A Study of the Effect of Data Normalization on Software and Information Quality Assessment

Morgan Ericsson, Welf Löwe, Tobias Olsson, Daniel Toll, and Anna Wingkvist
Department of Computer Science
Linnaeus University, Sweden
email: {Morgan.Ericsson | Welf.Lowe | Tobias.Ohlsson | Daniel.Toll | Anna.Wingkvist}@lnu.se

***Abstract*—Indirect metrics in quality models define weighted integrations of direct metrics to provide higher-level quality indicators. This paper presents a case study that investigates to what degree quality models depend on statistical assumptions about the distribution of direct metrics values when these are integrated and aggregated. We vary the normalization used by the quality assessment efforts of three companies, while keeping quality models, metrics, metrics implementation and, hence, metrics values constant. We find that normalization has a considerable impact on the ranking of an artifact (such as a class). We also investigate how normalization affects the quality trend and find that normalizations have a considerable effect on quality trends. Based on these findings, we find it questionable to continue to aggregate different metrics in a quality model as we do today.**

I. Introduction

Quality models define how to integrate and aggregate direct metrics to abstract scores that represent the quality of a software system or a technical documentation. These are useful to discuss an abstract notion such as quality with stakeholders. Approaches such as Goal-Question-Metric (GQM) [1] can help stakeholders focus on what to measure as well as why. Process models [2] help define weights for different metrics and questions according to their perceived impact on quality.

Different metrics have different scales and scale types, so values of metrics should be normalized to avoid comparing "apples and oranges". While stakeholders can easily validate direct metric values, they may find it difficult to validate normalized metrics scores and, hence, the integrated and aggregated quality scores. This can undermine trust in quality assessment as such, especially, if the results are unexpected.

We can use linear normalization models, that are easy to compute and to explain, or models that consider the distributions of the metrics values, which are mathematically more sound but rely on time consuming and error prone computations to normalize. In our experience, the latter are more difficult to explain to stakeholders and more difficult for them to validate. So, we ask ourselves: *Does normalization matter?* Do linear normalization models and normalization models based on approximate distributions result in: a) the same interpretation of quality in a specific release and b) the same quality trend?

We conduct a case study to investigate if different normalization approaches applied to the same set of metrics can, after integration in a quality model, result in different quality statements about the assessed software and documentation artifacts. The real world subjects span software and information, different programming and specification languages, as well as different branches of industry.

II. Background

The Goal-Question-Metric paradigm provides a method to create quality models that define conceptual quality *Goals*, operational *Questions*, and quantitative *Metrics*. To assess if a goal is achieved, the model is followed in reverse order of definition. The artifact is measured, and the metrics values are integrated (added) to answer the question in a quantitative way. Questions can be considered indirect measures whose values can be further integrated to provide a numerical score for the quality goal, which can be interpreted as the degree to which an artifact posses the desired quality properties.

All questions are not equally important and all metrics are not equally good indicators to answer a question. To reflect this, questions and metrics are weighted, and direct and indirect metrics are integrated using weighted sums. When computing indirect metrics corresponding to questions and quality goal, it is not meaningful to just add values of different direct metrics like, e.g., Number of call sites and Lines of code. Values should be normalized before they are integrated. There are different ways to normalize metrics values from different metrics. To simplify the discussion all our normalizations will compute scores between 0 and 1.

$$\mathit{Normalize}(X) = \frac{X - min}{max - min}. \tag{1}$$

We can normalize values by looking at max and min values in each domain according to Eq. 1. This corresponds to linear regression models. The values are now in the same range, but it is not more appropriate or mathematically sound to add them. We usually assume that extreme metrics values are indicators of bad quality. Whether large or small values are bad depends on the metrics and the quality goal. For this discussion we can, without loss of generality, assume that large values are bad. Assume the distribution of all possible values of a metric is known. Given a cumulative probability function of such a distribution, we can, for each value we measure, calculate the probability that this or a smaller value is observed. Very large (bad) values have a probability close to 1, and very small (good) values a probability close to 0.

