An Empirical Evaluation of Distribution-based Thresholds for Internal Software Measures

Luigi Lavazza
Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria
Varese, Italy
luigi.lavazza@uninsubria.it

Sandro Morasca
Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria
Varese, Italy
sandro.morasca@uninsubria.it

ABSTRACT

Background Setting thresholds is important for the practical use of internal software measures, so software modules can be classified as having either acceptable or unacceptable quality, and software practitioners can take appropriate quality improvement actions. Quite a few methods have been proposed for setting thresholds and several of them are based on the distribution of an internal measure's values (and, possibly, other internal measures), without any explicit relationship with any external software quality of interest.

Objective In this paper, we empirically investigate the consequences of defining thresholds on internal measures without taking into account the external measures that quantify qualities of practical interest. We focus on fault-proneness as the specific quality of practical interest.

Method We analyzed datasets from the PROMISE repository. First, we computed the thresholds of code measures according to three distribution-based methods. Then, we derived statistically significant models of fault-proneness that use internal measures as independent variables. We then evaluated the indications provided by the distribution-based thresholds when used along with the fault-proneness models.

Results Some methods for defining distribution-based thresholds requires that code measures be normally distributed. However, we found that this is hardly ever the case with the PROMISE datasets, making that entire class of methods inapplicable. We adapted these methods for non-normal distributions and obtained thresholds that appear reasonable, but are characterized by a large variation in the fault-proneness risk level they entail. Given a dataset, the thresholds for different internal measures—when used as independent variables of statistically significant models—provide fairly different values of fault-proneness. This is quite dangerous for practitioners, since they get thresholds that are presented as equally important, but practically can correspond to very different levels of user-perceivable quality. For other distribution-based methods, we found that the proposed thresholds are practically useless, as many modules with values of internal measures deemed acceptable according to the thresholds actually have high fault-proneness. Also, the accuracy of all of these methods appears to be lower than the accuracy obtained by simply estimating modules at random.

Conclusions Our results indicate that distribution-based thresholds appear to be unreliable in providing sensible indications about the quality of software modules. Practitioners should instead use different kinds of threshold-setting methods, such as the ones that take into account data about the presence of faults in software modules, in addition to the values of internal software measures.

PROMISE 2016, September 09 2016, Ciudad Real, Spain

DOI: http://dx.doi.org/10.1145/2972958.2972965

CCS Concepts

•Software and its engineering → Empirical software validation;

Keywords

Thresholds; Software measures; Internal measures; External measures; Fault-proneness; Faultiness; Software quality

1. INTRODUCTION

Software practitioners can greatly benefit from accurate estimates of the faultiness of software modules[1] along the entire development and maintenance lifecycle, to allocate resources in an effective way. By "faultiness," we denote the presence of at least one fault in a module. Modules that are estimated faulty need to be subject to close monitoring and, possibly, modifications. For instance, more Verification & Validation (V&V) effort should be allocated to modules that are estimated faulty after the coding phase than to those estimated non-faulty. As another example, if, at design time, a module is predicted to turn out to be faulty when it is coded, then the module's design characteristics (e.g., its interface) or the entire design of the system should be checked and possibly modified.

Estimating whether a module is faulty requires extracting and using information about the module's characteristics, i.e., extracting and using values of measures taken on the module. A large number of measures for so-called "internal" software attributes [11] (e.g., size, structural complexity, cohesion, coupling) have been defined. We call them "internal" measures and use X to denote any one of them.

[1]We use the term "module" to denote any piece of software, e.g., a subsystem, a component, a class, a method.

The sheer knowledge of the value of X for a software module, however, is not very useful when evaluating the module's faultiness. Practitioners need to have indications about which values of X should be considered acceptable and which unacceptable, so they can take adequate actions on modules whenever needed. The values of X are usually partitioned into acceptable and unacceptable ones by setting a threshold value for X such that, for instance, the values of X that exceed the threshold are considered unacceptable, and the others are considered acceptable.

However, the *crux* of the problem is the selection of the specific threshold values to be set on internal measures, to effectively partition modules into good and bad quality ones. Using inadequate thresholds results in inadequate module development. Thresholds that are overly risk-averse may lead to estimating too many modules as bad. Thus, there will be unnecessary refactorings, V&V, etc., and, as a consequence, a waste of resources. Thresholds that are too risk-prone may lead to estimating too few modules as bad. Thus, not all necessary refactorings and V&V will be carried out, and, in the end, several modules will be buggy and difficult to maintain, which may result in added costs later on.

Even though the need for reliable thresholds has long been recognized (see the related work reviewed in Section 8), there are no universally accepted indications for even the most well-known measures. Thus, practitioners do not have clear guidelines about which threshold values to use.

In addition, a number of approaches to setting thresholds have been defined in the literature (see Section 8). Some are based on experience (we call them experience-based approaches), some on the values of the internal variables alone (we call them distribution-based approaches), and some (we call them model-based approaches) on the existence of models that estimate fault-proneness, defined here as the probability that a module contains a fault. These methods obviously differ in the kind of information they use.

Using different methods may lead to setting different threshold values for the same measure X, so practitioners need to have indications about which methods they should or should not use. This calls for the evaluation of threshold-setting methods, to evaluate their effectiveness and characteristics.

The overall goal of this paper is to empirically evaluate some distribution-based methods. Specifically, this paper addresses the following two Research Questions.

RQ1 Do thresholds set by a distribution-based method on different internal measures entail consistent levels of fault-proneness?

RQ2 How accurate are the estimates obtained using distribution-based thresholds?

If the answer to RQ1 is negative, then modules may be over the thresholds of variables X_1 and X_2 (hence, considered at risk) when actually just the value of X_1 entails truly high risk, while the value of X_2 entails a moderate level of risk. Thus, threshold "violations" would be almost meaningless in this case, as their meaning is related to the internal measure chosen and not to the actual level of risk.

RQ2 instead investigates how effective distribution-based methods are anyway. To this end, we carried out an empirical study with datasets from the PROMISE repository [1], which contains data on a set of real-life applications.

We would like to clarify upfront that it is not the goal of this paper to propose a new threshold definition method. As our Research Questions show, we evaluate the performance of a few existing distribution-based methods.

The remainder of this paper is organized as follows. Section 2 introduces the distribution-based methods that we empirically evaluate in the paper. The empirical study is described in Section 3. Its results are summarized in Section 4 and discussed in Section 5. Section 6 gives suggestions for defining and using thresholds on internal measures. Threats to validity are discussed in Section 7, while the analysis of the related literature is in Section 8. The conclusions and an outline for future work are in Section 9.

2. DISTRIBUTION-BASED THRESHOLDS

We here describe the distribution-based methods for defining thresholds that are evaluated in this paper. Note that we have only taken into account methods that can be fully automated, in that they do not require that any parameter be set by a researcher or practitioner.

