Empir Software Eng (2017) 22:2585–2611
DOI 10.1007/s10664-017-9513-5

On the correlation between size and metric validity

Yossi Gil[1] · **Gal Lalouche**[1]

Published online: 17 June 2017

Abstract Empirical validation of code metrics has a long history of success. Many metrics have been shown to be good predictors of external features, such as correlation to bugs. Our study provides an alternative explanation to such validation, attributing it to the confounding effect of size. In contradiction to received wisdom, we argue that the validity of a metric can be explained by its correlation to the size of the code artifact. In fact, this work came about in view of our failure in the quest of finding a metric that is both valid and free of this confounding effect. Our main discovery is that, with the appropriate (non-parametric) transformations, the validity of a metric can be accurately (with R-squared values being at times as high as 0.97) predicted from its correlation with size. The reported results are with respect to a suite of 26 metrics, that includes the famous Chidamber and Kemerer metrics. Concretely, it is shown that the more a metric is correlated with size, the more able it is to predict external features values, and vice-versa. We consider two methods for controlling for size, by linear transformations. As it turns out, metrics controlled for size, tend to eliminate their predictive capabilities. We also show that the famous Chidamber and Kemerer metrics are no better than other metrics in our suite. Overall, our results suggest code size is the only "unique" valid metric.

Keywords Software engineering · Object-oriented programming

Communicated by: Richard Paige, Jordi Cabot and Neil Ernst

This research was supported by the *Israel Science Foundation* (ISF), grant No. 1803/13.

✉ Yossi Gil
Yogi@CS.Technion.AC.IL

Gal Lalouche
Lalouche@CS.Technion.AC.IL

[1] Department of Computer Science, The Technion–Israel Institute of Technology, Haifa, 3200003, Israel

1 Introduction

Internal software metrics or "code metrics", such as *Lines of Code* (henceforth, LOC), are concrete, well-defined, and easy to compute. In contrast, *external software metrics* (or features)—quality traits such as *maintainability*, *correctness*, and *robustness*—are more illusive, yet much more important (Meyer 1988, pp. 3–10).

A holy grail of the software engineering community is finding ties between the readily available and the useful, e.g., computing a predictor of maintainability from LOC. Indeed, classical metrics, such as McCabe's Cyclomatic Complexity (McCabe 1976) and Halstead's Software Science (Halstead 1977), have been frequently validated (Coleman et al. 1994; Khoshgoftaar and Munson 1990; Li and Cheung 1987).

In object-oriented design, the most widely cited suite is that of Chidamber & Kemerer (1994). The metric suite, suggested in 1994,[1] has been put to the test many times before. In 1996, Basili et al. (1996) were among the first to validate the suite. Since then, many have followed (Subramanyam and Krishnan 2003; Gyimothy et al. 2005; Olague et al. 2007).

Notwithstanding, it has also been shown that the traditional metrics are correlated with the size of the code (Shepperd 1988; Herraiz et al. 2007). It has been explicitly noted by El Emam et al. (2001) that correlation to size can "trick" us into thinking that some metrics are valid, when in fact they only appear so because of the aforementioned correlation to size. Somewhat more recently, Zhou et al. (2009) asserted that size has a confounding effect on the validity of metrics. (Both works included the Chidamber & Kemerer suite). We revisit this topic, studying the confounding effect of size on the validity of software code metrics in a more general setting.

Other than size, there are several metrics that have gained immense popularity in the literature. These include the C&K metrics, but also metrics such as McCabe's complexity. The confounding effect of size on these metrics has also been recognized (to the extent that it has been argued that LOC and MCCB are equivalent (Jbara and Feitelson 2014)). But, there were also successful attempts (e.g., Subramanyam and Krishnan (2003) and the list of references mentioned there) to control for size, i.e., to show that a metric has predictive value, even after normalizing for size.

Contrary to this common wisdom, our main discovery is that the more a metric is correlated with size, the more able it is to predict external features values, and vice-versa. Concretely, we demonstrate a linear relationship between two properties of metrics (computed with appropriate (non-parametric) transformations):

- the level of correlation of the metric with size, and,
- the quality of the metric as a predictor of external features,

Linear correlation values, i.e., R^2, were at times as high as 0.97. Henceforth, we shall refer to this phenomenon as the "Linearity of Meta-Correlation", or LMC for short.

LMC is demonstrated across these dimensions of generality:

- *Variety of metrics.* The linear relationship is demonstrated over a variety of metrics, ranging from well known to the famous. In total, we have 26 metrics, and by subjecting these to two distinct size normalization teqchniques, the number of metrics in our pool increases to 76.

[1] A full year before JAVA (Arnold and Gosling 1996) even came out.