$Normalize(X)$ is the cumulative probability of a metrics value X (i.e., the probability that other metrics values are $\leq X$) if the values of that metric are uniformly distributed between min and max. If, however, the metrics values were normally distributed, cumulative probabilities would be computed according to Eq. 2, where μ is the mean and σ the standard deviation.

$$\mathit{NormDist}(X) = \frac{1}{2}\left(1 + \mathit{erf}\left(\frac{X-\mu}{\sqrt{2\sigma^2}}\right)\right) \quad (2)$$

$$\mathit{erf}(x) = \frac{1}{\sqrt{\pi}}\int_{-x}^{+x} e^{-t^2}dt. \quad (3)$$

Given a fitted distribution for each metric, the normalized metrics scores indicate how extreme a measured value is compared to other values of a metric. Hence, these scores are comparable for different metrics. In a quality model, we can now integrate different metrics of an artifact by computing (weighted) sums of these normalized metric scores.

However, most software metrics are neither uniformly nor normally distributed [3, 4]. In order to find a sound normalization using cumulative probabilities, there is a need to consider and fit alternative distributions.

III. Methodology

We conduct an exploratory case study to investigate how different normalization models affect software and information quality assessment. The subject of investigation is the complete automated quality assessment process of 3 companies; the quality model, the metrics, the system (or documentation), and the assement tools. We rely on 3 subjects to provide data source triangulation.

Subject A is the information quality assessment process (cf. Figure 1) of a telecom infrastructure provider applied to one of their documentations. The documentation is in XML and consists of 18 parts (from 8 KiB to 8.5 MiB).

Subject B is the software quality assessment process(cf. Figure 2) of a small games development company applied to 6 of their games. The games are developed in C++ and have between 2,500 and 10,500 lines of code.

Subject C is the software quality assessment process (cf. Figure 3) of a large mobile hydraulics company applied to their integrated development tool. This tool is used to develop hydraulics control software in a domain-specific language. We follow their assessment over one year (8 minor versions). The tool is implemented in Delphi and consists of 110 Delphi Units. A single Unit contains 1 – 266 files.

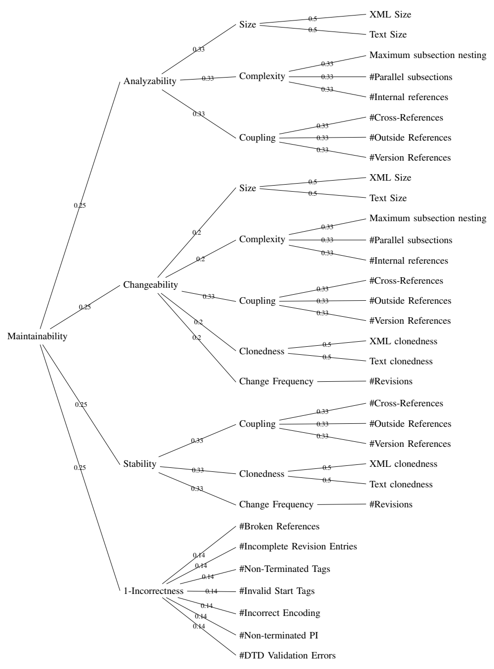

Figure 1. Quality model in Subject A.

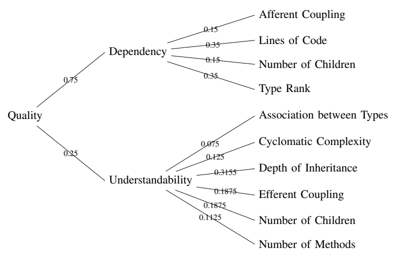

Figure 2. Quality model in Subject B.