2.1 Use of Mean and Standard Deviation

Some of the methods proposed in the literature suggest that the threshold for a given measure X should be based exclusively on its distribution, that is, on the frequency of the values x. More specifically, both Erni and Lewerentz [10] and Lanza and Marinescu [19] describe methods to define thresholds based on the mean value and the standard deviation of measures. As an applicability precondition, both proposals assume that X follows a normal distribution.

According to [10], the interval $[\mu-\sigma, \mu+\sigma]$ is regarded as the central range of "normal" values for X, where μ and σ are the average and the standard deviation of the distribution of X, respectively. The proposal of [19] takes $\mu-\sigma$ as the "low" threshold, $\mu+\sigma$ as the "high" threshold, and $1.5(\mu+\sigma)$ as the "very high" threshold.

2.2 Use of Weighted Benchmark Data

Alves et al. use data from multiple different software systems to derive thresholds that are expected to "(i) bring out the metric's variability between systems and (ii) help focus on a reasonable percentage of the source code volume" [2]. According to the authors, the method should also be resilient against outliers in measure values or system size.

The method is organized in six steps, as follows. 1. *Measure extraction.* The values of X and of LOC (or another size measure to be used for weighting) are computed for the modules of a "benchmark," i.e., a set of software systems.

2. *Weight ratio calculation.* Each module i of a given system j is given a weight $w_{i,j}$ equal to the fraction of its LOC $loc_{i,j}$ with respect to the total LOC in project j (i.e., $LOC_j = \sum_{i \in j} loc_{i,j}$):

$$w_{i,j} = \frac{loc_{i,j}}{LOC_j}$$

3. *Entity aggregation.* This is the computation of the distribution of weights for the values of X in the modules of each system j. Specifically, the weight $w_j(x)$ associated with value $X = x$ in system j is obtained by adding the weights $w_{i,j}$ of those modules of j in which $X = x$

$$w_j(x) = \sum_{\forall i \text{ such that } X_i = x} w_{i,j} = \sum_{\forall i \text{ such that } X_i = x} \frac{loc_{i,j}}{LOC_j}$$

4. *System aggregation.* The weights $w_j(x)$ are first nor-

malized for the number of systems, i.e., they are divided by the number of systems n. Then, an overall weight $w(x)$ is obtained for value x of X by summing these normalized weights over all systems in the benchmark

$$w(x) = \sum_{\forall j \in benchmark} \frac{w_j(x)}{n}$$

5. *Weight ratio aggregation.* The distribution of weights $w(x)$ is obtained. For instance, Figure 1 shows the distributions of weights for *CBO* in the benchmark we used for the empirical study.

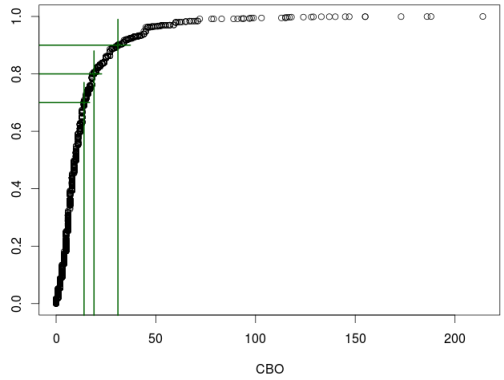

Figure 1: Distribution of weight for CBO.

6. *Thresholds derivation.* A threshold $\bar{x}$ on X is derived based on the choice of a fraction of the distribution of weights $w(x)$. By setting a fraction F on the distribution of weights $w(x)$, we define $\bar{x}$ as the minimal value such that

$$\sum_{X \in min(X)..\bar{x}} w(x) \geq F \quad (1)$$

Thus, the modules with $x \leq \bar{x}$ are considered "good" and the other "bad."

Three thresholds are then somewhat arbitrarily selected and used, to split modules into to four categories: low risk (between 0 and 70%), moderate risk (between 70 and 80%), high risk (between 80 and 90%), and very-high risk (greater than 90%). As an example, in Figure 1 the values of *CBO* corresponding $F = 70\%$, 80% and 90% are highlighted.

A consequence of this method is that modules in smaller systems may have a disproportionate influence on the final weight distribution. For instance, suppose that the benchmark is composed of two systems A (a "large" one) and B (a "small" one), where the largest module in B is smaller than the smallest module in A. Suppose that the largest module in B contains 60% of the *LOC* in B. Then, after the normalization and aggregation in Step 5, the value of X associated with that module has a weight equal to 30%, which might be much higher than the weight of any of the modules of A, which are larger than that module. Thus, it is not necessarily true that a fraction F is representative of the same proportion of *LOC*. For instance, it is not in general true that with $F = 90\%$, we are estimating as "good" a set of modules that contains 90% of the *LOC*.

2.3 Use of Quantiles

Vale et al. [37] introduce a threshold definition method for measures for Software Product Lines, though it can be used on any kind of software system. Like in [2], the method is organized in several steps and is based on a "benchmark." At any rate, the threshold for a variable X is ultimately derived based on quantiles of the distribution of the values of X computed by taking into account the values of X associated with the modules of the entire benchmark.

Vale et al. introduce the following ranges delimited by thresholds: very low values (between 0-3%), low values (3-15%), moderate values (15-90%), high values (90-95%), very high values (95-100%). The approach is then compared to the approach of (but without considering the high-level risk threshold) identification of God Classes and Lazy Classes, two well-known code smells. The paper shows that the method introduced gives better results than [19].

2.4 Other Techniques

A variation of the methods described in Section 2.1 was proposed by Shatnawi in [33]: thresholds are derived after a log transformation of measures. Given a set D of values for measure X, the threshold for X is computed as $e^{mean(log(D))+sd(log(D))}$. The method relies on the idea that points that are most distant from the average are more likely to suffer from problems like faultiness, difficult maintainability, etc. Thus, besides the technicality of the log transformation, the method proposed by Shatnawi in [33] is extremely similar to those described in Section 2.1. The thresholds obtained after log transformation are not evaluated here because of space limits.

The three approaches described in Sections 2.1-2.3 are essentially different from other approaches, which we review in Section 8. Here we would like to mention that these approaches, like others, define only one threshold value that does not take into account industrially-relevant quantities like the costs associated with classifying modules incorrectly. An approach like the one based on Receiver Operating Characteristic (ROC) [36] curves may be more flexible and provide indications related to different relative costs of incorrectly estimating too many modules as faulty or non-faulty. Section 8 concisely reviews methods based on ROC.

3. THE EMPIRICAL EVALUATION

To evaluate the methods described in Section 2, we applied them to real-life data in datasets from the PROMISE repository [1] that contain fault data, namely, the datasets provided by Marian Jureckzo, available at `http:// open-science.us/repo/defect/`. Each dataset contains data on the modules of a software application: whether the module is faulty, and the values of the internal measures described in the web page managed by Spinellis and Jureczko (`http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/metric.html`).

The empirical study involved a sequence of steps.

Step 1. Threshold computation.
Thresholds are computed using the methods described in Sections 2.1-2.3.