– *Variety of projects and minimization of selection bias.* The empirical analysis is carried over a dataset of 26 projects. The actual analysis, and the selection of projects were completely independent.
– *Invariance across methods of analysis.* LMC is demonstrated across two distinct methods for estimating correlation: Spearman and Kendall, and for three distinct external features.

Since the 26 metrics suite included, among others, Chidamber and Kemerer metrics and McCabe's cyclomatic complexity, LMC casts doubts on previous results on metrics validation.

LMC associates an *"**I**ntrinsic **P**redictive **P**ower"* value with any given metric μ, $\mathrm{IPP}(\mu)$. This is precisely the predictive power inferred from LMC and the correlation of metric μ with size.

For a metric μ to be of value, it should be shown that $\mathrm{P}(\mu)$, the actual predictive quality of μ, is significantly greater than $\mathrm{IPP}(\mu)$. In Section 7, we show no such advantage exists in the well studied metrics of Chidamber and Kemerer: These metrics fare no better than metrics such as naive "Entropy" and even red herrings such as "Euler's Totient Function", at least in the sense of the difference

$$d(\mu) = \mathrm{P}(\mu) - \mathrm{IPP}(\mu). \tag{1}$$

The large number of projects in the study, combined with the reduction of selection bias, raises yet another concern. Consider again the quantity $d(\mu)$. In our study, this quantity is averaged across all projects. In Section 7 we show that outliers exist when metrics are observed in each corpus individually. In other words, it is possible to find a combination of metric, feature, and project such that $d(\mu)$ is significant.

But, the significance of such finding is limited: all it tells that in an individual project, μ has predictive qualities beyond the LMC phenomenon. Such a conclusion may be very useful for this specific project, but it does not permit extrapolation to other projects. Worse, it could be the case that the natural selection in academic research preferring positive results over negative results, i.e., publishing bias, may lead the community as a whole to focus on metrics and on projects for which $d(\mu)$ is significant. The general LMC trend, which predicts that $d(\mu)$ is small in the majority of cases may be missed.

By employing linear regression we show that the validity of a metric can be accurately, e.g., $R^2 = 0.97$, predicted from its correlation with size. In terms of this predictability, the C&K metrics do not stand out in the crowd of other metrics.

Is it possible to control for the confounding effect of size by some sort of normalization? Our findings show that this tight linear fit is preserved even for "normalized" metrics, where normalization is either linear (dividing the metric value by size), or rank based (similar to linear normalization except that it uses relative rank rather than absolute value).

Outline Section 2 explains how our dataset was assembled. The suite of metrics is the subject of Section 3. In Section 4 we study correlation among metrics and describe techniques for eliminating the dependency of metrics on size. The predictive power of metrics with respect to features is studied in Section 6. Section 7 presents our main result, by which there is a simple linear relationship between the validity of a metric and its correlation with size. Threats to validity are the subject of Section 8, and Section 9 concludes with directions for further research.

2 Dataset

The *dataset* used in this study comprises twenty-six corpora. The term *corpus* here refers to an open-source JAVA software artifact: an applications, a library, or a framework. But, the corpus does not include the code only; it is augmented with its evolution history, as retrieved from the version control system in which the artifact is managed. Corpora are denoted by capital letters of the Latin alphabet, $A, \ldots, Z$, assigned in order of descending size.

Overall, the dataset represents over 53,000 JAVA source files;[2] the median number of files in a project is $\approx$ 800, the minimum is $\approx$ 300. Number of software developers involved in a project ranged between 16 and 197; the total being circa two thousand. The development time frames sampled in the study ranged from one year to 13 years. The average time frame is about five years. Altogether, our study covers over a century of development history.

The process for assembling the dataset tried to eliminate niché projects, and identify profiles common to projects that make the "top lists" of open source repositories. Key criteria were *public availability and reproducibility* (hence insisting on open-source available on publicly accessible repositories), *programming language uniformity* (primary language being JAVA), *longevity*, (at least a year of development history) *community of developers* (at least ten authors), *non-meager development effort* (at least 100 files added on course of the history we traced, and at least 300 commit operations were performed), and *reasonable recency* (most recent change being no older than 15 months). The names and versions of the datasets used are summarized in Table 1.

To minimize selection bias, artifacts were selected semi-automatically[3] JAVA projects in *GitHub's Trending repositories*[4] and the list provided by the *GitHub* JAVA *Corpus* due to Allamanis and Charles (2013),[5] and retrieved from either GitHub[6] and Google Code[7] classification in each corpus. The procedure of classifying bugs is discussed in length in Section 5.3. Interestingly, the percentage themselves are normally distributed (as measured using the Shapiro-Wilk normality test), after the exclusion of two outliers—corpora *O* and *P*. (Corpus *O* has a very large amount of new files, and a very low amount of bug reports; corpus *P* has a lot of bug fixes.).