For each subject, we use 3 different normalizations to integrate the direct metrics in the quality model. We use 1) linear regression ($Normalize$, cf. Eq. 1), 2) $NormDist$ (cf. Eq. 2), and 3) the best fit of the following distributions: Beta Distribution, Exponential Distribution, Extreme Value Distributions (Smallest Extreme Value, Largest Extreme Value, Weibull, Fréchet), Gamma Distribution, Johnson Family of Distributions, Logistic and Loglogistic Distributions, Log-

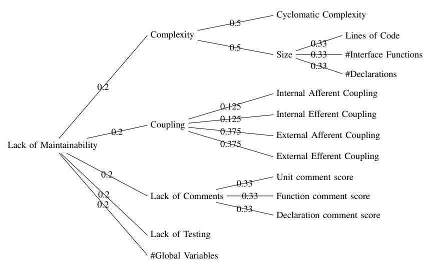

Figure 3. Quality model in Subject C.

Table I
BEST FIT DISTRIBUTIONS FOR EACH OF THE METRICS. PARAMETERS ARE OMITTED. $*$ DENOTES THAT THE SAMPLE DID NOT PASS A NORMALITY TEST AFTER TRANSFORMATION.

	Metrics	Best Fit Distribution
Subject A	XML Size	Beta applied to $ln(X)$
	Text size	Beta applied to $ln(X)$
	#Broken References	Beta*
	#Incomplete Revision Entries	Beta*
	#Non-Terminated Tags	Beta*
	#Invalid Start Tags	Beta*
	#Incorrect Encoding	Beta*
	#Non-Terminated PI	Beta*
	#DTD Validation Errors	Largest extreme value*
	Maximum subsection nesting	Johnson applied to $ln(X)$*
	#Parallel subsections	Johnson applied to $ln(X)$*
	#Internal references	Johnson*
	XML clonedness	Pearson applied to $ln(X)$
	Text clonedness	Beta*
	#Cross-References	Largest extreme value*
	#Outside References	Largest extreme value*
	#Version References	Largest extreme value*
	#Revisions	Exponential*
Subject B	Lines of Code	Gamma applied to $ln(X)$
	Number of Children	Beta applied to $ln(X)$*
	Type Rank	Largest extreme value applied to $ln(X)$*
	Afferent Coupling	Largest extreme value applied to $ln(X)$
	Efferent Coupling	Largest extreme value applied to $ln(X)*$
	Association between Types	Lognormal*
	Depth of Inheritance	Normal*
	Number of Methods	Gamma applied to $ln(X)$
	Cyclomatic Complexity	Johnsson applied to $ln(X)$
Subject C	Lines of Code	Johnson applied to $ln(X)$
	#Interface Functions	Largest extreme value applied to $ln(X)$
	#Declarations	Largest extreme value applied to $ln(X)$
	Cyclomatic Complexity	Pearson applied to $ln(X)$
	Afferent Internal Coupling	Largest extreme value applied to $ln(X)$*
	Efferent Internal Coupling	Largest extreme value applied to $ln(X)$*
	Afferent External Coupling	Beta*
	Efferent External Coupling	Beta*
	Unit comment score	Beta*
	Function comment score	Largest extreme value*
	Declaration comment score	Beta*
	#Global Variables	Johnson*
	Lack of Testing	Beta*

normal Distribution, Normal Distribution, Pearson Family of Distribution (Includes Inverse Beta and Inverse Gamma), and Uniform Distribution. We refer to the selected (parameterized) distribution as $BestFit$. Note that the $BestFit$ distribution is just the optimum among the set of tested transformations — there is no guarantee that the p-value is below a certain significance level, e.g., $\alpha = 0.05$. So, there can be significant departures from a normal distribution. Table I shows $BestFit$ for each metrics, followed by a $*$ if the sample did not pass a normality test after transformation.