Step 2. Fault-proneness model derivation.
Models of fault-proneness (i.e., the probability that a module is faulty) vs. code measures were derived, using Binary Logistic Regression (BLR) [16]. We have chosen BLR since it has been widely used in Empirical Software Engineering, with good results, as described in a comprehensive systematic literature review [15].

Only models that are statistically significant (at the usual 0.05 statistical significance level) according to the sign, Hos-

mer, and likelihood ratio tests are used in this paper.

BLR models fault-proneness as follows

$$fp(X) = \frac{e^{logit(X)}}{1 + e^{logit(X)}} \quad (2)$$

We use the most common kind of BLR, in which $logit(X)$ is linear: $logit(X) = c_0 + c_1 X$. For instance, the model of fault-proneness as a function of *CBO* for one of the PROMISE datasets is shown in Figure 3. Given the *CBO* value of a module, this model estimates the probability that the given module is faulty (i.e., the module's fault-proneness).

Step 3. Faultiness estimation with the thresholds. Given X such that there exists a statistically significant model $fp_X(X)$, given a threshold T_X for X, all modules with $x > T_X$ are estimated faulty, while the others are estimated non-faulty. Repeat the computation for all X and all thresholds.

Step 4. Evaluation of fault-proneness with the thresholds. Given a threshold setting method, compute the values of $fp_X(T_X)$ for all variables X by using T_X computed in the same way. For instance, by using the method of [10], compute T_X for all X as the $\mu + \sigma$ quantile for all measures X. Check whether the values of $fp_X(T_X)$ obtained with different measures X are concentrated, i.e., all similarly obtained thresholds provide consistent indications of risk. If, instead, the values of $fp_X(T_X)$ are dispersed, i.e., different measures are associated with different levels of risk, then "risky" may mean very different things depending on the measure. This step addresses Research Question RQ1.

Step 5. Evaluation of faultiness estimates. In this paper, to quantify the accuracy of the faultiness estimates, we use Precision, Recall and F-measure (FM) [38], because they are commonly used in the literature, and the F-measure has probably become the *de facto* overall accuracy indicator for estimated faultiness models. In addition to using the thresholds described in Section 2, we use a "reference" approach, in which we first set a fault-proneness threshold, equal to the number of faulty modules in the dataset divided by the number of modules in the dataset, and we then derive a threshold T_X for X. We use this threshold because it represents the probability of picking a faulty module at random. The accuracy of any method for estimating whether a module is faulty needs to be greater than the accuracy obtained without the method, by simply selecting modules at random. We compare all of the accuracy results obtained. This step addresses Research Question RQ2.

4. OUTCOMES OF THE EMPIRICAL STUDY

We here present the results obtained via our empirical study. First, we show the detailed results for the berek project from the PROMISE repository. Then, we summarize the results obtained for the other datasets (for space reasons, we cannot give the detail results for all datasets).

4.1 On the Distribution of Code Measures

Before applying the method described in Section 2.1, we checked the applicability condition that measures are normally distributed with the Shapiro-Wilk statistical test.

Unfortunately, it turned out that the distribution of the vast majority of measures is very far from normal: small values of the measures are much more frequent than large values. For instance, Figure 2 shows the distribution in project berek of *CBO*, which clearly is not normally distributed.

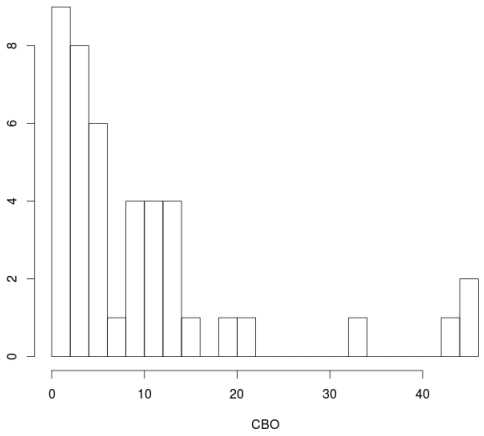

Figure 2: The frequency of CBO values for berek.

Thus, methods based on the mean and standard deviation of measures are usually not applicable. On top of that, we often obtained $\mu - \sigma < 0$ for measures that cannot be negative by definition, so the lower threshold cannot be used.

To overcome this problem, we adapted the method of [10] and [19] as follows. In a normal distribution, $x = \mu - \sigma$ and $x = \mu + \sigma$ approximately correspond to the 16% and 84% quantiles of the distribution, respectively, so, setting those two thresholds amounts to setting thresholds at the 16% and 84% quantiles. In a non-normal distribution, $\mu - \sigma$ and $\mu + \sigma$ may not be as representative, but the 16% and 84% quantiles preserve their meanings. Therefore, we used as the lower and higher thresholds the value of x corresponding to the 16% and 84% quantiles, respectively.

We call this extension the "Adapted Distribution-Based" (ADB) method. To be sure, our goal is *not* to introduce a new distribution-based approach to setting thresholds: we introduce ADB with the only purpose of making the method of [10, 19] applicable. Otherwise, it would be hardly ever possible to use this method in the PROMISE repository, or other repositories, e.g., the Java Qualitas Corpus [9].

4.2 Distribution-Based Thresholds

In the berek dataset, only the distribution of avgCC (the class average of methods' cyclomatic complexity) is distributed normally. Hence, we applied the ADB method as a replacement of those of Section 2.1. Table 1 summarizes the results we obtained. Subcolumn 'T' in columns "Tmin" and "Tmax" contains the thresholds on X corresponding to the 16% and 84% quantiles, respectively, and in column "Tref" the X threshold corresponding to a fault-proneness value given by the proportion of faulty modules. This proportion is often indicated as AP/n, where AP is the number of actual positives (i.e., faulty) modules, and n is the total number of modules. We discuss the values in the other subcolumns (those with the "fp" header) in Section 4.6. The thresholds concerning *CAM* (Cohesion Among Methods) deserve an explanation. For all the other measures, increasing values correspond to lower expected quality. On the contrary, greater values of *CAM* are associated with better quality. For this reason, we reversed *Tmin* and *Tmax* for *CAM* in Table 1. In practice, the thresholds in each column of Table 1 go together: for instance, a class having threshold 2

Table 1: Fault-proneness of thresholds obtained via the ADB method for berek.

	Tmin		Tmax		Tref	
	T	fp	T	fp	T	fp
WMC	4.00	0.10	36.56	0.92	15.69	0.37
DIT	1.00	0.24	5.00	0.65	2.39	0.37
CBO	2.00	0.02	14.56	0.91	8.91	0.37
RFC	12.16	0.01	89.56	1.00	41.74	0.37
CA	1.00	0.05	10.00	0.88	5.34	0.37
CE	0.72	0.08	9.56	0.81	5.14	0.37
LOC	62.2	0.01	874.3	1.00	404.6	0.37
CAM	0.64	0.04	0.19	0.79	0.38	0.37
AMC	6.27	0.04	46.21	0.82	28.81	0.37
maxCC	1.00	0.14	23.56	0.90	8.28	0.37
avgCC	0.67	0.14	2.43	0.64	1.63	0.37

for *CBO* and 62 for *LOC* also has threshold 0.64 for *CAM*.

