow the defined limits. They report a study of

applying this method to the Qualitas Corpus with 106 systems. Based on the results, they argue that the proposed thresholds express a balance between real and idealized design practices. The authors also report the results of a study conducted to validate the method that extracts relative metric thresholds from benchmark data. They used this method to extract thresholds from a benchmark of 79 Pharo/Smalltalk software systems, which were validated with five experts and 25 developers. The results indicate that good quality software systems respect metric thresholds, while noncompliant ones are not largely viewed as requiring more effort to be maintained.

Silva et. al. [14] conducted an industry multi-project study to evaluate the reusability of detection strategies in a critical domain. They assessed the degree of accurate reuse of anomalies detection strategies based on the judgment of domain specialists. The study revealed that even though the reuse of strategies in a specific domain should be encouraged, their accuracy is still limited when holistically applied to all the modules of a program. However, the accuracy and reuse were both significantly improved when the metrics, thresholds and logical operators were tailored to each recurring concern of the domain.

8. THREATS TO VALIDITY

The focus on this work was to obtain thresholds for metrics of classes contained in selected domains, in order to compare and verify the consistency of the resulting values. Throughout the process, some concerns with validity have emerged. The main concerns that threaten the validity of this work are presented and discussed below.

Internal Validity. We identified the following threats to the construct validity: selected domains and key word search strings. We argue that the selected domains (accounting, e-commerce, health, and restaurant) are representative, given that they are well-defined in terms of a diversity of recurrent requirements (e.g., user and product management). Therefore, we believe that differences in implementation might reflect in valid varying thresholds among systems of distinctive domains. Another threat is the reliance on the key word search string for selecting the initial set of systems. We cannot ensure that the GitHub search facilities return all relevant systems of each domains. However, we could observe that the search process was able to return systems that we consider as relevant to our research questions (high starred).

Construction and Conclusion Validity. Threats to the validity also reside on how we have interpreted and implemented the software metrics. From the perspective of the application of the results, different interpretations of the software metrics represent a threat to the conclusion validity of the study. To avoid this problem, the tool-supported method [17] [19] to derive threshold was used to derive the thresholds. It makes the derivation process easy and repeatable.

External Validity. The major risk here is related to the limitation on selected systems. First, it is not possible to ensure that they reflect the best samples of the recurrent practice. To reduce this risk, we proceed by selecting systems from GitHub based on the ranking of starred systems. As mentioned, in GitHub, stars are a meaningful measure for repository popularity, and they may support the selection of relevant and high-quality systems for study. We also excluded systems with less than 300 lines of code (LOC) because we considered them simple toy examples. Second, the sample size might be itself another threat to the validity of the study. We have selected forty systems from different domains. However, this decision allowed us to obtain more consistent results that could be interpreted in this specific context. Nevertheless, additional replications are necessary to determine if our findings can be generalized to other domains and systems.

9. CONCLUSIONS AND FUTURE WORK

This paper presented an empirical study to investigate the hypothesis that thresholds vary among systems of different domains, since these systems have different degrees of complexity and size. To perform this study, it was necessary to collect metrics of the systems contained in the domains, and using tools. Thresholds were calculated and then compared (Section 4 and 5).

To evaluate our hypothesis, we relied on a benchmark composed by 40 object-oriented Java systems of four domains: accounting, e-commerce, health and restaurants. We applied a set of eight well-known metrics to each system, namely Lines of Code (LOC), Number of Attributes (NOA), Number of Methods (NOM), Weighted Method per Class (WMC), Lack of Cohesion in Methods (LCOM), McCabe Cyclomatic Complexity (McCabe), Depth of Inheritance Tree (DIT) and Number of Children (NOC). After aggregating metrics per domain, we derived thresholds for each metric by using a Vale's Method and its supporting tool called TDTool.

We finally compared the derived thresholds of each metric among the four domains. The results indicate that lower-bound thresholds (e.g., 15% smaller classes) usually do not significantly vary across domains. However, for all analyzed metrics, upper-bound thresholds (i.e., 90% and 95%) are visible different in some domains. In addition to obtaining the results, they were also used in a comparison with Qualitas Corpus and SPL-Benchmark. As a result, we observed that our derived thresholds seems to be more equalized to the reality of each domain (health, e-commerce, restaurant and account) than the general thresholds derived from the others two benchmarks.