For each of the distribution based normalizations N, we define a derived normalization, $N > 80\%$: A metrics value X gets a score of $N_{>80\%}(X) = 1$ iff $N(X) > 0.8$ and $N_{>80\%}(X) = 0$ otherwise. The motivation is to further abstract the metrics and only distinguish critical metrics values from acceptable values. Finally, we use 3 normalizations that are agnostic to distributions. First, we compute the z-score of a metrics value and abstract the z-score further to distinguish critical from acceptable metrics values: $z_{>4.5}(X) = 1$ iff $z(X) > 4.5$ and $z_{>10}(X) = 1$ iff $z(X) > 10$, respectively. The motivation for these bounds is that (metrics) values from a normal distribution with a z-score larger that 4.5 are considered extreme and, regardless of the distribution, a z-score larger that 10 is considered an extreme value even for long-tailed distributions.

For each subject, the metrics data is normalized and aggregated, and a rank is determined. Each subject consists of a number of entities, e.g., classes, documents, and files. The metrics values for each entity are normalized. We compute the average of the normalized scores of the entities in their structural containers (e.g., a method is *contained* by a class). For example, in Subject B, we compute the average of the normalized Lines of Codes (and all other metrics') scores the classes contained by each game. We use the following containers: A) the package a document belongs to, B) the game a class belongs to, and C) the Delphi Unit a file belongs to.

The container ranked 1 is considered to have the "worst" quality, i.e., has the highest problem score. For each subject and normalization, we compare the different ranks of a specific container by computing the total edit distance[1]. The edit distance reflects the effect of various normalizations on the computed quality rank; the larger the edit distance, the larger effect the different normalizations have.

IV. ANALYSIS

Subject A considers the quality assessment of a technical documentation that consists of 18 packages (cf. Table II). The package ranks vary depending on normalizations; many packages are either considered to have good or bad quality. If we consider Package 19084, we find that linear normalization ranks the package as 2, while best fit ranks it as 8 out of the 18 packages. The package is either one of the worst or somewhere in the middle. Note that an entry in Table II depicts the ranking for an entire package, so even if the values are similar, there can still be a large difference in the rankings of individual documents and/or for individual metrics. The latter is analyzed in Table III. We compare the results of different normalization by looking at the average edit distance between rankings, i.e., the number of changes required to transform one to another. The average is computed over the edit distances between the rankings induced by the normalizations for each direct and indirect metrics. The edit distances of Subject A are generally quite large, which indicates that the normalizations produce different results even for individual metrics. For example, the distance between rankings induced by $Normalize_{>80\%}$ and $BestDist$ is 16.14 on average over all metrics assessed by Subject A.

Subject B considers the quality assessment of 6 computer games. Each game is measured, normalized and ranked as shown by Table IV. The ranks vary depending on normalization; almost every game is either considered best or worst. Consider, for example, the game "GoL". We find that if we use a linear normalization, the game is considered to have the best quality. However, if we only consider values that are larger than 80% as outliers of the population, it has the

[1]We use the Levenshtein distance between two ranking strings defined as the minimum number of edits needed to transform one string into the other; edit operations are insertion, deletion, or substitution of a single (rank) character.

Table II
THE EFFECT OF DIFFERENT NORMALIZATIONS ON THE QUALITY RANK OF THE XML PACKAGES OF SUBJECT A. THE PACKAGES ARE RANKED BY QUALITY PROBLEMS, SO RANK 1 HAS THE WORST QUALITY AND RANK 18 HAS THE BEST.

Package	Normalize	Normalize$_{>80\%}$	NormDist	NormDist$_{>80\%}$	z-score	z-score$_{>10}$	z-score$_{>4.5}$	BestDist	BestDist$_{>80\%}$
33	16	6	13	7	7	2	5	14	10
651	18	15	18	18	18	7	15	18	17
1531	9	13	9	9	10	3	11	7	8
1543	6	7	6	6	6	7	6	4	6
1550	15	14	17	14	16	4	12	16	14
1551	8	9	8	10	9	7	14	5	7
1553	10	10	10	11	13	7	8	11	12
10945	3	3	2	3	3	7	2	6	3
10948	1	2	1	1	2	7	3	1	1
12446	11	15	12	12	11	7	4	12	15
15372	5	5	5	5	5	7	15	2	2
15411	14	15	16	16	17	7	15	15	16
15451	13	11	15	17	14	7	13	17	18
19012	17	15	14	15	15	7	15	13	10
19080	7	8	7	8	8	6	10	9	9
19083	12	12	11	13	12	5	9	10	13
19084	2	1	3	2	1	1	1	8	5
22102	4	4	4	4	4	7	7	3	4