The values found appear to be quite extreme. Consider for instance *WMC* (note that in the datasets *WMC* is the number of methods, since the applied weight is one): only classes with fewer than 4 methods are below Tmin. Similarly, only classes with *CBO*=1 are below *Tmin*. In practice, only classes that are very small, very simple, and very loosely coupled are below the values of *Tmin*. Similarly, *Tmax* tends to select classes that are fairly large, complex, and coupled.

4.3 Thresholds Based on Weighted Benchmark Data

By applying to the PROMISE datasets the threshold derivation method by Alves et al. described in Section 2.2, we obtained the thresholds given in Table 2, where the 'T' subcolumns in the "T90," "T80," and "T70" columns are the thresholds corresponding to the 90%, 80% and 70% quantiles, as indicated in [2]. We discuss the values in the other subcolumns (those with the "fp" header) in Section 4.6.

Table 2: Fault-proneness of thresholds computed via the method by Alves et al. [2] for berek.

	T90		T80		T70		Tref	
measure	T	fp	T	fp	T	fp	T	fp
WMC	74	1.00	43	0.97	30	0.82	15.69	0.37
DIT	6	0.75	4	0.55	3	0.44	2.39	0.37
CBO	31	1.00	19	0.99	14	0.88	8.91	0.37
RFC	183	1.00	118	1.00	77	0.99	41.74	0.37
CA	15	0.99	8	0.72	6	0.46	5.34	0.37
CE	26	1.00	16	0.99	11	0.89	5.14	0.37
LOC	3211	1.00	1815	1.00	1084	1.00	404.6	0.37
CAM	0.137	0.86	0.179	0.81	0.201	0.77	0.377	0.37
AMC	91.25	1.00	62.43	0.97	47.39	0.84	28.81	0.37
maxCC	26	0.94	13	0.58	8	0.36	8.28	0.37
avgCC	3.57	0.90	2.43	0.64	1.94	0.48	1.63	0.37

4.4 Thresholds Based on Benchmark Quantiles

By applying to the PROMISE datasets the method by Vale and Figueiredo (see Section 2.3), we obtained the thresholds given in Table 3, where the subcolumns 'T' in the "T95," "T90," "T15," and "T3" columns are the thresholds corresponding to the 95%, 90%, 15% and 3% quantiles, as indicated in [37]. We discuss the values in the other subcolumns (those with the "fp" header) in Section 4.6.

Table 3: Fault-proneness of thresholds obtained via the method by Vale and Figueiredo [37] for berek.

	T95		T90		T15		T3		Tref	
Measure	T	fp	T	fp	T	fp	T	fp	T	fp
WMC	37	0.93	25	0.69	3	0.09	1	0.07	15.7	0.37
DIT	4	0.55	4	0.55	1	0.24	1	0.24	2.4	0.37
CBO	31	1.00	20	0.99	2	0.02	0	0.01	8.9	0.37
RFC	84	0.99	57	0.82	5	0.00	2	0.00	41.7	0.37
CA	19	1.00	10	0.88	0	0.03	0	0.03	5.3	0.37
CE	18	0.99	13	0.95	0	0.06	0	0.06	5.1	0.37
LOC	1151	1.00	677	0.95	15	0.00	2	0.00	405	0.37
CAM	0.17	0.81	0.21	0.76	0.7	0.02	1	0.00	0.38	0.37
AMC	79.1	0.99	49.7	0.87	3.25	0.03	0	0.02	28.8	0.37
maxCC	13	0.58	8	0.36	1	0.14	0	0.12	8.28	0.37
avgCC	3	0.80	2.25	0.58	0.64	0.13	0	0.06	1.63	0.37

4.5 Fault-proneness Models

By analyzing the berek dataset, we found a set of statistically significant fault-proneness models, for measures *WMC*, *DIT*, *CBO*, *RFC*, *CA*, *CE*, *LOC*, *CAM*, *AMC*, *maxCC*, *avgCC*. We do not report them explicitly for space reasons.

4.6 Fault-proneness Levels Corresponding to Distribution-based Thresholds

Tables 1, 2, and 3 show the values of fault-proneness in the subcolumns "fp" corresponding to the thresholds obtained via the considered distribution-based methods. In these tables, columns "Tref" show the fault-proneness corresponding to *AP/n*, which is 0.37 for berek.

4.7 Accuracy

Finally, we evaluated the accuracy of estimates based on the thresholds obtained via the considered distribution-based methods. The evaluation was carried out as described in Section 3, step 5. The computed accuracy indicators are in Tables 4, 5 and 6. In these tables, the accuracy indicators of the thresholds corresponding to *AP/n* are also given, in columns "Tref." The tables show that the *FM* values obtained with Tref are better than those obtained with the other thresholds 85 times out of 99.

Table 4: Accuracy of fault-proneness estimates (using ADB thresholds) for berek.

	Tmin			Tmax			Tref		
Measure	FM	Prec.	Rec.	FM	Prec.	Rec.	FM	Prec.	Rec.
WMC	0.63	0.46	1.00	0.52	0.86	0.38	0.80	0.86	0.75
DIT	0.44	0.40	0.50	0.30	0.75	0.19	0.57	0.67	0.50
CBO	0.64	0.47	1.00	0.61	1.00	0.44	0.86	0.79	0.94
RFC	0.62	0.44	1.00	0.61	1.00	0.44	0.91	0.88	0.94
CA	0.64	0.48	0.94	0.55	1.00	0.38	0.84	0.87	0.81
CE	0.58	0.42	0.94	0.61	1.00	0.44	0.73	0.79	0.69
LOC	0.62	0.44	1.00	0.61	1.00	0.44	0.91	0.88	0.94
CAM	0.62	0.44	1.00	0.43	0.71	0.31	0.69	0.63	0.75
AMC	0.62	0.44	1.00	0.61	1.00	0.44	0.71	0.67	0.75
maxCC	0.67	0.52	0.94	0.61	1.00	0.44	0.69	0.90	0.56
avgCC	0.60	0.44	0.94	0.61	1.00	0.44	0.57	0.53	0.63

4.8 Summary of Results for All Analyzed PROMISE datasets

The analysis described above was applied to all the datasets of the PROMISE repository provided by Marian Jureckzo. Because of space limits, we cannot describe the results in detail. Table 7 summarizes the results obtained with the ADB method. For each dataset, the following data are given:

Table 5: Accuracy of fault-proneness estimates (using the thresholds by Alves et al.) for berek.

