Contextual Operationalization of Metrics as Scores: Is My Metric Value Good?

Sebastian Hönel*, Morgan Ericsson, Welf Löwe, and Anna Wingkvist
Linnaeus University, Växjö, Sweden
{sebastian.honel, morgan.ericsson, welf.lowe, anna.wingkvist}@lnu.se
*corresponding author

***Abstract*—Software quality models aggregate metrics to indicate quality. Most metrics reflect counts derived from events or attributes that cannot directly be associated with quality. Worse, what constitutes a desirable value for a metric may vary across contexts. We demonstrate an approach to transforming arbitrary metrics into absolute quality scores by leveraging metrics captured from similar contexts. In contrast to metrics, scores represent freestanding quality properties that are also comparable. We provide a web-based tool for obtaining contextualized scores for metrics as obtained from one's software. Our results indicate that significant differences among various metrics and contexts exist. The suggested approach works with arbitrary contexts. Given sufficient contextual information, it allows for answering the question of whether a metric value is good/bad or common/extreme.**

Keywords*—*Software quality; Metrics; Scores; Software Domains;

I. Introduction

Metrics values are rarely of great utility to analysts and decision-makers, often because there are no apparent connections between what can be measured and the defined quality goals [1], [2]. Therefore, software quality models define an aggregation scheme for the underlying metrics. Together with other information, they can then be used to indicate quality [3]. This is required because most metrics do not indicate what an absolute desirable value might be. Even acceptable values may vary depending on the context. For example, in the context of computer games, functions with long bodies (*Method Lines of Code*; MLOC) might be acceptable, while they are less so in the context of unit test cases. Analysts often find themselves confronted with whether or not their metrics' values are good/bad, acceptable/alarming, or common/extreme. It turns out that defining an ideal value is difficult. Others have attempted to derive ideal values, e.g., from experience or surveys [4], benchmarks [5], or by just setting practical values [6]. We suggest that learning from sufficiently many metrics values observed in the relevant context(s) can be exploited to answer the question of what an ideal value is. By the same means, one can also gain a uniform understanding of how far off the actual values are.

We suggest transforming metrics into scores, generalizing and contextualizing [7]. A score always has a range of $[0, 1]$ and linear behavior, that is, a change of 0.1 always corresponds to an improvement or deterioration of 0.1, whether it is from, e.g., 0.1 to 0.2 or 0.8 to 0.9. This is not the case for metrics. We present a way to define uniformly distributed scores, considering that each metric value is not equally likely. For example, achieving lower and lower values for complexity metrics becomes exponentially more difficult. Moreover, we define the scores in a context-dependent way, considering that the same metric value is not equally likely in two different domains. For example, it might be more difficult to achieve lower values for complexity metrics in gaming than in testing.

For the empirical validation, we facilitate the "Qualitas.class corpus" of software metrics [8], [9] and a web application created for this work[1]. The corpus holds 23 types of pre-computed metrics for a total of 111 systems which are spread across eleven different domains. We study how metrics values are distributed among all the various contexts. The operationalization of a metric as a score lies in context-specific gathering of *usual* values (or deviations from ideal values). An appropriate context includes the most relevant or similar systems, such as computer games, middleware, or databases. Alternatively, it may also refer to, for example, earlier or derivative versions of the same system. If no previous context exists, a good starting point is perhaps to analyze open-source software in the desired domain. In this work, we equate the corpus' different domains with different contexts. The gathered values are used to approximate a probability distribution which allows for assessing how rare or common a value is. The cumulative distribution is then used to assign a score.

A. Notions

Metric. A measurement based on a well-defined standard, method, or calculation is called a metric. Metrics are often derived from counting certain software properties or occurrences of events [10]. For instance, afferent and efferent couplings (CA, CE) are the numbers of classes in other packages that depend upon classes within a package and that the classes in a package depend upon, respectively. Metrics can also be ratios. For instance, instability (RMI) is defined as CE $\div$ (CE $+$ CA). In this work, we use metrics, their names, and abbreviations as defined in [8] and [11].

Metrics can be described as random variables $\mathcal{M}$, i.e., as a numerical description of the measurement outcome as a statistical experiment. The probability distribution for a metric describes how the probabilities are distributed over its values. For counting metrics (discrete random variables), the probability distribution is defined by a probability function, mapping each metric value x to a probability. For real-valued metrics (continuous random variables), ratios are conveniently modeled as such. The probability distribution is defined by a probability *density* function. Its integral over any interval of

[1] Metrics As Scores, a web application. https://metrics-as-scores.ml/.

2693-9177/22/$31.00 ©2022 IEEE
DOI 10.1109/QRS57517.2022.00042

metric values provides the probability that the variable will take on a value within that interval. We denote the probability (density) function of $\mathcal{M}$ by $f_{\mathcal{M}}$ and drop the subscript if the metric is evident or does not matter.

Distance. Assume there exists an ideal value i for $\mathcal{M}$. Then for each metric value $x \in \mathcal{M}$, a function $D_i(x) = |\, x - i \,|$ denotes the distance of x from the ideal i. Distances too can be described as random variables $\mathcal{D}$, i.e., as a numerical description of the outcome of measurement and distance calculation as a statistical experiment. For example, McCabe's Cyclomatic Complexity (VG; [12]) has the lowest-possible and most-desirable value of $i = 1$. The domain of the corresponding random variable $\mathcal{D}_1^{\mathsf{VG}} = \{\mathbb{R}^+ \cup 0\}$, i.e., the distances for VG assume only values ≥ 0. Other distance functions D are possible, as well. However, in this paper, distances are used to order values of a random variable from good (small) to bad (large).

Score. Given an ordered random variable $\mathcal{X}$, e.g., a metric $\mathcal{M}$, or a distance $\mathcal{D}$, its cumulative distribution function (CDF) is $F_{\mathcal{X}} : \mathbb{R} \rightarrow [0, 1]$, with $F_{\mathcal{X}}(x) = \int_{-\infty}^{x} f_{\mathcal{X}}(x)\, dx$. The associated score function (sometimes also called "survival function") is the *complementary* CDF (CCDF), that is, $S_{\mathcal{X}}(x) = 1 - F_{\mathcal{X}}(x)$. Scores are therefore high for good values, e.g., low distances to the ideal, and low for bad values, e.g., high distances to the ideal.

B. Organization of the Paper

The remainder of this paper is structured as follows. In Section II, we give some background information on the problem, which should clarify our motivation for this work to the reader. The research design, i.e., the methodology used to conduct the studies, is then presented in Section III. Section IV is dedicated to the application "Metrics As Scores", which accompanies this paper. Results and discussions are found in Sections V and VI, respectively. Some threats to the validity of our study are examined in Section VII. The most relevant and related work is summarized and briefly discussed in Section VIII, before the paper is concluded with prospects for potential and already ongoing future work in Section IX.

II. Background

Most metrics are not directly useful for quality assessment since they individually only have a weak relation to the quality, i.e., they cannot be linked to quality properties directly [13], [2]. Therefore, some quality-related applications resort to using metrics as, e.g., fault indicators [14], [15] or as indicators for reliability and complexity [11]. This is partly because comparing and aggregating metrics in a mathematically sound way is difficult due to their different scales and distributions [16].

We argue that the limited use of metrics is also caused by the issue of what constitutes a good or a bad metric value. To mitigate this issue, the aforementioned quality-related applications often require fitting some regression or classification model. Others attempt to answer this question by setting thresholds. While some have an empirical approach, e.g., [17], [5], others search for practical values. For example, Hewlett Packard used to enforce a maximum value of 16 for complexity (VG), a value beyond which any modules needed to be re-designed [6].

However, none of these approaches paid great attention to the statistical differences of metrics in different software domains, especially their context-dependent distributions and ideal values, which is the main motivation of our work.

The originality of our work lies in suggesting non-parametric ways to derive domain-specific ideal values (if none is explicitly available) and transforming metrics into uniform scores that can be used for mathematically sound aggregation. Scores, unlike metrics, can easily be compared and aggregated.