This is notable since it goes to show that the selected projects themselves show similar properties with regards to this feature. As evident on Table 2, using this method can sometimes cause new files—files with no previous version—to be considered bug fixes, especially in the outlier P (note the penultimate column).

3 Software metrics suite

Hundreds of code metrics are discussed in the literature (Lorenz and Kidd 1994; Henderson-Sellers 1996; Fenton and Bieman 2014). We restrict ourselves to module level metrics,

[2] Non-JAVA files, though found in many of the projects, were ignored.

[3] We used the first to download from the top members of both lists.

[4] https://github.com/trending?l=java&since=monthly.

[5] http://groups.inf.ed.ac.uk/cup/javaGithub/.

[6] http://github.com/.

[7] https://code.google.com/.

Table 1 Software projects constituting the dataset (in descending number of commits)

Id	Project	First Version	Last Version	#Days	#Authors	Last Commit ID
A	wildfly	’10-06-08	’14-04-22	1413	194	5a29c860
B	hibernate-orm	’09-07-07	’14-07-02	1821	150	b0a2ae9d
C	hadoop-common	’11-08-25	’14-08-21	1092	69	0c648ba0
D	jclouds	’09-04-28	’14-04-25	1823	100	f1a0370b
E	elasticsearch	’11-10-31	’14-06-20	963	129	812972ab
F	hazelcast	’09-07-21	’14-07-05	1809	65	3c4bc794
G	spring-framework	’10-10-25	’14-01-28	1190	53	c5f908b1
H	hbase	’12-05-26	’14-01-30	613	25	c67682c8
I	netty	’11-12-28	’14-01-28	762	72	3061b154
J	voldemort	’01-01-01	’14-04-28	4865	56	fb3203f3
K	guava	’11-04-15	’14-02-25	1047	12	6fdaf506
L	openmrs-core	’10-08-16	’14-06-18	1401	119	05292d98
M	CraftBukkit	’11-01-01	’14-04-23	1208	156	62ca8158
N	Essentials	’11-03-19	’14-04-27	1134	67	229ff9f0
O	docx4j	’12-05-12	’14-07-04	783	19	8edaddfa
P	atmosphere	’10-04-30	’14-04-28	1459	62	557e1044
Q	k-9	’08-10-28	’14-05-04	2014	81	95f33c38
R	mongo-java-driver	’09-01-08	’14-06-16	1984	75	5565c46e
S	lombok	’09-10-14	’14-07-01	1721	22	3c4f6841
T	RxJava	’13-01-23	’14-04-25	456	47	12723ef0
U	titan	’13-01-04	’14-04-17	468	17	55de01a3
V	hector	’10-12-05	’14-05-28	1270	95	f2fc542c
W	junit	’07-12-07	’14-05-03	2338	91	56a03468
X	cucumber-jvm	’11-06-27	’14-07-22	1120	93	fd764318
Y	guice	’07-12-19	’14-07-01	2386	17	76be88e8
Z	jna	’11-06-22	’14-07-07	1110	46	a5942aaf
			Total	38250	1932	
			Average	1471	74	
			Median	1239	68	

We trace over a century of development and follow the behavior of almost 2,000 software engineers. The last column is added for reproducibility

i.e., metrics pertaining to individual classes (rather than to an entire project, packages or individual methods).

The rationale for focusing on module level metrics is a bit technical: With git and other version control systems, the association of bugs and changes to modules which occupy an entire file is natural. It is possible to conduct a finer grained analysis to make this association at the method level, but, doing so requires also investing effort in defining when such association should be made; this effort seems pointless in dealing, as we do here, with object oriented programs in which classes, not methods, are the primary modular unit.

Table 2 The distribution of buggy files, as classified using commit message parsing, for corpora in the dataset