Table III
THE AVERAGE EDIT DISTANCE BETWEEN RANKINGS INDUCED BY DIFFERENT NORMALIZATIONS ON SUBJECT A. THE AVERAGE IS TAKEN OVER THE PACKAGE RANKINGS WITH RESPECT TO EACH DIRECT AND INDIRECT METRICS (NOT ONLY THE QUALITY RANKING).

	Normalize$_{>80\%}$	NormDist	NormDist$_{>80\%}$	z-score	z-score$_{>10}$	z-score$_{>4.5}$	BestDist	BestDist$_{>80\%}$
Normalize	13.24	8.48	11.29	3.38	14.38	13.05	9.71	10.95
Normalize$_{>80\%}$		16.19	12.76	14.48	8.95	8.67	16.14	13.00
NormDist			13.90	7.76	16.71	16.00	7.86	13.24
NormDist$_{>80\%}$				13.05	14.14	12.62	14.62	7.67
z-score					15.14	14.29	10.33	12.71
z-score$_{>10}$						9.67	16.67	14.10
z-score$_{>4.5}$							16.00	12.90
BestDist								12.76

worst quality. We find a similar effect when we consider the z-score; when we only look at the more extreme values, the quality of GoL drops from rank 4 (good) to 1 (worst). If we only consider the distribution-based normalization, we find less variation, but GoL still ranks as 5 or 6, depending on the normalization. Note that $Normalize$ and z-score produce an edit distance of 0.0 (cf. Table V), which suggests that they produce the same ranking, even though the rankings in Table IV differ. This is an effect of calculating the edit distance as the average of the rankings of all metric values and requires further investigation.

Our analysis of subjects A and B shows that the different normalizations have an effect; no two normalizations produces the same ranking for the total quality and the average edit distances over individual metrics rankings are quite high. It also suggests that the normalizations that consider only the extremes of a population are more unstable than rankings induced by the distribution-based normalizations. If we consider subject C, we find similar results. Tables VI and VII depicts the ranking and edit distance, respectively, for Subject C. Note that due to the large amount of Delphi Units in Subject C, Table VI only shows the worst 20. The edit distances are then computed on the rankings of all Delphi Units (hence the large values of almost 110).

The analysis of subjects A – C suggests that the choice of normalization has an impact on our understanding of the entities' and containers' quality. However, we only consider a single release in these 3 subjects, so it is not clear whether the normalization will affect our understanding of whether the quality improves or deteriorates over several releases of the same system. In order to determine if different normalizations converge to a single trend, we analyze eight uploads of a single system (in Subject C). Table VIII shows the trends of the overall quality using different normalizations. A plus $(+)$ denotes that the overall quality improved while a minus $(-)$ denotes that the quality worsened. Different normalizations result in different trends; the quality of the system can be considered to be only improving, only deteriorating, or almost anything in between, depending on which normalization we use. However, compared to the analysis of a single release, there is no major difference between distribution-based normalizations and the derived ones.