Consider the metrics LCOM [11], assessing (lack of) method cohesion, and SIX [8] (specialization index), assessing the ratio of added, overridden, and inherited methods. For both, lower values are better and the lowest possible value is 0. Let's assume that both metrics are equally important (weighted) for our notion of quality. A change of -0.3 in either of the two metrics from one software version to another can be interpreted qualitatively as an improvement. However, as their scales and distributions may differ, as we will show even for different domains, we cannot compare this improvement, let away suggest the development team focus on cohesion or inheritance hierarchy refactoring.

We can, however, compare a change of $+0.3$ in either score because a unit change is considered equally good between these two and across all other scores. This also implies that not only can we answer the question "is my metric value good?", but also whether the software as a whole with all its scores is good. Furthermore, if we can compare scores, we can compare (same) aggregations thereof, meaning we can compare the total scores of two software applications. All of this allows for guiding software development toward some goal. If we had for every metric a corresponding score, the score can become a fitness function and be used to guide a search for optimal or near-optimal individuals in a search space of possible solutions much more directly [18].

A. Research Questions

Our work is supported by extensive empirical validation using the Qualitas.class corpus [8]. We pose four research questions to address the significance of using scores instead of metrics and the differences among metrics and domains in the corpus.

> **Research Question 1.** Are any of the metrics in the Qualitas.class corpus distributed uniformly?
> **Research Question 2.** Are there significant statistical differences for each metric across all domains?
> **Research Question 3.** Is each domain in its entirety (i.e., considering all metrics) distinguishable from the other domains?
> **Research Question 4.** What are good/bad or common/extreme scores for each domain of the Qualitas.class corpus?

III. Research Design

The methodology used in this paper was designed to study the potential differences across the various contexts of a dataset. In this work, the data are metrics, and the context is given by the various domains of the software systems of the Qualitas.class corpus. For details about the corpus itself, such as the metrics or domains, please refer to [8], [9], [11], [19], [12]. Mind that we use the metrics' abbreviations as used in [8]. The research design applies to other kinds of data, too, as the primary goal is to explore and understand the statistically significant differences in the data across contexts. This is based on the hypothesis that the same data (here metrics) value might be average or extreme in different contexts (here, software domains).

A. Calculating a Distance

Recall that the mapping that produces a distance does not need to be constant (see Section I-A). Let $D_i : \mathcal{X} \to \mathbb{R}$ be the distance from an ideal value $i \in \mathcal{X}$. We recommend D_i to meet the requirements of a statistical distance. These are:

- Non-negativity: Any distance is greater than or equal to zero, that is, $D_i(x) \geq 0 \; \forall \, x \in \mathcal{X}$. Especially, for the ideal value i, $D_i(i) = 0$.
- Identity of indiscernibles: The distance for two values x_1, x_2 from an ideal value i is identical if $x_1 = x_2$, that is, $D_i(x_1) = D_i(x_2)$.
- Symmetry (commutativity): $D_i(x) = D_x(i)$ (the distance between origin and destination is the same when interchanged).
- Triangle inequality: The sum of intermediate distances is greater than or equal to the largest distance: $D_i(x_2) \leq D_i(x_1) + D_{x_1}(x_2)$.

The previously suggested transformation using the absolute value, that is, $D_i(x) = |\, x - i \,|$, fulfills the above requirements. Again, other distance definitions fulfilling them are eligible, too.

B. Metrics With and Without Ideal Values

Most software-related metrics do not define what might be an (un-)desirable value. That means (a distribution derived from) observed values cannot be transformed into (a distribution of) distances, as no suitable ideal exists. Relating to the Qualitas.class corpus, we argue that such an ideal value can be derived (learned, approximated) from the metrics values for a domain. Given a distribution of observed domain values of some metric, we identify and provide a rationale for the following transformations leading to an approximation of an ideal.

1) Infimum and Supremum: Assuming the sample infimum, which is the lowest observed value, as ideal, i.e., $i = \inf \mathcal{X}$, then $D_i(\mathcal{X})$ does not reorder the random variable values but only translates its domain such that it starts with 0. This transform is still useful when an absolute ideal value is not available. Still, artifacts with lower metric values are of greater utility. Using the infimum attaches the notion of *lower is better* to a metric.

Likewise, using the sample supremum, which is the largest observed value, as ideal, i.e., $i = \sup \mathcal{X}$, has the same effect of translating the variable's domain. In addition, the order of distances is reversed, which is equal to the notion of *higher is better* for metrics' values.

2) The Median: The median is defined as the value that splits a probability distribution into a lower and higher half. Equally many values are to be found in either half. Using the (sample) median as an ideal value is useful in contexts where we want to check outlier metric values. Then the notion attached by the median is how far the metric lies in the direction of either extreme. Picking the median as the ideal value is reasonable when high scores reflect proximity to a value that is not considered extreme in either half and, therefore, fits in with previously observed values.

3) The Expectation: The expectation $\mathbb{E}\left[\mathcal{X}\right]$ of a Random Variable $\mathcal{X}$ with probability density function $f_{\mathcal{X}}$ is the weighted average of $\mathcal{X}$, with $f_{\mathcal{X}}(x)$ serving as the weight of each realization $x \in \mathcal{X}$. For discrete $\mathcal{X}$, $\mathbb{E}\left[\mathcal{X}\right] = \sum_{i=1}^{n} x_i \, f_{\mathcal{X}}(x_i)$. For continuous $\mathcal{X}$, $\mathbb{E}\left[\mathcal{X}\right] = \int_{-\infty}^{\infty} x \, f_{\mathcal{X}}(x) \, dx$. $\mathbb{E}\left[\mathcal{X}\right]$ can be approximated for samples of $\mathcal{X}$. Similar to the median, using the expectation as the ideal value is a good choice when metrics values *close to the expectation* are considered favorable.

4) The Mode: The mode of a random variable is the most frequently occurring value, i.e., the value with the highest probability (density). For a probability density $f_{\mathcal{X}}$, the mode $\hat{x}$ is defined as $\hat{x} = \arg\max_{x \in \mathcal{X}} \, f_{\mathcal{X}}(x)$. Therefore, the mode of a probability distribution is a reasonable choice for an ideal value for transforming a metric into a distance. It yields low distances for metric values *close to the most-frequently occurring value*.

All of the above transformations have in common that they are *non-parametric*, i.e., the ideal value can be derived using descriptive statistics of (samples of) the distributions, regardless of the true distribution of the data. The interested reader is directed at introductory texts, such as [20]. It is certainly possible to use any other arbitrary user-preferred ideal value as long as it is supported by sufficient rationale. While a user-preferred ideal value may not express the best practice for a metric in software engineering, it might still represent a useful value in the user's context that helps to avoid producing an abnormal method, class, or package [17].

However, it is important to note that the shown transformations are sensitive to the context since they operate on the distributions of values that might differ for different contexts. Practically, they are also sensitive to the number of observations (sample sizes) for each context as they depend on *empirical* distributions.

Example. Let us assume a quality model based on metrics that do not have an absolute ideal value. Suppose we want to add a component to an existing software system. Without having an explicit notion of ideal quality, a desirable property of the to-be-added component is that it should *fit* into the existing system (which we assume has reasonable quality). For example, we would expect the new component to have methods with a certain coupling and complexity so that it can interact properly with the existing components. Suppose the corresponding metrics are significantly different from those of the existing components. In that case, we have likely introduced a component with lower quality, e.g., low maintainability (because the new component is too coupled and too complex). In other words, using the suggested transformations, such as median, expectation, or mode, allows us to find a *typical* value that best represents the existing code. Using the value of this statistic to transform observed new metric values into distances allows us to create a score for each metric interpretable as a quality that is valid for the property captured by that metric (coupling, size, complexity). Furthermore, the scores can be aggregated to an overall quality value that indicates how far off the new component is from the existing system. This approach continues to work for metrics that have an actual ideal value. For example, while the ideal value for VG [8] is 1, for an average meaningful software component, this value is hardly achievable. Instead, we might assume a more realistic value of 3.12 and score the distances from that (this value is the expectation for VG in the Qualitas.class corpus for the domain "Games").