Id	#File	Changes			New files		
	Commits	Total	Bugs	Other	Total	Bugs	Other
A	39910	31536 (79%)	4078 (13%)	27458 (87%)	8374 (21%)	380 (5%)	7994 (95%)
B	30677	23062 (75%)	964 (4%)	22098 (96%)	7615 (25%)	153 (2%)	7462 (98%)
C	31079	25391 (82%)	3462 (14%)	21929 (86%)	5688 (18%)	330 (6%)	5358 (94%)
D	29094	24439 (84%)	1362 (6%)	23077 (94%)	4655 (16%)	98 (2%)	4557 (98%)
E	27685	23921 (86%)	4301 (18%)	19620 (82%)	3764 (14%)	49 (1%)	3715 (99%)
F	22917	20487 (89%)	4939 (24%)	15548 (76%)	2430 (11%)	297 (12%)	2133 (88%)
G	22816	17411 (76%)	3062 (18%)	14349 (82%)	5405 (24%)	18 (0%)	5387 (100%)
H	12908	10834 (84%)	1051 (10%)	9783 (90%)	2074 (16%)	34 (2%)	2040 (98%)
I	11068	10006 (90%)	1293 (13%)	8713 (87%)	1062 (10%)	48 (5%)	1014 (95%)
J	9776	8822 (90%)	1309 (15%)	7513 (85%)	954 (10%)	52 (5%)	902 (95%)
K	9313	7648 (82%)	978 (13%)	6670 (87%)	1665 (18%)	34 (2%)	1631 (98%)
L	8559	7064 (83%)	836 (12%)	6228 (88%)	1495 (17%)	33 (2%)	1462 (98%)
M	7648	7107 (93%)	1477 (21%)	5630 (79%)	541 (7%)	42 (8%)	499 (92%)
N	7018	6651 (95%)	1140 (17%)	5511 (83%)	367 (5%)	10 (3%)	357 (97%)
O	6452	3676 (57%)	188 (5%)	3488 (95%)	2776 (43%)	8 (0%)	2768 (100%)
P	5938	5603 (94%)	2359 (42%)	3244 (58%)	335 (6%)	145 (43%)	190 (57%)
Q	5565	5218 (94%)	817 (16%)	4401 (84%)	347 (6%)	5 (1%)	342 (99%)
R	4360	4001 (92%)	450 (11%)	3551 (89%)	359 (8%)	42 (12%)	317 (88%)
S	3672	2970 (81%)	635 (21%)	2335 (79%)	702 (19%)	112 (16%)	590 (84%)
T	3626	3176 (88%)	192 (6%)	2984 (94%)	450 (12%)	20 (4%)	430 (96%)
U	3242	2708 (84%)	535 (20%)	2173 (80%)	534 (16%)	52 (10%)	482 (90%)
V	3094	2635 (85%)	241 (9%)	2394 (91%)	459 (15%)	26 (6%)	433 (94%)
W	2800	2414 (86%)	397 (16%)	2017 (84%)	386 (14%)	24 (6%)	362 (94%)
X	2645	2183 (83%)	214 (10%)	1969 (90%)	462 (17%)	23 (5%)	439 (95%)
Y	2283	1760 (77%)	588 (33%)	1172 (67%)	523 (23%)	35 (7%)	488 (93%)
Z	2003	1700 (85%)	426 (25%)	1274 (75%)	303 (15%)	5 (2%)	298 (98%)
Total	316148	262423	37294	225129	53725	2075	51650
	Average	84	15	84	15	6	93
	σ	7	8	8	7	8	8
	Median	84	14	85	15	5	95
	MAD	4	4	4	4	3	3
	Min	57	4	58	5	0	57
	Max	95	42	96	43	43	100

Our suite includes a sample of the most common module metrics, some variants of these, a number of newly introduced metrics, and even some nonsensical ones:

– *The Chidamber & Kemerer metrics* (Chidamber and Kemerer 1994) CBO, LCOM, DIT, NOC, RFC and WMC. The variant used for weighting methods for WMC is a count of the number of the tokens present in a methods. Metric NOM (***N****umber* ***o****f* ***M****ethods*) represents another variant of WMC, in which all methods receive the weight of 1.

- *Class metrics:* In addition to NOM, we employ NIV (***N**umber **o**f **I**nstance **V**ariables*), MTE (***M**utability* (***E**xplicit*)–number of non-**`final`** instance fields), and CON (***C**onstructor **C**ount*). The CHAM (***Cham**eleonicity*) metric we introduce here also belongs in this group; it is defined as the maximum, over all methods, of the number of polymorphic variables accessed by the method.[8]
- *Size metrics:* LOC (**L**ines **o**f **C**ode), as well as NOT (***N**umber **o**f **T**okens*) and NOS (***N**umber **o**f **S**tatements*).[9]
- *Comment metrics:* Comment lines are counted in LOC, but the suite include two more specific metrics: ABC (***A**lpha betical Comment Character Count*) and CCC (***C**omment **C**haracters **C**ount*). Both are computed on all line-, block-, and JAVAdoc- comments, with the exclusion of comments occurring before the **`package`** declaration, as these are typically boilerplate. The reason for excluding of non-alphabetical characters (punctuation and the such) in ABC is the practice, witnessed in our dataset, of using these for embellishing comments.
- *Regularity metrics:* It was suggested (e.g., by Jbara and Feitelson (2014)) that regularity is a better predictor of maintainability than size. The suite therefore includes metric LZW (*Regularity (**L**empel-**Z**iv-**W**elch compression)*), defined as the number of compressed tokens after applying the LZW compression algorithm (Ziv and Lempel 1978) to the tokens of a class. Another regularity metric is GZIP (*Regularity (**GZIP** compression)*), the number of bytes in the GZip (Deutsch 1996) compressed byte-stream of the class's source code. A third variant is ENT (**Ent**ropy), the information-theoretical entropy value of the multi-set of tokens of a class's source code.
- *Flow complexity metrics:* First of these is MCC, the famous (McCabe 1976) Cyclomatic Complexity. Metric MCCS (***M**cCabe **C**yclomatic **C**omplexity (simple)*) is a variant of MCC that excludes instances of short-circuit evaluation since these may be attributed to language semantics rather than to actual complexity. Two newly introduced metrics are also present in this group: LPC (***L**oop **C**ount*) is the number of iteration commands (such as **`for`**), and HOR (***Hor**izontal complexity*), which is similar to NOS except that in the statements count, each statement is weighted by the number of control structures (**`if`**, **`while`**, etc.) that surround it.[10]
- *Nonsensical metrics:* The weirdest of our suite is ETF (***E**uler's **T**otient **F**unction*), defined as value of φ, Euler's totient function,[11] applied to the sum of the Unicode numerical values of characters the code. In addition, this group includes MNK (***Monk**ey*) which assigns a class a random, uniformly distributed value in [0, 1] and its discrete and deterministic counterpart SHA (*SHA-1*), ranging in [0, 255] and defined as the last byte of the SHA-1(Eastlake and Jones 2001) value of the class source code.