For future work, others could use the presented data to compare with data obtained for a number of systems per domain. Following this idea, it would be possible to increase the number of domains and verify the variation behavior of the values. In addition, future work may apply a varied group of techniques to validate the thresholds for smaller groups within the same domain and to compare with other domains of systems. These further replications would verify the representativeness and reliability of the calculated values. This study was carried out on systems collected from GitHub, so a case study could be elaborated to compare with thresholds obtained from systems collected from companies. In addition to the highlights, it would be possible to use other metrics to analyze and compare with other studies and to be able to verify the validity of the thresholds found in the industry context.

10. ACKNOWLEDGMENTS

This work was partially supported by CAPES, CNPq (grant 424340/2016-0), and FAPEMIG (grant PPM-00382-14).

11. REFERENCES

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

[2] S. Apel, D. Batory, C. Kästner, G. Saake. "Feature-Oriented Software Product Lines". Springer, 2013.

[3] S. Apel, C. Kästner, C. Lengauer. “FeatureHouse: Language-Independent, Automated Software Composition”. In: Proceedings of the 31st International Conference on Software Engineering (ICSE), pp 221–231, 2009.

[4] D. Batory, J. Sarvela, A. Rauschmayer. “Scaling Step-Wise Refinement”. In Proceedings of the 25th International Conference on Software Engineering (ICSE), pp. 187–197, 2003.

[5] S. Chidamber, C. Kemerer. “A Metrics Suite for Object Oriented Design”. IEEE Trans. Software Engineering, 20(6), 476–493, 1994

[6] K. Ferreira, M. Bigonha, R. Bigonha, L. Mendes, H. Almeida. “Identifying Thresholds for Object-Oriented Software Metrics”. Journal of Systems and Software, 85(2), pp. 244–257, 2012.

[7] M. Fowler. “Refactoring: Improving the Design of Existing Code”. Pearson Education, 1999.

[8] M. Lanza, R. Marinescu. Object-Oriented Metrics in Practice. Springer, 2007.

[9] M. Lorenz, J. Kidd. Object-Oriented Software Metrics. Prentice Hall, 1994.

[10] R. Marinescu. “Detection Strategies: Metrics-based Rules for Detecting Design Flaw”s. In Proceedings of the 20th International Conference on Software Maintenance (ICSM), pp. 350–359, 2004.

[11] T. McCabe. “A Complexity Measure”. IEEE Transactions on Software Engineering, 2(4), pp. 308–320, 1976.

[12] P. Oliveira, M. Valente, 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.

[13] K. Pohl, G. Böckle, F. van der Linden. “Software Product Line Engineering”. Springer Science & Business Media, 2005.

[14] A. Silva, A Garcia, E. Cirilo, C. Lucena. “Reuse of Domain-Sensitive Strategies for Detecting Code Anomalies: A Multi-Case Study”. In Proceedings of the Brazilian Symposium on Software Engineering (SBES), 2013.

[15] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, J. Noble. “Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies”. In Proceedings of the Asia Pacific Software Engineering Conference (APSEC), pp336–345, 2010.

[16] G. Vale, D. Albuquerque, E. Figueiredo, A. Garcia. “Defining Metric Thresholds for Software Product Lines: A Comparative Study”. In proc. of Int’l Software Product Line Conf. (SPLC), 176–185, 2015.

[17] G. Vale, E. Figueiredo. A Method to Derive Metric Thresholds for Software Product Lines. In Proceedings of the 29th Brazilian Symposium on Software Engineering (SBES), pp. 110–119, 2015.

[18] G. Vale. “A Benchmark-based Method to Derive Thresholds”. MSc Dissertation, Federal University of Minas Gerais, 2016.

[19] L. Veado, G. Vale, E. Fernandes, and E. Figueiredo. “TDTool: Threshold Derivation Tool”. In proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE), Tools Session, 2016.