C. Why Using Scores?

The most significant difference between a metric and its score is that the latter is *by definition* uniformly distributed. Scores are required for aggregation whenever a metric or the distance from ideal are *not* uniformly distributed. While we have a strong assumption that the metrics in the Qualitas.class corpus are not uniformly distributed, we will find empirical evidence for this to be true (otherwise a simple normalization would be sufficient). For that, we suggest performing a statistical test for uniformity. In particular, we will conduct a one-sample Kolmogorov–Smirnov Test (KS), where we check each metric from each domain against a possible uniform distribution.

D. Differences Between Metric Distributions in Contexts

When operationalizing a metric as a score in a certain context, the question of whether the context matters needs to be answered. The answer can be given from two perspectives: Are the contexts distinguishable from each other and/or are the metrics different conditional on the context? For the Qualitas.class corpus where the domain gives the context that each software system is associated with, we need to estimate whether statistically significant differences exist. We suggest a test suite containing three tests in particular.

1) Analysis Of Variance (ANOVA): This test analyzes the differences among means [21]. The procedure is to check, for each metric, if its mean is significantly different in each context. The null hypothesis of this test is that there are *no* significant differences. This test yields a p-value and an F-statistic. The latter is the mean square of each independent variable divided by the mean square of the residuals. Large F-statistics indicate that the variation among contexts is likely. The p-value then indicates how likely it is for the F-statistic to have occurred, given the null hypothesis is true.

2) Two-sample Kolmogorov–Smirnov Test (KS2): This test is non-parametric and tests whether two samples stem from the same probability distribution [22]. KS2 does not check for a certain type of probability distribution since it uses the samples' empirical CDFs. Its test statistic is the maximum vertical distance between the two CDFs. For two samples $\boldsymbol{x}, \boldsymbol{y}$, the statistic is calculated as $D_{\boldsymbol{x},\boldsymbol{y}} = \sup_t \mid F_{\mathcal{X}}(t) - F_{\mathcal{Y}}(t) \mid$. The null hypothesis is that the samples' CDFs are identical, that is, $F_{\mathcal{X}} = F_{\mathcal{Y}}$. This test is used to compare one metric between two contexts.

A potential alternative to KS2 is Welch's t-test which is used to test whether two samples have equal means [23]. As a variant of Student's t-test, it should be preferred over it, since it tolerates unequal variances and sample sizes (which is almost always the case with our data). It should be noted that both tests assume that the samples' means are normally distributed. For a large number of sample means, this follows from the central limit theorem. However, since we only have comparatively few samples, it cannot be guaranteed that the assumption holds. Therefore, we only use the KS2 test in this work.

3) Tukey's Honest Significance Test (TukeyHSD): This test is used to gain insights into the results of an ANOVA test. While the former only allows obtaining the amount of corroboration for the null hypothesis, TukeyHSD performs all pairwise comparisons [24]. For example, by choosing a specific metric and domain in the Qualitas.class corpus, we obtain a list of other domains that are significantly statistically different. The null hypothesis of this test is the same as for the ANOVA test.

IV. Application: Metrics as Scores

This work is accompanied by an interactive open-source application called *Metrics As Scores* (MAS). It is partially shown in Figure 1. Currently, it loads the Qualitas.class corpus by default. The goal is for the application to support additional datasets in the future. Its purpose is to enable the user to inspect, in great detail, the corpus' metrics' conditional distributions, and to check what score, cumulative probability, or relative likelihood a metric's value corresponds to. For that, it currently supports various types of PDFs, CDFs, and CCDFs (which are used to determine scores). It is also interesting to visually inspect the distributions and to find, for instance, where large accumulations of certain values for some metric are. A total of five transforms are available for each metric in each domain. These are $\mathbb{E}[X]$ (expectation), median (50th percentile), mode (most likely value), infimum, and supremum (minimum and maximum observed value). These transforms cover the most common ideal values derived from descriptive non-parametric statistics. It is not yet possible to choose

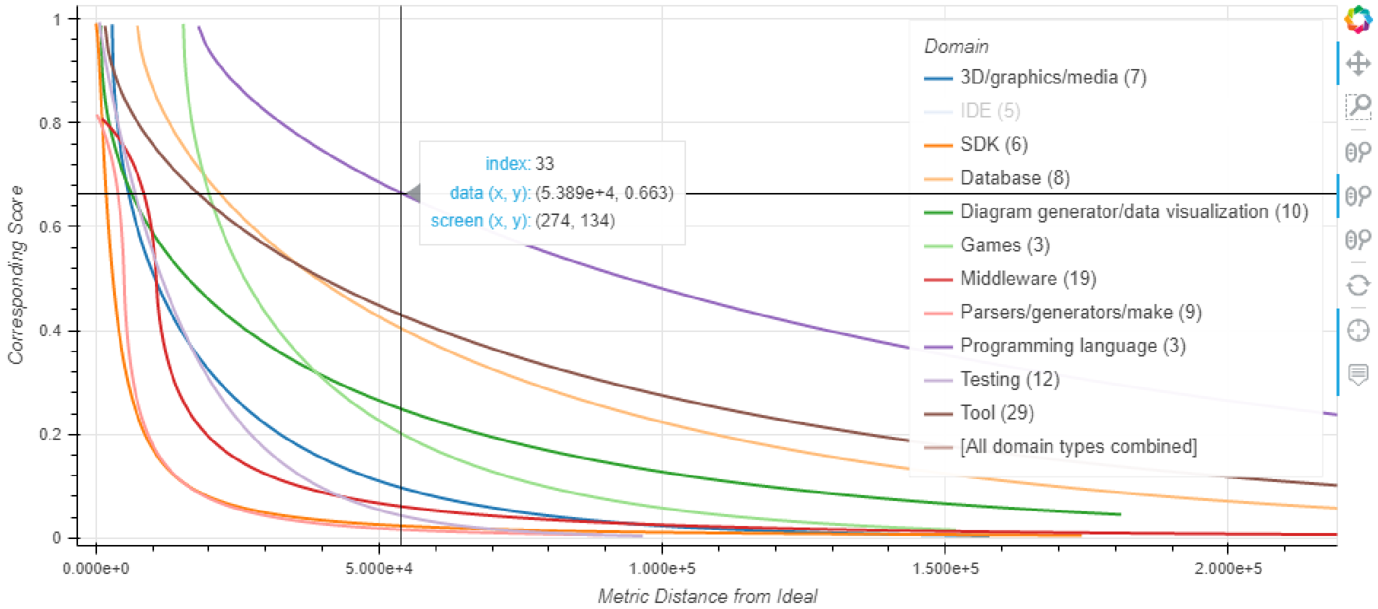

Figure 1. Main plot area of the application "Metrics As Scores". Using the Qualitas.class corpus, metrics values of own applications can be scored against the corpus' domains. Shown are the CCDFs (scores) of the fitted parametric distributions for the metric TLOC transformed using the infimum (per domain). Available online: https://metrics-as-scores.ml/.

your ideal value because all of the distributions were precomputed and are held in memory to guarantee a tractable user experience.

The application's main interactive element is the plot of distributions. It allows the user to zoom, pan, enable/disable domains, and manually hover the graphs to obtain precise x/y-values. While not shown here, the application also features an interactive table and additional controls. Per domain, the table shows the used transformation value (if used), the user's entered metric's distance from the ideal value (if a transform was applied), the score, cumulative probability, or relative likelihood of the user's (transformed) value, and the name and KS test's D-statistic of the parametric distribution that fitted the data best (if the type of selected distribution is parametric). There are three drop-down controls for selecting the metric type, distribution, and transformation. An additional button makes sure the shown plot occupies all available space. An input field for the user to check if their metric value exists. An additional checkbox automatically transforms that value into a distance using the current domain's transformation (if any). Lastly, the application contains an extensive help section and visual aids for displaying user values and the cutting-off of smoothed distributions beyond actual observed values. The application is written in Python and made available on GitHub.