The metric values may obviously change in the course of the lifetime of a class. We consider each class as a single entity that may be manifested in multiple versions. Accordingly, the metric value associated with the class is the arithmetical average of the metric

[8]The rationale is that CHAM is the best approximation (within the limits of the depth of our code analysis) of the "Chameleonicity" (Gil et al. 1998) metric, which was shown to be linked to the algorithmic complexity of analyzing virtual function calls.

[9]both are based on a syntactical code analysis: tokens are the language lexical tokens while the term "statements" refers to the respective production in the language EBNF, except that bracketed blocks are ignored.

[10]Similar metrics are mentioned in the literature, e.g., Harrison and Magel (1981) suggested "a complexity measure based on nesting level". See also the work of Piwowarski (1982) and that of Hindle et al. (2008).

[11]The number of positive integers smaller than a given number which are relatively prime to it.

value in all its versions. An anecdotal consequence of this perspective is that the metrics MNK and SHA which seem to be useless, do reflect useful information. As the number of versions increases, the value of MNK will tend to 0.5 and that of SHA will tend to 127.5.

For the exact definition of each metric we used, our source code has been uploaded to `git`.[12]

4 Independence of metrics

We anticipate the metrics in our suite to be correlated, because the definitions of some of them (e.g., ABC and CCC) are similar. Moreover, it is well known that even seemingly different software metrics tend to be correlated (Herraiz et al. 2007; Shepperd 1988).

In this section we study the extent of this correlation and then present two techniques for controlling for the correlation with size.

4.1 Kendall rank correlation coefficient

Kendall τ rank correlation coefficient is employed, here and henceforth, rather than the more familiar *Pearson r correlation* coefficient. The first reason for this is that Kendall correlation is non-parametric. Indeed, it is quite unnatural to characterize as normally distributed many metrics in our suite (think, e.g., of DIT and LCOM). The second reason has to do with outliers, whose existence in software is common knowledge (see, e.g., Alan and Catal (2011)). As it turns out, Pearson correlation is sensitive to outliers in long tailed distributions which typify (Baxter et al. 2006) software metrics. A handful of outliers in the high end of the metric's range are capable of completely invalidating Pearson correlation.

Note that Kendall τ values offer intuitive interpretation which we may describe as follows: Suppose that classes c_1 and c_2 are selected at random, and suppose that the number of lines of code in c_1 is greater than of that of c_2, i.e, $\mathsf{LOC}(c_1) > \mathsf{LOC}(c_2)$ (ties are ignored in this description). Then, what can be inferred about the number of non-`final` instance fields found in c_1 as compared to that of c_2?