	T90			T80			T70			Tref		
measure	FM	Prec.	Rec.	FM	Prec.	Rec.	FM	Prec.	Rec.	FM	Prec.	Rec.
WMC	0.12	1.00	0.06	0.48	1.00	0.31	0.58	0.88	0.44	0.80	0.86	0.75
DIT	0.00	1.00	0.00	0.62	0.80	0.50	0.62	0.80	0.50	0.57	0.67	0.50
CBO	0.40	1.00	0.25	0.48	1.00	0.31	0.61	1.00	0.44	0.86	0.79	0.94
RFC	0.40	1.00	0.25	0.48	1.00	0.31	0.77	1.00	0.63	0.91	0.88	0.94
CA	0.40	1.00	0.25	0.74	0.91	0.63	0.80	0.86	0.75	0.84	0.87	0.81
CE	0.40	1.00	0.25	0.40	1.00	0.25	0.55	1.00	0.38	0.73	0.79	0.69
LOC	0.40	1.00	0.25	0.40	1.00	0.25	0.55	1.00	0.38	0.91	0.88	0.94
CAM	0.00	1.00	0.00	0.30	0.75	0.19	0.62	0.80	0.50	0.69	0.63	0.75
AMC	0.00	1.00	0.00	0.40	1.00	0.25	0.55	1.00	0.38	0.71	0.67	0.75
maxCC	0.55	1.00	0.38	0.69	0.90	0.56	0.69	0.90	0.56	0.69	0.90	0.56
avgCC	0.00	1.00	0.00	0.61	1.00	0.44	0.59	0.56	0.63	0.57	0.53	0.63

Table 6: Accuracy of fault-proneness estimates (using thresholds by Vale and Figueiredo) for berek.

	T95			T90			T15			T3			Tref		
measure	FM	Prec.	Rec.	FM	Prec.	Rec.	FM	Prec.	Rec.	FM	Prec.	Rec.	FM	Prec.	Rec.
WMC	0.52	0.86	0.38	0.64	0.89	0.50	0.60	0.43	1.00	0.55	0.38	1.00	0.80	0.86	0.75
DIT	0.62	0.80	0.50	0.62	0.80	0.50	0.44	0.40	0.50	0.44	0.40	0.50	0.57	0.67	0.50
CBO	0.40	1.00	0.25	0.48	1.00	0.31	0.64	0.47	1.00	0.55	0.38	1.00	0.86	0.79	0.94
RFC	0.67	1.00	0.50	0.81	1.00	0.69	0.56	0.39	1.00	0.55	0.38	1.00	0.91	0.88	0.94
CA	0.40	1.00	0.25	0.55	1.00	0.38	0.56	0.39	1.00	0.56	0.39	1.00	0.84	0.87	0.81
CE	0.40	1.00	0.25	0.48	1.00	0.31	0.58	0.42	0.94	0.58	0.42	0.94	0.73	0.79	0.69
LOC	0.55	1.00	0.38	0.86	1.00	0.75	0.55	0.38	1.00	0.54	0.37	1.00	0.91	0.88	0.94
CAM	0.11	0.50	0.06	0.67	0.82	0.56	0.57	0.40	1.00	0.57	0.40	1.00	0.69	0.63	0.75
AMC	0.00	1.00	0.00	0.55	1.00	0.38	0.57	0.40	1.00	0.56	0.39	1.00	0.71	0.67	0.75
maxCC	0.69	0.90	0.56	0.69	0.90	0.56	0.67	0.52	0.94	0.58	0.41	1.00	0.69	0.90	0.56
avgCC	0.12	1.00	0.06	0.64	0.75	0.56	0.59	0.42	1.00	0.58	0.41	1.00	0.57	0.53	0.63

- The name of the dataset;
- The number of statistically significant models found (datasets with one or zero models are omitted).
- The difference between the minimum and maximum fault-proneness corresponding to low thresholds (i.e., thresholds corresponding to the 16% quantile).
- The difference between the minimum and maximum fault-proneness corresponding to high thresholds (i.e., thresholds corresponding to the 84% quantile).
- The proportion of faulty modules, which is also the fault-proneness value adopted for the reference model.

The differences in fault-proneness are often quite large. For instance, for the szybkafucha project, though only two statistically significant fault-proneness models were found, the thresholds for their independent variables (namely, *CE* and *NPM*) correspond to fault-proneness values that differ by approximately 13%. In other words, even though the two thresholds were derived with the same criteria and quantile, they correspond to quite different risk levels.

Table 8 gives, for each dataset, the best and worst F-measure obtained using the threshold in the fault-pronenss models. It can be seen that the high threshold generally provide very bad accuracy (remember that values of F-measure less than AP/n are even worse than those obtained by wild guessing). We can also observe that *Tmin* provides a level of accuracy that is sometimes better than that provided by the reference model, and sometimes worse. In practice, setting the threshold at percentile 16% provides random levels of accuracy at random levels of risk.

Table 7: Difference in fault-proneness for thresholds computed with the ADB method.

		max fp difference		
Dataset	#mod.	Tmin	Tmx	AP/n
ivy-1.1	5	4.8%	11.8%	56.8%
lucene-2.2	3	8.4%	6.6%	58.3%
lucene-2.4	6	6.0%	11.5%	59.7%
nieruchomosci	3	8.7%	23.5%	40.0%
pdftranslator	10	23.6%	25.3%	45.5%
poi-1.5	5	7.3%	11.3%	60.6%
poi-2.5	3	16.7%	4.5%	65.4%
poi-3.0	6	4.8%	15.7%	64.2%
szybkafucha	2	12.8%	13.8%	56.0%
velocity-1.4	4	13.3%	5.1%	74.9%
xerces-1.4	10	30.9%	14.6%	74.3%
xalan-2.5	7	4.0%	4.6%	48.5%
zuzel	9	14.9%	21.5%	44.8%
berek	11	23.5%	35.6%	37.2%
kalkulator	3	4.8%	25.3%	22.2%
wspomaganiepi	2	14.4%	2.6%	66.7%

The results obtained with the thresholds proposed by Alves et al., and Vale and Figueiredo—not shown for space limits—are even worse than those obtained with the ADB method.

It appears that, with distribution-based thresholds, there is always the risk that the threshold's values are unrelated to external qualities that may be of interest for developers.

Table 8: Best and worst F-measures obtained for each dataset with the ADB thresholds.

	Worst FM values			Best FM values		
Dataset	Tmin	Tmax	Tref	Tmin	Tmax	Tref
ivy-1.1	0.74	0.31	0.63	0.77	0.42	0.73
lucene-2.2	0.71	0.27	0.56	0.74	0.35	0.72
lucene-2.4	0.73	0.27	0.58	0.76	0.37	0.71
nieruchomosci	0.65	0.46	0.74	0.65	0.57	0.84
pdftranslator	0.64	0.40	0.56	0.77	0.57	0.79
poi-1.5	0.73	0.13	0.62	0.79	0.33	0.73
poi-2.5	0.80	0.14	0.67	0.82	0.26	0.80
poi-3.0	0.79	0.16	0.64	0.81	0.35	0.82
szybkafucha	0.76	0.00	0.77	0.85	0.25	0.80
velocity-1.4	0.76	0.00	0.74	0.89	0.28	0.89
xerces-1.4	0.64	0.02	0.58	0.96	0.34	0.83
xalan-2.5	0.50	0.24	0.40	0.65	0.34	0.49
zuzel	0.65	0.00	0.72	0.72	0.56	0.80
berek	0.44	0.30	0.57	0.67	0.61	0.91
kalkulator	0.43	0.00	0.53	0.56	0.73	0.67
wspomaganiepi	0.81	0.40	0.82	0.92	0.40	0.92