A. Estimating Probability Density

While the data in the Qualitas.class corpus is extensive, many metrics' values are repetitive or exhibit low variance. This is often due to the nature of the metrics since they are merely counts of, e.g., attributes or events and, therefore, discrete. Numerically, we can treat any metric as a continuous random variable, even if it is discrete. Continuous probability distributions will naturally increase the entropy of discrete metrics with only a few unique values. Also, it allows us to observe the likelihood of values between two discrete values. Therefore, we offer more than one kind of probability distribution.

1) Empirical: Obtaining the empirical CDF (ECDF) is straightforward and accurately represents the observed data. For obtaining exact scores, the ECDF should always be preferred. However, it may exhibit large jumps. The PDF of any finite random sample is a discrete probability mass function (PMF) that assigns a fractional probability to each value, depending on how often it appears. For variables with many unique samples, this is not feasible. One remedy for this situation is the usage of histograms and binning. Histograms, however, are misleading[2] and have many shortcomings, such as a strong dependence on the number of bins (resolution), binning of otherwise relevant values, strong dependence on the variable's infimum and supremum, and the resulting difficulty of comparing distributions. Because of these, the application does not offer PMFs for discrete metrics.

2) Kernel Density Estimation (KDE): In density estimation, a smooth kernel is fit to the empirical data. The most common kernel used is the Gaussian kernel. KDE is a non-parametric method to estimate a continuous PDF and works with discrete and continuous data. Using KDE over histograms is highly recommended, as it adequately eliminates most of its shortcomings. Since we are more interested in cumulative distributions for the application, we obtain a CDF by inverse sampling from a KDE-fitted PDF and then obtain the ECDF

[2] See, for example, https://aakinshin.net/posts/misleading-histograms/.

of that sample. That is, given the estimated PDF f_{KDE}, the CDF is defined as $F_{\text{KDE}}(x) = \int_{-\infty}^{x} f_{\text{KDE}}(x)\,dx$. Typically, we take a sample $\hat{\boldsymbol{x}}$ from F_{KDE}^{-1} that is much larger than the sample $\boldsymbol{x}$ that was used for estimating the PDF using KDE to ensure smoothness of the resulting ECDF, i.e., $\hat{\boldsymbol{x}} \ggg \boldsymbol{x}$.

3) Parametric: Fitting a parametric probability distribution combines most of the advantages of the other approaches. Finding the parameters, such as scale or location, allows us to use a closed-form expression for both PDF/PMF and CDF. However, most metrics in the Qualitas.class corpus follow extreme value distributions and/or are heavily skewed, such that no such distribution can be fit using a common p-value threshold. The goodness-of-fit was determined using a Kolmogorov–Smirnov test [22]. If more than one distribution was eligible, the one with the lowest D-statistic was used. The few metrics for which a parametric probability distribution could be found are included.

V. Results

In this section, we demonstrate results that relate to the posed research questions. Each subsection corresponds to one of the research questions. For a qualitative discussion of these results, refer to the next Section (VI).

A. Uniformity Tests For Metrics

For the first research question, we attempt to find out whether or not the metrics of the Qualitas.class corpus are uniformly distributed. We perform the one-sample KS test for each of the 23 metrics in each of the 12 domains (including a virtual domain that merges all metrics' values). In total, we perform $23 \times 12 = 276$ tests. Figure 2 shows some metrics' complementary cumulative distribution (CCDF) in a certain domain, together with what would have been the corresponding CCDF of a uniform distribution. This allows us to expose the (non-)uniformity. In 273 cases, the uniformity test reports no statistically significant corroboration for an actual uniform distribution. The three remaining cases are almost certainly outliers. The estimate is based on a mere three data points in the sample in two cases. In the third case, only thirteen data points result in a p-value of ≈ 0.0505. Therefore, the test suggests the sample is uniform. The D-statistic in all cases is greater than $\frac{1}{3}$, which suggests poor goodness of fit.

B. Significant Differences Among Metrics Across Domains

For the second research question, we apply two statistical tests in particular. For each of the 23 metrics, an ANOVA test is performed. It indicates whether the metrics' means vary significantly across domains. To better understand these differences, we also conduct the KS2 test. For each metric, we take all its recorded values from one domain and compare them to all of its values from another domain. We add a virtual domain for both tests in which we merge the values of all domains and effectively disregard the domain attribute. The ANOVA test also indicates whether metrics' values are different in a specific domain when compared to all recorded values. The KS2 test requires pairwise comparisons. We compare each metric and domain to this metric in all other domains. Since we have eleven domains plus one virtual domain ($n = 12$), the number of pairwise comparisons per metric is calculated as $n \times (n-1) \div 2 = 66$. In other words, each metric can significantly stick out anywhere between zero and 66 times.

TABLE I
Results of the ANOVA tests as conducted per metric.

Metric	p-value	F-statistic
CA	$5.68e^{-15}$	8.4089
CE	$8.46e^{-33}$	16.4429
DIT	0	418.9608
LCOM	$3.249e^{-137}$	61.2257
MLOC	0	146.4968
NBD	0	1146.7435
NOC	$4.36e^{-49}$	23.6001
NOF	$4.74e^{-49}$	23.5012
NOI	$3.56e^{-50}$	24.0754
NOM	$1.226e^{-160}$	71.1691
NOP	$9.75e^{-10}$	6.0359
NORM	$3.611e^{-154}$	68.4245
NSC	$8.36e^{-9}$	5.4768
NSF	$9.58e^{-30}$	15.0548
NSM	$2.91e^{-26}$	13.5088
PAR	0	1171.1105
RMA	$1.65e^{-65}$	30.7523
RMD	$2.57e^{-35}$	17.5561
RMI	$8.35e^{-77}$	35.6537
SIX	$8.57e^{-91}$	41.4512
TLOC	$1.14e^{-8}$	5.4974
VG	0	340.9044
WMC	$5.284e^{-109}$	49.2226

1) Results ANOVA: The results of the ANOVA tests are shown in Table I. It appears that in not a single case, we find sufficient corroboration in the form of a p-value ≥ 0.05; most p-values are almost zero. As expected, those which are zero exhibit large values for the F-statistic. So, we cannot accept the null hypothesis that metrics have equal means across all domains. Since we included the virtual domain that contains all metrics' values across all domains, this result also means that the domain from which metrics values are gathered is important.

2) Results KS2: The results of the KS2 tests are illustrated in Figure 3. For each metric, it shows the number of times the metric was *not* distributed statistically significantly different in other domains. Seven metrics (DIT, MLOC, NBD, NOF, NOM, VG, and WMC) have a count of zero. Those metrics are therefore *always* different, across all 66 pairwise comparisons. The average of how often a metric was similar is ≈ 9.43 times. The metrics PAR, SIX, NSF, NORM, LCOM, RMD, CA, and RMI are rarely similar with a frequency in the range $[1, 11]$. The metrics NSC, NSM, RMA, NOC, CE, and NOI are moderately often similar with a frequency between $[14, 21]$. The metrics NOP (29) and TLOC (34) on the other hand are frequently similar.

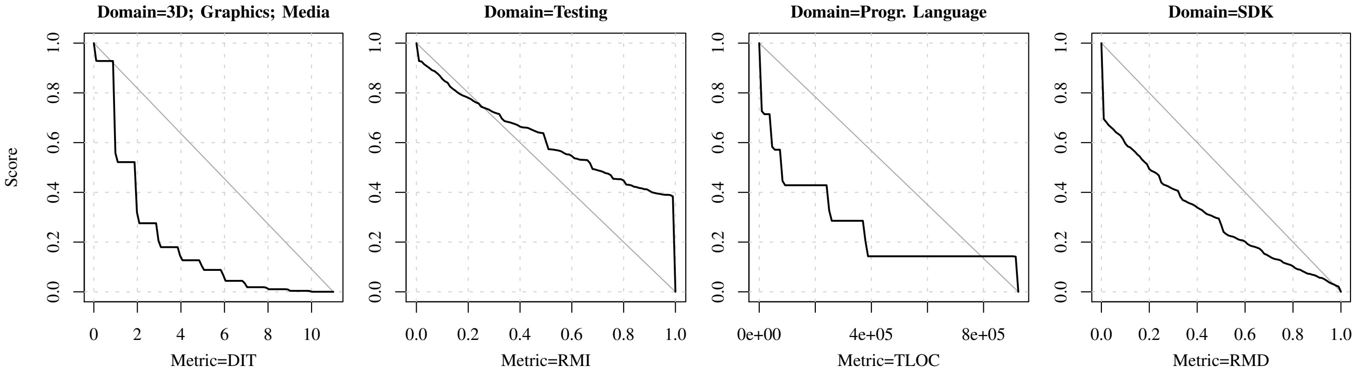

Figure 2. Four randomly chosen ECCDFs from the Qualitas.class corpus (276 possibilities). The lighter gray line indicates what would have been the analog uniform CCDF, were the corresponding metric distributed uniformly. Mind the different scales on the x-axis.