V. RELATED WORK

Several efforts, e.g., ISO/IEC [5], McCall et al. [6], and Fan et al. [2] suggest that different software measurements can be combined to form quality models and used to assess one or more quality attributes of a software system, such as maintainability. Efforts to develop models and tools to automatically assess these include VizzAnalyzer [7] and Crocodile [8]. The models and tools combine several measurements, and need to carefully consider the implications of (software) measurement theory, such as scale levels and what operations are allowed. Zuse [9] discusses the problems involved, such as the need for and problem of normalization of different measurements. For example, VizzAnalyzer assumes that the measurement results are of normal or uniform distribution. Kitchenham et al. [10] discuss the variation of a distribution, and why it is important to have a representative population model to be able to interpret the measured values. There is, however, no discussion on the

Table IV
THE EFFECT OF DIFFERENT NORMALIZATIONS ON THE QUALITY RANK OF THE GAMES OF SUBJECT B. THE GAMES ARE RANKED AFTER QUALITY PROBLEMS, SO RANK 1 HAS THE WORST QUALITY AND RANK 6 HAS THE BEST.

Game	Normalize	Normalize$_{>80\%}$	NormDist	NormDist$_{>80\%}$	z-score	z-score$_{>10}$	z-score$_{>4.5}$	BestDist	BestDist$_{>80\%}$
GoL	6	1	5	5	4	1	4	5	5
Hero	2	3	3	1	2	3	5	3	1
TWTPB	5	4	6	6	5	2	1	6	6
Time Breaker	1	5	1	2	1	3	2	2	2
Frontline	3	6	2	3	3	3	6	1	3
PirateQuest	4	2	4	4	6	3	3	4	4

Table V
THE AVERAGE EDIT DISTANCE BETWEEN RANKINGS FROM DIFFERENT NORMALIZATIONS APPLIED TO SUBJECT B.

	Normalize$_{>80\%}$	NormDist	NormDist$_{>80\%}$	z-score	z-score$_{>10}$	z-score$_{>4.5}$	BestDist	BestDist$_{>80\%}$
Normalize	4.42	2.67	3.33	0.00	4.83	4.25	3.00	3.83
Normalize$_{>80\%}$		4.42	4.58	4.42	3.75	3.75	4.42	4.75
NormDist			3.58	2.67	4.92	4.42	2.00	3.42
NormDist$_{>80\%}$				3.33	4.92	4.50	3.75	3.17
z-score					4.83	4.25	3.00	3.83
z-score$_{>10}$						3.50	4.92	5.08
z-score$_{>4.5}$							4.75	4.75
BestDist								3.42

Table VI
THE EFFECT OF DIFFERENT NORMALIZATIONS ON THE QUALITY RANK OF THE PACKAGES OF SUBJECT C. THE PACKAGES ARE RANKED AFTER QUALITY PROBLEMS, SO RANK 1 HAS THE WORST QUALITY. WE DISPLAY ONLY THE TOP 20 (WORST) PACKAGES OF THE 110 PACKAGES.

Rank	Normalize	Normalize$_{>80\%}$	NormDist	NormDist$_{>80\%}$	z-score	z-score$_{>10}$	z-score$_{>4.5}$	BestDist	BestDist$_{>80\%}$
1	78	21	104	57	78	78	78	104	57
2	6	7	21	104	27	38	34	13	34
3	2	78	57	21	7	7	38	57	12
4	7	6	34	83	38	28	43	33	33
5	21	2	6	33	34	57	94	5	13
6	104	13	17	4	57	12	7	21	5
7	33	11	33	34	21	56	27	4	54
8	11	33	4	7	104	21	28	54	31
9	4	8	2	6	54	11	54	34	97
10	8	4	7	78	17	17	57	6	99
11	5	5	54	2	6	27	37	17	4
12	17	24	22	75	4	6	12	88	10
13	13	16	13	17	28	2	56	2	35
14	1	1	11	8	11	1	97	7	104
15	16	104	75	22	75	3	101	12	36
16	34	17	8	41	2	4	21	41	88
17	24	46	41	13	33	5	18	1	101
18	41	32	1	11	13	10	75	11	75
19	57	41	61	61	12	11	93	75	81
20	88	88	5	5	22	12	13	32	17

Table VII
THE AVERAGE EDIT DISTANCE BETWEEN RANKINGS FROM DIFFERENT NORMALIZATIONS APPLIED TO SUBJECT C.