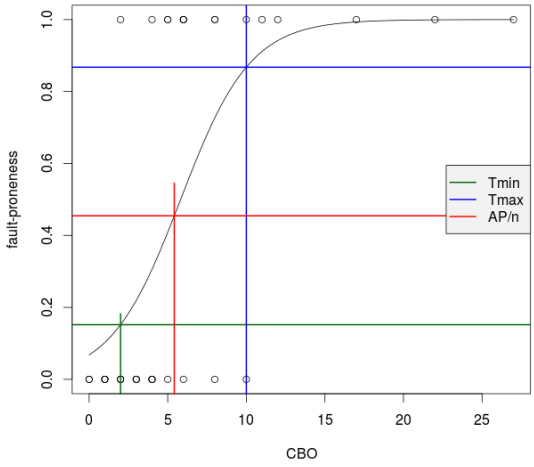

Figure 3: BLR model of fault-proneness vs. CBO for pdftranslator, with ADB thresholds.

Consider Figure 3, which shows the model of fault-proneness vs. *CBO* for the pdftranslator dataset. The lower threshold *Tmin* is actually too low: *all* modules with *CBO* < *Tmin* are non-faulty. The higher threshold *Tmax* is too high: *all* modules with *CBO* > *Tmax* are faulty. Instead, the reference thresholds is more representative: it separates the values of *CBO* of *mostly* non-faulty modules from those of *mostly* faulty modules.

Consider now Figure 4, which shows the model of fault-proneness vs. *WMC* for the zuzel dataset, along with the thresholds computed with the method by Alves et al. [2]. All of the thresholds are too high: all the modules with *WMC* greater than the T70 threshold are faulty. Actually, in the zuzel dataset no module has *WMC* greater than the T90 threshold: the threshold is ineffective, in this case. Being the thresholds computed for the whole benchmark, this situation occurs often. Similar situations occur with the thresholds computed using the method by Vale and Figueiredo.

Note that the cases illustrated in the figures above do not necessarily occur for every dataset and internal measure. However they happen frequently, and the user has no means

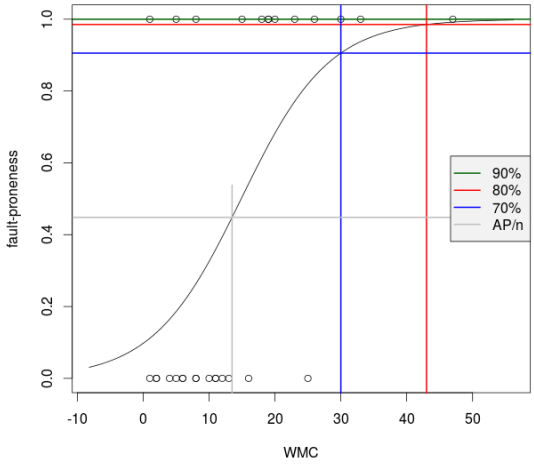

Figure 4: BLR model of fault-proneness vs. WMC for zuzel, with thresholds by Alves et al.

to know whether the thresholds he or she derived are possibly "too low" or "too high" in the sense described above.

5. DISCUSSION OF RESULTS

As for RQ1, the results show that each of the distribution-based threshold-setting methods has a considerable variability in the associated levels of risk with different internal measures. For instance, suppose that the manager of the berek project decides to use the method by Vale and Figueiredo, and considers worth of additional V&V those modules whose measures' values are "moderate," "high," or "very high" [37], i.e., those whose measures are above the threshold corresponding to the 15% percentile. As shown in Table 3, this criterion leads to selecting for additional V&V all the modules whose fault-proneness is above 0.5% when *RFC* or *LOC* are considered: a very—probably excessively—risk-averse approach. Instead, with *DIT*, only modules with fault-proneness greater than 24% are selected for additional V&V: a much less risk-averse approach! Note that similar differences between the minimum and maximum fault-proneness among internal measure thresholds occur for all the methods and quantiles considered in this paper.

So, when using a distribution-based method to set thresholds, one may estimate as unacceptable modules that are associated with very different levels of risk of being faulty and, conversely, as acceptable modules that are associated with very different levels of risk. Thus, the indications about the level of risk provided by these thresholds are unreliable.

As for RQ2, the results (see Tables 4, 5 and 6) show that the accuracy of the distribution-based threshold-setting methods is generally quite low. In particular, the achieved accuracy appears constantly lower (often much lower) than the accuracy achieved by the "reference" threshold, which is set simply by assuming that the proportion of faulty modules is equal to the proportion observed in the training dataset.

6. SUGGESTIONS FOR DERIVING AND USING THRESHOLDS

Thresholds defined based on the distribution of an internal measure alone (and, possibly, the value of other internal

measures) are built without using any data on any specific practically relevant quality of interest.

We used faultiness in our paper and we defined fault-proneness as the probability that a module contains at least a fault. Note that defining the measure for an external quality [11] as a model that estimates a probability is the theoretically sound way of quantifying the external quality [18, 23]. Suppose therefore that we measure maintainability as the probability of carrying out some modification activity in a specified amount of time, which can be quantified via models that take internal measures as their independent variables. Or, we measure usability as the probability of completing some operation in a specified amount of time, which, again, we can quantify via models.

The thresholds for the internal measures defined via distribution-based methods would remain the same, no matter the specific quality (fault-proneness, maintainability, usability, etc.) practitioners are interested in. There is no *a priori* reason to believe that the threshold for an internal measure should be the same regardless of the quality of interest.

Even worse, distribution-based methods define a threshold for X that should be applied even when no relationship exists between X and fault-proneness, i.e., when there is no sufficient evidence that X has any impact on fault-proneness. As a consequence, there is no actual reason to set a threshold on X in the first place, let alone the specific value that should be given to it.

Thus, we advocate the use of model-based threshold-setting approaches, which explicitly use a quantitative link between an internal measure and a quality of interest (the presence of faults in a module in our case). Specifically, a fault-proneness threshold is set first as the maximum acceptable fault-proneness value, based on the specific goals and needs of the practitioners. Based on this threshold and the existence of a fault-proneness model, one *derives* a threshold on X, which is therefore linked to the practitioners' specific goals and needs. For instance, Figure 4 shows that, once one sets value 0.45 as the fault-proneness threshold (i.e., the maximum acceptable fault-proneness value), the threshold value 13 is derived for *CBO*.

This approach is customizable, since different practitioners may very well define different fault-proneness thresholds. Also, different thresholds may be set for different qualities. For instance, if we are interested in maintainability, we may want to define a minimum acceptable probability that a module undergoes some modification in a specified time interval. Suppose that we have a statistically significant model that shows that maintainability is a decreasing function of *CBO*. Based on the maintainability threshold and the model we can established a maximum acceptable value for *CBO*, which may be very different from the threshold value that may be set on *CBO* when fault-proneness is concerned.