More specifically, we may ask how likely is the *concordant* prediction

$$\mathcal{P}^{\uparrow} = \mathsf{MTE}(c_1) > \mathsf{MTE}(c_2). \tag{2}$$

The value of Kendall correlation $\tau(\mathsf{LOC}, \mathsf{MTE})$ gives a precise answer to this question. If

$$\tau(\mathsf{LOC}, \mathsf{MTE}) = 1,$$

then the concordant prediction (2) is certain. Conversely, if

$$\tau(\mathsf{LOC}, \mathsf{MTE}) = -1,$$

the *disconcordant* prediction

$$\mathcal{P}^{\downarrow} = \mathsf{MTE}(c_1) < \mathsf{MTE}(c_2) \tag{3}$$

is certain. On the other hand, if Kendall's coefficient is unsigned, i.e., $\tau(\mathsf{LOC}, \mathsf{MTE}) = 0$, then both predictions are equally likely. Finally, if, as it is really the case,

$$\tau(\mathsf{LOC}, \mathsf{MTE}) = 0.35, \tag{4}$$

[12] https://github.com/GalLalouche/validity_independence.

then the concordant prediction has 35 points advantage over the disconcordant prediction:

$$Pr[P^{\uparrow}] - Pr[P^{\downarrow}] = 0.35. \quad (5)$$

We prefer Kendall correlation over Spearman correlation (the non-parametric variant of Pearson correlation) since the latter does not offer a similar simple intuitive interpretation.

4.2 Correlation between the metrics

Table 3 presents the values of Kendall rank correlation coefficient in all metric pairs.

Let μ_1 and μ_2 be metrics. Then, the correlation coefficient $\tau(\mu_1, \mu_2)$ presented in the table is the average of $\tau_A(\mu_1, \mu_2)$, $\tau_B(\mu_1, \mu_2)$, ..., $\tau_Z(\mu_1, \mu_2)$, i.e., the correlation coefficients of μ_1 and μ_2 in the corpora that make the dataset. (To avoid bias towards larger corpora, the plain mean is employed, rather than its size weighted counterpart.)

Table 3 employs shading as a visualization aide: lighter cells represent lower τ values. This shading makes it quite noticeable that positive and high (e.g., $\tau \geq 0.35$) pair-wise correlation values make the rule rather than the exception.

We also see that the majority of metrics are positively correlated with NOT. Under the somewhat arbitrary threshold of $\tau \geq 0.30$ correlation we find only five metrics: The two "random" metrics, MNK and SHA (which, as expected, are completely independent of NOT), two C&K metrics DIT ($\tau = -0.06$) and NOC ($\tau = 0.02$), followed by CON ($\tau = 0.14$).

Hence, this section highlights these three as prime candidates of metrics which are free from the confounding effect of size. It remains to be seen whether after normalizing for the predictive power of DIT, NOC, and CON is any better than that of MNK or SHA. More generally, we need to check whether predictive power found in any of the other metrics remains despite controlling for size.

Metric NOT shall henceforth serve as our main code size metric. The choice is somewhat arbitrary, as the majority of metrics are positively correlated with it, e.g., LOC and GZIP. Robustness and simplicity are the reasons for the (slight) preference of NOT over other candidates.

4.3 Linear normalization

The usual antidote against bias and confounding effect is normalization. A country's GDP is dependent on its size, so it is customary to compare the GDP per capita. Similarly, in assessing the traffic-related death rate, it is typical to normalize per capita, per vehicle, and per km-vehicle (total mileage driven by all vehicles).

Drawing inspiration we introduce linear normalization which simply divides the value of a metric in a given module by the NOT of this module. Rank based normalization is presented below. Other linear normalizations using other metrics will be employed as well.

We will see here that normalization for size is not so simple with software metrics: normalization may introduce confounding effect in cases it does not exist; it may also just change the level of the effect when it is present, and, in some cases, it may nullify it.

Our main result will however show that the basic rule that validity is almost a linear function of correlation with size remains also for the normalized metrics. In other words, it is possible to normalize for size, either linearly (as done here), or by rank (Section 4.4), and sometimes neutralize the effect of size, however, the more independent the metric is of size, the less valid it is.

Table 3 Kendall τ correlation values (in points) for all pairs of metrics in the suite