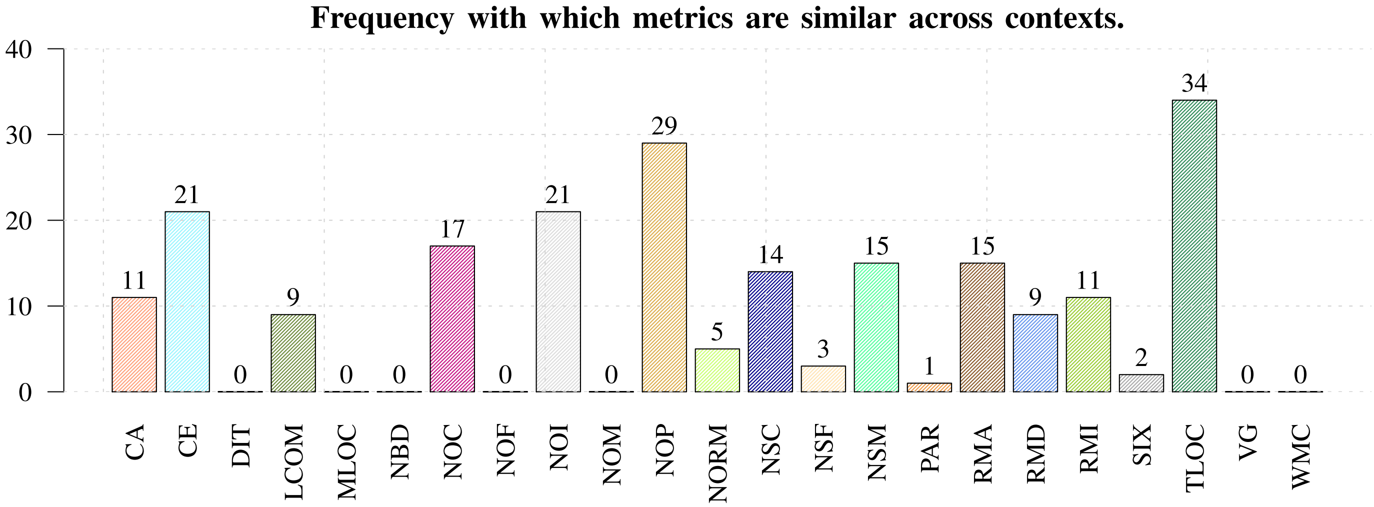

Figure 3. The frequency with which metrics were considered to come from the same distribution out of 66 pairwise comparisons across domains using the KS2 test. The metrics DIT, MLOC, NBD, NOF, NOM, VG, and WMC are always considered significantly statistically different across domains.

C. Discrimination Of Domains

For the third question, we analyze how much the different domains distinguish themselves from each other. We apply the TukeyHSD test and perform pairwise comparisons, i.e., we compare each domain to each other domain. We then count the number of metrics that are statistically significantly different. Since we include the virtual domain that merges all values, we end up with twelve domains. Therefore, we can maximally obtain $23 \times 11 = 253$ different metrics for a single domain.

1) Results TukeyHSD: On average, a total of 136.5 (53.95%) metrics were different across all domains (with a range of $[115, 198]$). The domain with the combined least differences was "Testing," and the domain with the most differences was "Programming Language" (Figure 4). These results are highly aggregated, so we also provide a more detailed view of the differences in Table II. In it, for each domain, we show the corresponding other domain(s) that has (have) the least and most metric differences (out of possible 23). In other words, this table shows, for each domain, the most similar and most dissimilar other domain(s). The range of values is $[5, 21]$. For example, the domains "Databases" and "Diagrams; Visualizations" are least distinguishable from the domain "Testing", while they are quite distinct from the domain "Programming Language". We observe that in eight out of eleven cases, "Programming Language" is the most dissimilar domain from any other domain in comparison.

D. Good/Bad Or Common/Extreme Values

For the last question, we extract some interesting values from the Qualitas.class corpus after transforming the metrics to scores. All eleven (+1 common) domains and all 23 metrics would break the mold of this paper. Therefore, we decide to show quantiles of scores of the metrics VG, TLOC, NOP, and CA. Only the metric VG has a minimum possible value of $= 1$, which is also the most desirable value. The three other metrics require a user-preferred ideal value for transformation into a score. Instead of picking these, we derive them from each domain using non-parametric statistics. We use the expectation ($\mathbb{E}[X]$), median, and mode. The scores then reflect the distance

TABLE II
THE MOST SIMILAR AND DISSIMILAR DOMAIN FOR EACH DOMAIN IN TERMS OF HOW MANY METRICS WERE FOUND TO BE DIFFERENT IN THE MUTUAL OTHER DOMAIN (OUT OF POSSIBLE 23). VALUES BELOW THE 25TH- OR ABOVE THE 75TH PERCENTILE ARE IN BOLD SCRIPT.

Domain	Lowest	Lowest Other Domains	Highest	Highest Other Domains
[All combined]	9	3D; Graphics; Media	**19**	Progr. Language
3D; Graphics; Media	7	Tool	17	Progr. Language
Databases	6	Testing	**20**	Progr. Language
Diagrams; Visualiz.	**5**	Testing	**19**	Progr. Language
Games	9	3D; Graphics; Media	14	SDK
IDE	10	SDK	**18**	Middleware
Middleware	8	Testing	**21**	Progr. Language
Parsers; Generators	9	Databases	16	Progr. Language
Progr. Language	12	Games	**21**	Middleware
SDK	9	Diagrams; Visualiz.	**18**	Progr. Language
Testing	**5**	Diagrams; Visualiz.	**20**	Progr. Language
Tool	7	3D; Graphics; Media	**18**	Middleware

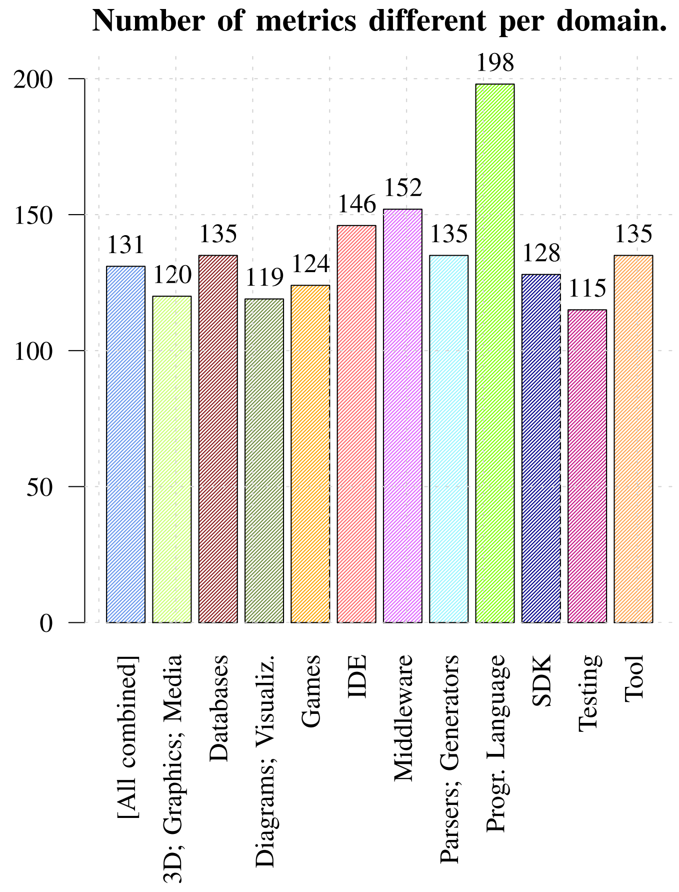

Figure 4. Number of metrics that are different per domain, by comparing each domain to all other domains. The maximum number of differences is 253.

from these statistically chosen ideal values. While this could be interpreted as a quality, it can give at least a notion of what is a common or extreme value for a certain metric and domain.