	Normalize$_{>80\%}$	NormDist	NormDist$_{>80\%}$	z-score	z-score$_{>10}$	z-score$_{>4.5}$	BestDist	BestDist$_{>80\%}$
Normalize	100.38	55.62	88.77	29.85	106.31	106.31	68.31	87.54
Normalize$_{>80\%}$		100.85	100.85	100.62	50.31	108.69	101.62	109.08
NormDist			96.08	59.77	107.00	106.54	78.31	103.62
NormDist$_{>80\%}$				97.31	107.00	106.08	98.62	91.62
z-score					106.46	106.38	76.92	101.54
z-score$_{>10}$						83.85	109.08	98.08
z-score$_{>4.5}$							108.38	98.54
BestDist								93.23

Table VIII
THE EFFECT OF DIFFERENT NORMALIZATIONS ON THE QUALITY RANK OF DIFFERENT VERSIONS OF THE SYSTEM IN SUBJECT C. A $+$ DENOTES THAT QUALITY HAS IMPROVED, $-$ DENOTES THAT IT WORSENED, AND $\sim$ DENOTES NO CHANGE OR NO REFERENCE.

Upload Date	Normalize	Normalize$_{>80\%}$	NormDist	NormDist$_{>80\%}$	z-score	z-score$_{>10}$	z-score$_{>4.5}$	BestDist	BestDist$_{>80\%}$
Upload 120101	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
Upload 120706	$+$	$+$	$+$	$-$	$+$	$-$	$-$	$+$	$+$
Upload 120806	$+$	$+$	$-$	$-$	$-$	$-$	$+$	$-$	$-$
Upload 120917	$-$	$+$	$-$	$-$	$-$	$+$	$+$	$-$	$-$
Upload 121011	$+$	$+$	$-$	$-$	$+$	$+$	$+$	$+$	$-$
Upload 121023	$+$	$+$	$-$	$-$	$-$	$-$	$-$	$-$	$-$
Upload 121119	$+$	$+$	$-$	$-$	$-$	$-$	$-$	$+$	$+$
Upload 121212	$+$	$+$	$+$	$-$	$+$	$-$	$+$	$+$	$-$

effects of normalization that assumes improper distributions, or how combinations of different distributions propagate. To our knowledge, there exist no such studies.

Some effort has been directed towards understanding the shape of software, i.e., the population models of different measurements. Early efforts include Knuth's study of FORTRAN programs Knuth [11] and Laemmel and Shooman [12] show that Zipf's law can be applied to operator and operand counts in software to derive measurements similar to those of Halstead. Barkmann et al. [4] investigates how measurements on open source Java programs are distributed, and found the skewness value positive. Wheeldon and

Counsell [13] also study measurements from open source Java programs and find that they are power-laws. In an attempt to reproduce the result of Wheeldon and Counsell using a larger corpus of Java programs, Baxter et al. [3] find that their results only provide support for power-laws for some measurements. Others follow what they refer to as a "truncated curve distribution". They suggest that programmer awareness and type of application can affect the distribution of certain measurements.

VI. Conclusions and Future Work

We find it questionable to continue to aggregate different metrics in a quality model as we do today. In each of the studied quality models, the ranking of packages and, hence, the interpretation highly depends on the normalization used. We argue that for any of the normalizations suggested not only the quality results (quality scores) but also their interpretation (ranking of packages and files with respect to these quality scores) is volatile. The quality trend, i.e., the change in the quality score observed for a complete system over different releases also highly depends on the normalization used. Two different normalizations can, for the same metrics and quality model, present contradicting trends; one suggests that the quality is constantly improving, the other that the quality is constantly deteriorating (e.g., cf. Table VIII columns $Normalize_{>80\%}$ and $NormDist_{>80\%}$).