Note that model-based approaches have been proposed and used in Empirical Software Engineering (see Section 8).

7. THREATS TO VALIDITY

Internal Validity. We modified the approach of [10, 19] by using 16% and 84% as the lower and upper thresholds. As explained, we could not use the $\mu - \sigma$ and $\mu + \sigma$ thresholds because the distributions of the internal measures cannot be considered normal, as required in [10, 19]. However, we also explained that 16% and 84% are the quantiles corresponding to, respectively, $\mu - \sigma$ and $\mu + \sigma$ in a normal distribution. Thus, it is as if the methods of [10, 19] actually fixed two quantiles when defining the lower and upper thresholds, like we do.

Also, all of the BLR models described in the empirical study are statistically significant, so the computations of the fault-proneness values corresponding to the threshold values on internal measures are based on statistical evidence.

External Validity. We have used a limited number of datasets, and this may affect the external validity of our study. However, we chose datasets related to applications of different nature and with different numbers of datapoints. The results we obtained seem to be consistent on all of them.

Construct Validity. A construct validity threat comes from the use of *Precision*, *Recall*, and FM as accuracy indicators. We selected these indicators to allow for comparison between our results and those of the literature, because these indicators are the most widely used.

8. RELATED WORK

Given the importance of fault-proneness in Software Engineering, a large number of studies have addressed the definition of fault-proneness models based on internal measures. They are too many to review here. Comprehensive systematic literature reviews on fault prediction approaches and their performance can be found in [4] and [15], respectively.

Here, we are interested in threshold definition, so we here review papers in the literature addressing this problem.

Several papers define thresholds on an internal variable X based only on its distribution, i.e., they do not take into account a response variable, like the fact that a module may or may not be faulty. Thus, it is not clear if they have any practical usefulness, since it is not known whether a relationship between X and fault-proneness truly exists and, so, whether it makes sense to set thresholds on X. In Section 2, we have already described the four distribution-based methods [10, 19, 2, 37] that we used in our empirical study.

The study of [27] proposes the definition of a threshold based on a so-called "relative threshold," based on two parameters, Min and $Tail$, whose values are based on real and idealized design rules, respectively. Min is the minimum proportion of values of the variable that correspond to how design rules are implemented in practice, so its computation will be based on a corpus of existing data. $Tail$ is the percentage of values of the variable at which the "fat tail" of the distribution of the variable starts.

In paper [12], thresholds for a number of OO measures have been provided based on three ranges, i.e., "good," "regular," and "bad." The distributions of the variables examined in the software applications considered follows a power law. The "good" range is where most or a significant part of the distribution is concentrated, as it contains the typical applications, even though, as the authors state, not necessarily the best ones. The region containing a small portion of the distribution is identified as the "bad" one.

The approach for computing distribution-based thresholds for different contexts proposed in [14] returns a quantile as the threshold.

Thresholds for the measures of [19] are defined in [3] based on the data in the Qualitas Corpus [35] to find code smells.

In [13], based on their previous work [12], the authors introduce a catalog of thresholds for 17 object-oriented measures by using the Qualitas Corpus data [35], and apply these thresholds to assess bad smells in a software system.

Oliveira et al. [26] and Veado et al. [39] describe two tools computing the thresholds of the approaches of, respectively, [27], and [2, 12, 27, 37].

Other approaches are based on experience. These proposals too do not provide a clear, explicit, mathematical link between an internal measure and fault-proneness. Like with distribution-based approaches, it is not possible to assess if a relationship between X and fault-proneness really exists and therefore if it makes sense to set thresholds.

For instance, when introducing Cyclomatic Complexity (CC) as a control-flow complexity measure, McCabe [21] suggests 10 as an upper limit of CC. Later, the same threshold was still suggested [40] in the context of testing. In the same report, the authors state that 60 is typically seen as a threshold for LOC for a routine, basically because 60 is the number of lines of code that fit in one page. Other experience-based proposals are defined in [6, 8, 28].

Model-based approaches are based on the relationship between X and fault-proneness. Schneidewind [30] introduces an approach to setting thresholds for any number of variables at the same time. Later papers [31, 17] extend and somewhat modify the original approach.

Several model-based proposals also take into account the approaches used in epidemiological studies when estimating which patients need to be closely monitored or treated. Following [5], a model can be used in two ways: a threshold can be set as the value of the maximum acceptable fault-proneness, or one can set a threshold on the slope of the model linking X and fault-proneness, i.e., a maximum acceptable variation rate of fault-proneness with X.

Shatnawi's proposal [32] uses BLR models to investigate how to set thresholds for the CK metrics suite [7], by setting a threshold on the probability first and then deriving a threshold for X from it. Four different probability thresholds (0.06, 0.065, 0.075, and 0.1) are arbitrarily selected. The so-called g-measure, which is the harmonic mean of *Recall* and the ratio of true negatives to actual negatives, is used to evaluate the accuracy of the estimated classification. The results favor threshold 0.065.

The H-index has been extended [24] to identify, in a set of software modules, the top fault-prone ones (which should be considered for further V&V) in much the same way as the H-index identifies a researcher's top publications (the ones that should be considered, as the others do not contribute to the H-index). An empirical analysis shows promising results in terms of *Recall* and *FM*.

An approach to setting thresholds to identify "early symptoms" of possible problems is defined in [25] based on the slope of a fault-proneness model, instead of the actual values provided by the model. The approach appears to be effective at identifying those values of a variable X that may signal the presence of possible problems early during development.

Another approach uses an optimistic-pessimistic approach to setting thresholds and identifying so-called "grey zones" [20]. Specifically, given a training set and a test set, an optimistic and a pessimistic models are built. The optimistic (resp., pessimistic) model is built by combining the training and test sets in a new dataset, assuming that all modules in the test set are non-faulty (resp., faulty), and building a fault-proneness model based on this combined dataset. Optimistic and pessimistic fault-proneness thresholds are set and used on the optimistic and pessimistic models, respectively. Modules that are estimated non-faulty (resp., faulty) by both models and associated thresholds are estimated non-faulty (resp., faulty). Modules for which the estimates of the optimistic and pessimistic models with associated thresholds conflict are not classified, and belong to the "grey zone," so they will need to be further examined to be estimated as either faulty or non-faulty. The Naive Bayes classifier (i.e., the other effective technique according to [15]) is also used to estimate if an OO class is positive [36]. Specifically, thresholds are defined based on ROC curves with *Recall* on the y-axis and the ratio of the false positives to the actual negatives on the x-axis. The optimal threshold is the one corresponding to the point in the ROC curve at minimum distance from the ideal point $(0, 1)$, which represents perfect classification. Other studies [22, 29, 34] also use ROC curves to derive fault-proneness thresholds for OO measures.