Metric	ABC3	CBO	CCC	CHAM	CON	DIT	ENT	ETF	GZIP	HOR	LCOM	LOC	LPC	LZW	MCC	MCCS	MNK	MTE	NIV	NOC	NOM	NOS	NOT	RFC	SHA1	WMC
ABC3	100	44	98	47	21	-10	57	54	62	48	41	60	39	56	47	48	0	27	36	7	47	50	54	49	0	49
CBO	44	100	44	69	26	-2	64	60	68	64	46	65	51	70	57	58	0	30	42	4	55	65	70	69	0	66
CCC	98	44	100	47	21	-10	56	54	62	49	41	60	39	56	47	47	0	27	36	7	47	50	54	49	0	49
CHAM	47	69	47	100	31	-5	60	59	64	62	52	66	50	67	58	58	0	28	40	8	63	64	67	67	0	63
CON	21	26	21	31	100	6	27	24	26	25	29	28	20	27	26	26	0	25	34	5	27	27	26	27	0	22
DIT	-10	-2	-10	-5	6	100	-9	-9	-9	-9	-6	-10	-9	-9	-10	-10	0	-10	-14	1	-12	-9	-9	-7	0	-8
ENT	57	64	56	60	27	-9	100	62	76	66	50	70	53	74	64	64	0	35	50	6	57	67	70	67	0	65
ETF	54	60	54	59	24	-9	62	100	73	66	50	74	48	73	59	59	0	33	45	6	60	69	73	67	0	68
GZIP	62	68	62	64	26	-9	76	73	100	75	53	86	54	89	68	68	0	36	50	7	63	78	85	77	0	77
HOR	48	64	49	62	25	-9	66	66	75	100	49	76	59	80	79	79	0	35	47	5	60	89	80	77	0	84
LCOM	41	46	41	52	29	-6	50	50	53	49	100	59	36	55	44	45	0	38	43	10	74	54	56	59	0	51
LOC	60	65	60	66	28	-10	70	74	86	76	59	100	53	88	67	67	0	40	53	7	73	82	88	79	0	78
LPC	39	51	39	50	20	-9	53	48	54	59	36	53	100	56	63	64	0	27	37	3	43	58	55	56	0	58
LZW	56	70	56	67	27	-9	74	73	89	80	55	88	56	100	71	71	0	37	51	6	67	85	94	82	0	84
MCC	47	57	47	58	26	-10	64	59	68	79	44	67	63	71	100	96	0	33	45	5	53	76	69	68	0	74
MCCS	48	58	47	58	26	-10	64	59	68	79	45	67	64	71	96	100	0	33	45	5	53	77	70	68	0	74
MNK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0
MTE	27	30	27	28	25	-10	35	33	36	35	38	40	27	37	33	33	0	100	64	6	42	38	37	38	0	35
NIV	36	42	36	40	34	-14	50	45	50	47	43	53	37	51	45	45	0	64	100	4	50	50	51	50	0	45
NOC	7	4	7	8	5	1	6	6	7	5	10	7	3	6	5	5	0	6	4	100	10	5	6	6	0	5
NOM	47	55	47	63	27	-12	57	60	63	60	74	73	43	67	53	53	0	42	50	10	100	67	69	71	0	64
NOS	50	65	50	64	27	-9	67	69	78	89	54	82	58	85	76	77	0	38	50	5	67	100	86	82	0	87
NOT	54	70	54	67	26	-9	70	73	85	80	56	88	55	94	69	70	0	37	51	6	69	86	100	82	0	85
RFC	49	69	49	67	27	-7	67	67	77	77	59	79	56	82	68	68	0	38	50	6	71	82	82	100	0	80
SHA1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0
WMC	49	66	49	63	22	-8	65	68	77	84	51	78	58	84	74	74	0	35	45	5	64	87	85	80	0	100

Linear normalization of a metric μ (with respect to NOT) is defined by the fraction

$$\mu_L = \frac{\mu}{\mathsf{NOT}}. \tag{6}$$

For many of the metrics our suite, linear normalization has an intuitive interpretation, e.g., NOS_L is the average number of tokens per statements. However, thinking of e.g., DIT_L makes one realize that this is not always the case.

Table 4 repeats Table 3 for the τ values among the linearly normalized metrics, except that NOT is used instead of NOT_L (since, by definition, $\mathsf{NOT}_L \equiv 1$).

We see that the shades in Table 4 are much lighter than those of Table 3. Indeed, linear normalization decouples some metrics from size, e.g., MCC and NOS. It fails in doing so for other metrics, e.g., ENT and GZIP. The reason, which is yet another demonstration that metrics should be considered as non-normally distributed and that Pearson correlation should be avoided when possible, is that correlation of metrics with size is not necessarily linear. For example, the inverse correlation of GZIP_L with NOT is merely a reflection of the fact that compression ratio tends to be better for larger files.

4.4 Rank normalization

Rank normalization is designed to address these issues of linear normalization. For each metric μ, let μ_R, the rank normalized version of μ, be defined by:

$$\mu_R = \frac{\text{rank}\,(\mu)}{\text{rank}\,(\mathsf{NOT})}, \tag{7}$$

where function rank$(\cdot)$ maps first a value to its ordinal rank among its cohorts in the same corpus and then rescales this ordinal to the range $[0, 1] \subset \mathbb{R}$.

Table 5 repeats Table 4 for the rank normalized version of metrics.

Evidently, shades in Table 5 are even lighter than those of Table 4.

Still, the fine print at the NOT row (or column) shows that large correlation values, e.g., $|\tau| > 0.5$ exist. There are two explanations to this phenomena: First, applying normalization with respect to size to metrics which are initially uncorrelated with size (e.g., MNK), introduces such correlation through the back door. Second, normalizing by size introduces a dependency on size for metrics such as DIT and LPC, in which a small number of metric values is repeatedly assumed by many data points.