The quantiles in Table III and Table IV were chosen deliberately to examine the scores' behavior for extreme metrics values. Since scores have a standard uniform distribution, their quantiles are linearly anti-proportional. For metrics, however, going from one percentile to the next (e.g., from 10% to 20%) will *not* result in the same change as going from 20% to 30%. The results for the first research question showed that practically none of the corpus' metrics has a uniform distribution. Therefore, we cannot rely on a linear behavior for going from one percentile to the next. For example, we often observe large jumps in metrics values between a score of 0.95/0.99 and 1. Therefore, the results in both tables show the non-linear behavior of metrics' values (which is a direct result of their underlying non-uniform distributions). For instance, the expectation for TLOC is $\approx 2.5e^4$ and to achieve a score of 1, a distance of 35 (or fewer) lines of code is required in the virtual domain that combines all metrics' values. However, to achieve a score of 0.99, a distance of less than or equal to $\approx 1,630$ is required. This means that the tolerance for distances is quite large for TLOC, which makes sense since we observe values in the hundreds of thousands. Another example is that of VG and the difference among domains "3D; Graphics; Media" and "Tool". Lower complexities are much more *normal* in the former domain. We can achieve a score of 0.5 by reaching a complexity of less than or equal to ≈ 1.667. In the latter domain, however, a score of 0.5 corresponds to a complexity of no more than ≈ 2.707 ($+\,1.04$ in complexity).

VI. DISCUSSION

The goal of this work was to gather empirical evidence for the fact that software metrics should better be operationalized, i.e., normalized and contextualized in a certain domain, to answer the question *"is my metric value good?"*. For that, we first suggest observing sufficiently many values of a metric and then transforming it into a score. The normalizing transformation, however, needs to be justified. This justification was given by the results of the first research question. We find that all but three metrics are not uniformly distributed and therefore warrant said transformation. The three metrics that follow a uniform distribution are all justifiably outliers. Also, the transformation to a score does not hurt, even if the metric already follows a uniform distribution. In that case, the rectification goes without effect. Recall that we cannot compare any two metrics if their distributions are not exactly the same. Even if two metrics' distributions are the same, they

TABLE III

THE REQUIRED MAXIMUM DISTANCE FROM THE IDEAL VALUE TO ACHIEVE A SCORE LESS THAN OR EQUAL TO x FOR THE METRICS VG AND TLOC. WHILE THE FORMER HAS AN IMPLICIT IDEAL VALUE, THE LATTER IS TRANSFORMED USING THE DOMAIN'S EXPECTATION.

	VG (transformed using *implicit constant* ideal value = 1)								TLOC (transformed using explicit ideal value = $\mathbb{E}[X]$)								
Domain / Score	0	0.1	0.25	0.5	0.75	0.875	0.95	1	0	0.1	0.25	0.5	0.75	0.875	0.99	1	$\mathbb{E}[\boldsymbol{X}]$
[All combined]	1159	5.726	3.037	1.474	0.638	0.303	0.118	0	$2.46e^6$	$3.04e^4$	$3.04e^4$	$2.2e^4$	$1.74e^4$	$1.2e^4$	1630	35	$\mathbf{2.456e^4}$
3D; Graphics; Media	191	5.972	2.346	0.667	0.187	0.083	0.031	0	$1.47e^5$	$7.32e^4$	$4.3e^4$	$3.56e^4$	$2.44e^4$	$1.62e^4$	8438	7678	$\mathbf{3.756e^4}$
Databases	338	4.740	1.858	0.759	0.335	0.160	0.062	0	$5.45e^5$	$3.29e^5$	$1.1e^5$	$1.07e^5$	$1.02e^5$	$8.36e^4$	$1.37e^4$	$1.27e^4$	$\mathbf{1.082e^5}$
Diagrams; Visualiz.	284	4.282	1.793	0.691	0.313	0.151	0.059	0	$1.66e^5$	$1.1e^5$	$7.19e^4$	$4.77e^4$	$2.9e^4$	$1.92e^4$	8425	7310	$\mathbf{6.747e^4}$
Games	366	8.050	3.169	1.267	0.562	0.269	0.105	0	$1.49e^5$	$1.35e^5$	$1.2e^5$	$9.89e^4$	$6.57e^4$	$3.73e^4$	$1.75e^4$	$1.59e^4$	$\mathbf{1.223e^5}$
IDE	877	6.125	3.216	1.606	0.693	0.328	0.128	0	$2.33e^6$	$3.06e^4$	$1.84e^4$	$1.49e^4$	$1.15e^4$	8361	1390	148	$\mathbf{1.781e^4}$
Middleware	390	4.226	1.765	0.785	0.337	0.159	0.062	0	$3.13e^5$	$5.83e^4$	$3.43e^4$	$2.78e^4$	$1.74e^4$	9054	2059	1183	$\mathbf{3.305e^4}$
Parsers; Generators	280	5.647	2.222	0.937	0.419	0.201	0.079	0	$1.57e^5$	$9.84e^4$	$2.87e^4$	$2.72e^4$	$2.45e^4$	$1.88e^4$	1260	751	$\mathbf{2.764e^4}$
Progr. Language	280	6.110	3.051	1.529	0.660	0.313	0.122	0	$7.24e^5$	$6.53e^5$	$2.54e^5$	$2.01e^5$	$1.48e^5$	$1.07e^5$	$1.15e^4$	5250	$\mathbf{2.387e^5}$
SDK	618	4.349	2.543	0.996	0.433	0.206	0.080	0	$2.04e^5$	$1.41e^4$	$1.01e^4$	8113	5556	3524	747	236	$\mathbf{1.04e^4}$
Testing	338	4.496	1.883	0.772	0.354	0.172	0.068	0	$8.2e^4$	$6.63e^4$	$3.48e^4$	$2.88e^4$	$2.21e^4$	$1.5e^4$	1734	130	$\mathbf{3.279e^4}$
Tool	822	5.716	3.110	1.707	0.683	0.325	0.126	0	$3.61e^5$	$8.27e^4$	$6e^4$	$4.9e^4$	$3.54e^4$	$2.25e^4$	4925	3348	$\mathbf{5.927e^4}$

TABLE IV

THE REQUIRED MAXIMUM DISTANCE FROM THE MEDIAN (NOP) / MODE (CA) TO ACHIEVE A SCORE LESS THAN OR EQUAL TO x FOR THE METRICS NOP AND CA, WHICH BOTH HAVE NO IMPLICIT IDEAL VALUE. TRANSFORMATION IS APPLIED USING EACH DOMAIN'S MEDIAN/MODE.