9. CONCLUSIONS AND FUTURE WORK

In this paper, we empirically investigated methods that define thresholds on an internal measure X based only on the distribution of X alone, or possibly, on additional information from other internal variables. From a conceptual point of view, these methods do not take into account any externally tangible, practically relevant quality, e.g., the presence of faults in software modules. Our study investigated if, despite this conceptual flaw, these methods can still provide thresholds associated with (1) consistent level of risk and/or (2) high accuracy when applied on real-life data.

Our empirical analysis, carried out on datasets from the PROMISE [1] repository, shows that the distribution-based methods we evaluated perform quite poorly in both respects. Thus, we advocate the need of using threshold-setting methods like the model-based ones, since they are based on information about the quality of interest, in addition to the values of the internal measures.

Future work will include

- defining new model-based threshold-setting methods;
- carrying out additional empirical validations for evalating model-based threshold-setting methods;
- assessing results by means of other accuracy indicators, in addition to Precision, Recall, and the F-measure.

10. ACKNOWLEDGMENTS

Parts of this work are supported by the FP7 Collaborative Project S-CASE (Grant Agreement No 610717), funded by the European Commission, and the "Fondo di ricerca d'Ateneo" funded by the Università degli Studi dell'Insubria.

11. REFERENCES

[1] The PROMISE repository of empirical software engineering data. http://openscience.us/repo, North Carolina State Univ., Computer Science Dept., 2015.

[2] T. L. Alves, C. Ypma, and J. Visser. Deriving metric thresholds from benchmark data. In *ICSM 2010*, 2010.

[3] F. Arcelli-Fontana, V. Ferme, M. Zanoni, and A. Yamashita. Automatic metric thresholds derivation for code smell detection. In *WETSoM 2015*, 2015.

[4] S. Beecham, T. Hall, D. Bowes, D. Gray, S. Counsell, and S. Black. A systematic review of fault prediction approaches used in software engineering. *Lero Technical Report Lero-TR-S20P1L0–2004*.

[5] R. Bender. Quantitative risk assessment in epidemiological studies investigating threshold effects. *Biometrical Journal*, 41(3):305–319, 1999.

[6] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer. Managerial use of metrics for object-oriented software: An exploratory analysis. *IEEE Trans. Software Eng.*, 24(8):629–639, 1998.

[7] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Software Eng.*, 20(6):476–493, 1994.

[8] D. M. Coleman, B. Lowther, and P. W. Oman. The application of software maintainability models in industrial software systems. *Journal of Systems and Software*, 29(1):3–16, 1995.

[9] G. Concas, M. Marchesi, S. Pinna, and N. Serra. Power-laws in a large object-oriented software system. *IEEE Trans. Software Eng.*, 33(10):687–708, 2007.

[10] K. Erni and C. Lewerentz. Applying design-metrics to object-oriented frameworks. In *METRICS 1996*, 1996.

[11] N. E. Fenton and J. M. Bieman. *Software Metrics: A Rigorous and Practical Approach, 3rd ed.* CRC Press. 2014.

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

[13] T. G. S. Filó, M. A. da Silva Bigonha, and K. A. M. Ferreira. A catalogue of thresholds for object-oriented software metrics. In *Int. Conf. on Advances and Trends in Soft. Eng.*, 2015.

[14] M. Foucault, M. Palyart, J. Falleri, and X. Blanc. Computing contextual metric thresholds. In *SAC 2014*, 2014.

[15] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Trans. on Software Eng.*, 38(6), 2012.

[16] D. W. Hosmer Jr and S. Lemeshow. *Applied logistic regression*. John Wiley & Sons, 2004.

[17] T. M. Khoshgoftaar. Improving usefulness of software quality classification models based on boolean discriminant functions. In *ISSRE 2002*, 2002.

[18] D. H. Krantz, R. D. Luce, P. Suppes, and A. Tversky. *Foundations of Measurement*, volume 1. Academic Press, San Diego, 1971.

[19] M. Lanza and R. Marinescu. *Object-Oriented Metrics in Practice - Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer, 2006.

[20] L. Lavazza and S. Morasca. Identifying thresholds for software faultiness via optimistic and pessimistic estimations. In *Accepted for publication at ESEM 2016*, 2016.

[21] T. J. McCabe. A complexity measure. *IEEE Trans. on Software Eng.*, (4), 1976.

[22] J. Mendling, L. Sánchez-González, F. García, and M. L. Rosa. Thresholds for error probability measures of business process models. *Journal of Systems and Software*, 85(5):1188–1197, 2012.

[23] S. Morasca. A probability-based approach for measuring external attributes of software artifacts. In *ESEM 2009*. IEEE Computer Society, 2009.

[24] S. Morasca. Classifying faulty modules with an extension of the H-index. In *ISSRE 2015*, 2015.

[25] S. Morasca and L. Lavazza. Slope-based fault-proneness thresholds for software engineering measures. In *EASE 2016*, 2016.

[26] P. Oliveira, F. P. Lima, M. T. Valente, and A. Serebrenik. Rttool: A tool for extracting relative thresholds for source code metrics. In *30th IEEE Int. Conf. on Software Maintenance and Evolution*, 2014.

[27] P. Oliveira, M. T. Valente, and F. P. Lima. Extracting relative thresholds for source code metrics. In *CSMR-WCRE 2014*, 2014.

[28] L. H. Rosenberg, R. Stapko, and A. Gallo. Risk-based object oriented testing. In *Proc. 24th Annual NASA-SEL Software Engineering Workshop*.

[29] L. Sánchez-González, F. García, F. Ruiz, and J. Mendling. A study of the effectiveness of two threshold definition techniques. In *EASE 2012*, 2012.

[30] N. F. Schneidewind. Software metrics model for integrating quality control and prediction. In *ISSRE 1997*, 1997.

[31] N. F. Schneidewind. Investigation of logistic regression as a discriminant of software quality. In *METRICS 2001*, pages 328–337, 2001.

[32] R. Shatnawi. A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems. *IEEE Trans. Software Eng.*, 36(2):216–225, 2010.

[33] R. Shatnawi. Deriving metrics thresholds using log transformation. *Journal of Software: Evolution and Process*, 27(2):95–113, 2015.

[34] R. Shatnawi, W. Li, J. Swain, and T. Newman. Finding software metrics threshold values using ROC curves. *Journal of Software Maintenance*, 22(1), 2010.

[35] E. D. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. The qualitas corpus: A curated collection of Java code for empirical studies. In *APSEC 2010*, 2010.

[36] A. Tosun and A. B. Bener. Reducing false alarms in software defect prediction by decision threshold optimization. In *ESEM 2009*, 2009.

[37] G. A. D. Vale and E. M. L. Figueiredo. A method to derive metric thresholds for software product lines. In *SBES 2015*, 2015.

[38] C. vanRijsbergen. *Information Retrieval*. Butterworths, 1979.

[39] L. Veado, G. Vale, E. Fernandes, and E. Figueiredo. Tdtool: threshold derivation tool. In *EASE 2016*. ACM, 2016.

[40] A. H. Watson and T. J. McCabe. Structured testing: A testing methodology using the cyclomatic complexity metric. NIST Special Publication 500-235, National Institute of Standards and Technology, 1996.