The best fit approximation of a sample distribution suggests a mathematically sound normalization. However, we observe that about 50% of the metrics values do not pass a normality test after transformation. Further, the best fit distribution for similar metrics in different projects can be different, for example Lines of code where the best fit is Gamma in Subject B and Johnson in Subject C. Hence, sound normalizations are not easy to calculate in practice.

Our research question assume that we can find a proper normalization that provides an accurate assessment based on our understanding of the quality of the system under assessment. However, since we in many cases find that metrics values do not pass a normality test after transformation and that the best fit is volatile depending on Subject, release, etc., we find that the ground truth (i.e., the accurate quality ranking) is less important. The volatility of the ranking depending on normalizations and that we often cannot find a proper choice are major contributions of this paper.

We still need to investigate whether the distributions of individual metrics are stable across several releases of a system, similar systems from different companies, or even systems from different domains. It would be interesting to develop robust mathematical and software tools to reliably approximate best-fit distributions for samples of metrics values. In the long run, lab experiments using benchmark system where we have an understanding of the quality should determine the normalization methods that best correlate with our understanding of the quality of those systems. This should provide us with "the right normalization" for a specific system and metric.

Finally, we should consider alternatives to today's quality models for integrating and aggregating metrics values of entities to quality scores of systems. Bakota et al. [14] introduces a quality model based on probabilities. It would be interesting to compare the results of such a model to the result of the various normalizations.

References

[1] V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," in *Encyclopedia of Software Engineering*. Wiley, 1994.

[2] M. Fan, Y. Luo, G. Wu, and X. Fu, "An improved analytic hierarchy process model on software quality evaluation," in *Int Conf Information Science and Engineering*, dec. 2010, pp. 1838–1842.

[3] G. Baxter, M. Frean, J. Noble, M. Rickerby, H. Smith, M. Visser, H. Melton, and E. Tempero, "Understanding the shape of Java software," *SIGPLAN Not.*, vol. 41, no. 10, pp. 397–412, Oct. 2006.

[4] H. Barkmann, R. Lincke, and W. Löwe, "Quantitative evaluation of software quality metrics in open-source projects," in *Proc Works. Advanced Information Networking and Applications*, 2009, pp. 1067–1072.

[5] ISO/IEC, *ISO/IEC 9126. Software Engineering – Product Quality*. ISO/IEC, 2001.

[6] J. A. McCall, P. G. Richards, and G. F. Walters, "Factors in Software Quality," NTIS, NTIS Springfield, VA, Tech. Rep. Volume I, 1977.

[7] R. Lincke, J. Lundberg, and W. Löwe, "Comparing software metrics tools," in *Proc Int. Symp. Software Testing and Analysis*, 2008, pp. 131–142.

[8] C. Lewerentz and F. Simon, "A product metrics tool integrated into a software development environment," in *Proceedings of the Workshop on Object-Oriented Technology at ECOOP'98*, 1998.

[9] H. Zuse, *Software complexity: measures and methods*, ser. Programming complex systems, 1991.

[10] B. Kitchenham, S. L. Pfleeger, and N. Fenton, "Towards a framework for software measurement validation," *IEEE Trans. Softw. Eng.*, vol. 21, no. 12, pp. 929–944, 1995.

[11] D. E. Knuth, "An empirical study of FORTRAN programs," *Software: Practice and Experience*, vol. 1, no. 2, pp. 105–133, 1971.

[12] A. Laemmel and M. Shooman, *Software Modeling Studies, Volume II*, 1978.

[13] R. Wheeldon and S. Counsell, "Power law distributions in class relationships," in *IEEE Int'l Works. Source Code Analysis and Manipulation*, 2003, pp. 45–54.

[14] T. Bakota, P. Hegedus, P. Kortvelyesi, R. Ferenc, and T. Gyimóthy, "A probabilistic software quality model," in *ICSM*. IEEE, 2011, pp. 243–252.