	NOP (transformed using explicit ideal value from Median)									CA (transformed using explicit ideal value from Mode)								
Domain / Score	0	0.1	0.25	0.5	0.75	0.875	0.99	1	**50%**	0	0.1	0.25	0.5	0.75	0.875	0.95	1	**mode**
[All combined]	1416	27	9.72	7.26	4.77	3.08	0.70	0.38	**9.38**	6519	57	21	11	5.31	2.73	1.29	0.36	**3.64**
3D; Graphics; Media	67	49	19	15	9.30	5.04	1.01	0.57	**18**	308	56	13	4.85	2.79	1.70	0.85	0.05	**4.05**
Databases	812	407	90	53	44	38	27	26	**53**	1271	43	8.90	4.02	2.40	1.55	0.90	0.37	**3.37**
Diagrams; Visualiz.	376	80	52	32	18	12	5.28	4.68	**50**	994	38	7.68	3.64	2.27	1.49	0.90	0.39	**3.39**
Games	10	9.15	7.85	5.96	3.86	2.69	1.59	1.49	**42**	1264	51	17	11	7.45	5.14	2.78	0.13	**11**
IDE	492	8.14	5.92	3.55	2.02	1.17	0.18	0.08	**4.92**	3911	53	29	12	6.21	3.24	1.49	0.33	**6.33**
Middleware	603	95	29	20	11	6.29	1.36	0.82	**27**	841	27	6.31	2.52	1.46	0.97	0.67	0.47	**1.53**
Parsers; Generators	133	76	10	9.28	8.45	6.15	2.56	2.33	**10**	621	36	6.00	2.93	1.74	1.16	0.75	0.38	**2.38**
Progr. Language	364	348	73	63	50	37	26	25	**70**	2023	93	20	9.85	5.95	3.70	1.85	0.07	**10**
SDK	195	16	6.22	4.55	2.95	1.91	0.33	0.07	**6.93**	1199	60	9.52	5.35	3.12	2.07	1.22	0.42	**4.42**
Testing	146	117	37	27	17	9.63	2.25	1.42	**35**	353	33	6.94	2.60	1.56	1.02	0.60	0.24	**2.24**
Tool	362	87	36	24	13	6.94	0.94	0.34	**31**	1692	44	10	4.87	2.88	1.89	1.13	0.48	**4.52**

need to be uniform, as otherwise, we run into the next problem of non-linear scaling, hence the statistical tests conducted for research question one for a) same distribution and b) uniformity. The most important facet is perhaps comparability, not only across different metrics but also for the same metric across contexts. We find that without the transformation into scores, none of the original metrics possesses the trait of comparability.

The contextualizing transformation, too, needs to be justified. The results of the second research question show that every single metric is different in all of the domains found in the Qualitas.class corpus. The most interesting result is the one obtained using the KS2 test. We find that the seven metrics DIT, MLOC, NBD, NOF, NOM, VG, and WMC are never similar in pairwise domain comparisons. This implies that in software quality models, these metrics must be used with great caution and attention to the context they were obtained from. To a certain degree, this is also true for all of the remaining 16 metrics. This is because none of them was similar across all pairwise domain comparisons. When operationalizing a metric, the analyst wants to know whether or not its context matters. For example, even if we observe differences across contexts, they might still not matter. However, these differences are significant in the case of the Qualitas.class corpus.

In research question three, we examine the same problem from the perspective of each domain. We compare each domain to all other domains and aggregate the results. On average, each context is different by 53.95% (with a range of $[45.45\%, 78.26\%]$). This means that there are no subtle differences. The result of that analysis is that domains are not interchangeable. For example, having observed metrics in the domain "IDE" will yield misleading or even invalid results when operationalizing those metrics as scores and using these for software from the domain "Middleware". The domain "Programming Language" is significantly different from all other domains and often distinguishes itself from others. Table II shows the most (dis-)similar domain for each other domain. However, the amount of different metrics between any two domains is not lower than five (21.74% difference). Still, it goes as high as $\mathbf{21}$ (91.3% difference). Even the lowest numbers of differences will likely produce invalid scores, especially if the few metrics that are different are in the set of those seven that the KS2 test from research question two always found to have statistically significantly different distributions.

The point of research question four was to substantiate and demonstrate some of the differences caused by the significant differences among metrics and domains. We have previously claimed that, given sufficient contextual information, the question of whether a metric value is good/bad or common/extreme can be answered. The results obtained for the last research question make ample use of the metrics as captured across various contexts in the Qualitas.class corpus. We present results that substantiate our earlier claim by unveiling significant differences for the same metrics across these contexts (see

Tables III and IV). The most significant difference between a metric and its score is that the latter has linear behavior. That is, an improvement from, e.g., 0.2 to 0.5 is considered equally good as an improvement from 0.6 to 0.9. Practically none of the metrics in the Qualitas.class corpus is distributed *uniformly* (see results of research question one). That means, for example, given a metric with observed range $[0, 100]$, that the cumulative probability for any range of the same length (e.g., $10 \leq x \leq 20$ or $75 \leq x \leq 85$) will differ. This justifies and substantiates the necessity of the process of transforming a metric into a score, as otherwise, we cannot compare metrics or make statements about the goodness (quality) a change of a metric value actually corresponds to. The empirical results for research question four then give some direct indication as to the differences between metrics values and corresponding scores. To draw on the given example, we might assume that a metric change from 30 to 20 is equally good as the change from 20 to 10 when in reality it becomes exponentially more difficult to obtain lower and lower values. Only a score has the ability to "rectify" this behavior and return a value that reflects the actual improvement on a linear scale.

VII. Threats To Validity

The internal validity of our work is given and ensured by the research design. We have carefully discussed, and selected/dismissed different (un-)suitable statistical tests that allow us to detect significant differences among the data. In some cases, we have applied more than one test to detect (in-)significant results. We have attempted to exhaust the available combinations by trying all possible constellations and pairwise comparisons. This ensures the robustness of the obtained results and leaves only very little to chance.

The external validity of the suggested approach of transforming metrics into scores has yet to be demonstrated using other datasets. It is fair to say that this work aimed to define a methodology for examining datasets containing multiple features (here: domains) that are conditioned on two or more contexts (here: domains). Therefore, there are no generalizable results per se for which we would require to ensure external validity.

VIII. Related Work

Many issues with metrics appear to stem from trying to associate them with quality properties directly [13], [2]. However, most metrics do not have meaning, especially when they are mere counts. Issues also arise in cases where such a meaning is attached, which is the case for some complexity metrics [25]. This happens for various reasons. One of them is the absence of a metric's clear definition. Other reasons include, for example, inconsistent behavior among metrics-extracting applications [26]. To still obtain quality from metrics, quality models that use some form of aggregation are used. Those then act essentially as regression models for quality [27]. Because of the difficulty of directly associating metrics with software quality, many resorts to using common metrics and quality models to predict faults (e.g., [14], [15]).

Using random variables and their associated probability distributions have been exploited previously for matters of software quality. For example, software reusability and reliability models, such as the Goel-Okumoto model [28], are fit to fault-frequency metrics. Other use probability distributions, such as the Weibull distribution, directly to model the (expected) timely accumulation of software defects [29]. These models are probability distributions that are then leveraged in predictive scenarios. More recently, attempts to derive quality from joint probability distributions were made [16]. That work, however, focused on aggregation and did not attempt to transform metrics to scores using some ideal-value function.

Previous studies concerning finding or deriving threshold values for metrics are most relevant to our work. Thresholds are used as decision rules for categorizing metrics' values (e.g., good/bad or high/low). Similar work analyzing several systems and domains empirically to find thresholds was done by [30]. They conclude that *no* significant differences among domains exist (while contrary to our results, the software systems and domains used were different from those used in this work). Later work uses the Qualitas.class corpus as well [17], disregarding its domains based on the previous study's findings. Both these studies derive (un-)common values from the metrics' distribution (if possible). While the former study assigns metrics values to one of three categories manually after inspecting the data, the latter study does so by defining percentiles manually after visual appeal. Other studies define threshold values empirically [5]. One study to use the context of metrics was [31]. There, the operationalization of metrics lies in empirically finding a threshold value suitable to distinguish between error and no error. A similar approach had been developed earlier in [4]. That work summarizes other work which suggests different thresholds for various metrics based on, e.g., surveys or experience (i.e., no statistical approaches). In any of these cases, however, those thresholds are not used to convert a metric into a distance and then into a score.

IX. Future Work

The advantage of the scores presented in this paper is that they are comparable in an apples-to-apples manner. This allows for a number of operationalizations. For example, we could obtain the metrics values and associated scores of a new system. Then, we would aggregate these scores and rank the new system according to which domain scores the highest. This ranking can be exploited for the system identification task, that is, to what domain the tested system belongs most likely. This could answer the question, "is my system really what it pretends to be?". It could also be used as a quality goal: When developing a new system for a certain domain, one might want to end up with a system that scores high in the target domain and lower in other domains. Since scores are freestanding quality properties, the aggregation of scores can also be used as a quality model. Our results showed that the context given by the domain in the Qualitas.class corpus matters. Therefore, we have already begun to rank each system within the corpus' domains to determine whether it is associated with the most suitable domain. Some early results show that this is not always the case.

The approach presented in this paper is meant to work with arbitrary but similarly structured datasets. We plan to test the approach and the application using other datasets that are not necessarily related to software. Supposedly, it should work with any dataset that contains features distributed conditionally.

Acknowledgments

This work was supported by the Linnaeus University Centre for Data Intensive Sciences and Applications (DISA) and the Swedish Research School of Management and IT (MIT).

We express our gratitude to the four anonymous reviewers whose comments helped to improve this paper and made it easier to understand.

References

[1] S. Wagner, K. Lochmann, L. Heinemann, M. Kläs, A. Trendowicz, R. Plösch, A. Seidl, A. Goeb, and J. Streit, "The quamoco product quality modelling and assessment approach," in *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, M. Glinz, G. C. Murphy, and M. Pezzè, Eds. IEEE Computer Society, 2012, pp. 1133–1142.

[2] C. Kaner *et al.*, "Software engineering metrics: What do they measure and how do we know?" in *In METRICS 2004. IEEE CS*. Citeseer, 2004.

[3] J. P. Miguel, D. Mauricio, and G. Rodriguez, "A review of software quality models for the evaluation of software products," *CoRR*, vol. abs/1412.2977, 2014. [Online]. Available: http://arxiv.org/abs/1412.2977

[4] S. Benlarbi, K. E. Emam, N. Goel, and S. N. Rai, "Thresholds for object-oriented measures," in *11th International Symposium on Software Reliability Engineering (ISSRE 2000), 8-11 October 2000, San Jose, CA, USA*. IEEE Computer Society, 2000, pp. 24–39. [Online]. Available: https://doi.org/10.1109/ISSRE.2000.885858

[5] T. L. Alves, C. Ypma, and J. Visser, "Deriving metric thresholds from benchmark data," in *26th IEEE International Conference on Software Maintenance (ICSM 2010), September 12-18, 2010, Timisoara, Romania*, R. Marinescu, M. Lanza, and A. Marcus, Eds. IEEE Computer Society, 2010, pp. 1–10. [Online]. Available: https://doi.org/10.1109/ICSM.2010.5609747

[6] R. B. Grady, *Practical software metrics for project management and process improvement*. Prentice-Hall, Inc., 1992.

[7] M. Ulan, W. Löwe, M. Ericsson, and A. Wingkvist, "Copula-based software metrics aggregation," *Softw. Qual. J.*, vol. 29, no. 4, pp. 863–899, 2021. [Online]. Available: https://doi.org/10.1007/s11219-021-09568-9

[8] R. Terra, L. F. Miranda, M. T. Valente, and R. da Silva Bigonha, "Qualitas.class corpus: a compiled version of the qualitas corpus," *ACM SIGSOFT Softw. Eng. Notes*, vol. 38, no. 5, pp. 1–4, 2013. [Online]. Available: https://doi.org/10.1145/2507288.2507314

[9] 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 *17th Asia Pacific Software Engineering Conference, APSEC 2010, Sydney, Australia, November 30 - December 3, 2010*, J. Han and T. D. Thu, Eds. IEEE Computer Society, 2010, pp. 336–345. [Online]. Available: https://doi.org/10.1109/APSEC.2010.46

[10] W. A. Carleton, Anita D.; Florac, *Measuring the software process: statistical process control for software process improvement*, ser. SEI series in software engineering. Addison-Wesley Professional, 1999.

[11] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[12] T. J. McCabe, "A complexity measure," *IEEE Trans. Software Eng.*, vol. 2, no. 4, pp. 308–320, 1976.

[13] E. Bouwers, A. van Deursen, and J. Visser, "Evaluating usefulness of software metrics: An industrial experience report," in *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, D. Notkin, B. H. C. Cheng, and K. Pohl, Eds. IEEE Computer Society, 2013, pp. 921–930.

[14] M. Caulo, "A taxonomy of metrics for software fault prediction," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 1144–1147.

[15] S. R. Aziz, T. A. Khan, and A. Nadeem, "Experimental validation of inheritance metrics' impact on software fault prediction," *IEEE Access*, vol. 7, pp. 85 262–85 275, 2019.

[16] M. Ulan, W. Löwe, M. Ericsson, and A. Wingkvist, "Introducing quality models based on joint probabilities," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 216–217. [Online]. Available: https://doi.org/10.1145/3183440.3195103

[17] T. G. Filó, M. Bigonha, and K. Ferreira, "A catalogue of thresholds for object-oriented software metrics," *Proc. of the 1st SOFTENG*, pp. 48–55, 2015.

[18] M. Harman and J. A. Clark, "Metrics are fitness functions too," in *10th IEEE International Software Metrics Symposium (METRICS 2004), 11-17 September 2004, Chicago, IL, USA*. IEEE Computer Society, 2004, pp. 58–69. [Online]. Available: https://doi.org/10.1109/METRIC.2004.1357891

[19] M. West, "Object-oriented metrics: Measures of complexity, by brian henderson-sellers, prentice hall, 1996 (book review)," *Softw. Test. Verification Reliab.*, vol. 6, no. 3/4, pp. 255–256, 1996.

[20] S. C. Jean Dickinson Gibbons, *Nonparametric Statistical Inference*, 4th ed., ser. Statistics, textbooks and monographs 168. Marcel Dekker, 2003.

[21] J. M. Chambers and T. J. Hastie, "Statistical models," in *Statistical Models in S*. Routledge, Nov. 2017, pp. 13–44.

[22] M. A. Stephens, "EDF statistics for goodness of fit and some comparisons," *Journal of the American Statistical Association*, vol. 69, no. 347, pp. 730–737, 1974. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01621459.1974.10480196

[23] B. L. Welch, "The generalization of "student's" problem when several different population varlances are involved," *Biometrika*, vol. 34, no. 1-2, pp. 28–35, 01 1947. [Online]. Available: https://doi.org/10.1093/biomet/34.1-2.28

[24] J. W. Tukey, "Comparing individual means in the analysis of variance," *Biometrics*, vol. 5, no. 2, pp. 99–114, 1949. [Online]. Available: http://www.jstor.org/stable/3001913

[25] M. J. Shepperd and D. C. Ince, "A critique of three metrics," *J. Syst. Softw.*, vol. 26, no. 3, pp. 197–210, 1994. [Online]. Available: https://doi.org/10.1016/0164-1212(94)90011-6

[26] M. Nilson, V. Antinyan, and L. Gren, "Do internal software quality tools measure validated metrics?" in *Product-Focused Software Process Improvement - 20th International Conference, PROFES 2019, Barcelona, Spain, November 27-29, 2019, Proceedings*, ser. Lecture Notes in Computer Science, X. Franch, T. Männistö, and S. Martínez-Fernández, Eds., vol. 11915. Springer, 2019, pp. 637–648. [Online]. Available: https://doi.org/10.1007/978-3-030-35333-9_50

[27] N. Fenton, *Software Metrics: A Rigorous and Practical Approach, Third Edition*, ser. Chapman & Hall/CRC Innovations in Software Engineering. CRC Press, 2014.

[28] A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE transactions on Reliability*, vol. 28, no. 3, pp. 206–211, 1979.

[29] E. Simmons, "When will we be done testing? software defect arrival modeling using the weibull distribution," in *Pacific Northwest Software Quality Conference, Portland, OR*. Citeseer, 2000.

[30] 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," *J. Syst. Softw.*, vol. 85, no. 2, pp. 244–257, 2012. [Online]. Available: https://doi.org/10.1016/j.jss.2011.05.044

[31] R. Shatnawi, W. Li, J. Swain, and T. Newman, "Finding software metrics threshold values using ROC curves," *J. Softw. Maintenance Res. Pract.*, vol. 22, no. 1, pp. 1–16, 2010. [Online]. Available: https://doi.org/10.1002/smr.404