Full discussion of this phenomena is beyond the scope of this work. Here we will simply add the linear- and rank- normalized versions of metrics to our basket of metrics, regardless of whether these versions are correlated with size.

5 External features

This section defines the three external features used in this study. The names we use for these, “Instability”, “Change complexity” and “Bugginess”, are not to be interpreted at absolute face value. They are rather a reminder of the abstract properties this features’ trio approximates.

5.1 Instability

Our instability code feature relates to the frequency of changes made to a file—the higher the instability, the higher this frequency. However, instead of employing time units, we

Table 4 Kendall τ correlation values (in points) for metrics after linear normalization

Metric	ABC_L	CBO_L	CCC_L	$CHAM_L$	CON_L	DIT_L	ENT_L	ETF_L	$GZIP_L$	HOR_L	$LCOM_L$	LOC_L	LPC_L	LZW_L	MCC_L	$MCCS_L$	MNK_L	MTE_L	NIV_L	NOC_L	NOM_L	NOS_L	NOT	RFC_L	SHA_L	WMC_L
ABC_L	100	-12	97	4	-3	-14	-12	15	13	1	10	26	3	-3	0	1	-10	2	0	5	1	-3	-13	-10	-4	13
CBO_L	-12	100	-12	20	16	34	34	2	29	-17	-20	2	-11	35	-12	-14	25	-12	-3	-3	-5	-24	14	25	-24	-34
CCC_L	97	-12	100	4	-3	-14	-12	15	12	1	11	26	2	-3	0	1	-10	2	0	5	1	-3	-13	-10	-4	13
$CHAM_L$	4	20	4	100	11	-5	-6	2	-6	0	13	0	5	-6	-4	-4	-6	-8	-9	4	11	-4	4	-6	-4	7
CON_L	-3	16	-3	11	100	26	27	2	19	-11	-1	17	-9	23	-2	-3	17	2	16	0	4	-7	10	17	-26	-27
DIT_L	-14	34	-14	-5	26	100	81	5	56	-37	-33	28	-30	68	-9	-12	55	-9	1	-2	2	-35	10	55	-42	-82
ENT_L	-12	34	-12	-6	27	81	100	7	63	-39	-35	32	-30	78	-8	-11	60	-8	4	-3	4	-37	9	60	-47	-97
ETF_L	15	2	15	2	2	5	7	100	16	-10	-1	18	-8	9	-6	-6	5	-2	0	1	1	-11	-5	4	-13	-6
$GZIP_L$	13	29	12	-6	19	56	63	16	100	-32	-27	44	-24	75	-5	-7	44	-8	5	-1	-1	-34	5	44	-43	-61
HOR_L	1	-17	1	0	-11	-37	-39	-10	-32	100	9	-26	38	-34	45	46	-30	7	-1	0	-15	70	6	-30	58	39
$LCOM_L$	10	-20	11	13	-1	-33	-35	-1	-27	9	100	3	9	-34	-2	-1	-27	19	7	10	39	16	9	-27	8	35
LOC_L	26	2	26	0	17	28	32	18	44	-26	3	100	-24	35	-7	-9	24	9	14	7	30	-19	1	24	-43	-32
LPC_L	3	-11	2	5	-9	-30	-30	-8	-24	38	9	-24	100	-25	33	32	-25	0	-5	0	-14	32	3	-25	36	31
LZW_L	-3	35	-3	-6	23	68	78	9	75	-34	-34	35	-25	100	-4	-6	53	-7	7	-3	0	-35	11	53	-44	-76
MCC_L	0	-12	0	-4	-2	-9	-8	-6	-5	45	-2	-7	33	-4	100	89	-7	0	-2	1	-14	39	1	-7	29	9
$MCCS_L$	1	-14	1	-4	-3	-12	-11	-6	-7	46	-1	-9	32	-6	89	100	-9	1	-1	2	-14	39	1	-9	31	11
MNK_L	-10	25	-10	-6	17	55	60	5	44	-30	-27	24	-25	53	-7	-9	100	-7	3	-3	3	-28	7	44	-34	-60
MTE_L	2	-12	2	-8	2	-9	-8	-2	-8	7	19	9	0	-7	0	1	-7	100	50	3	20	14	7	-7	1	8
NIV_L	0	-3	0	-9	16	1	4	0	5	-1	7	14	-5	7	-2	-1	3	50	100	0	15	4	9	3	-15	-4
NOC_L	5	-3	5	4	0	-2	-3	1	-1	0	10	7	0	-3	1	2	-3	3	0	100	9	1	0	-2	-1	3
NOM_L	1	-5	1	11	4	2	4	1	-1	-15	39	30	-14	0	-14	-14	3	20	15	9	100	-3	25	3	-18	-5
NOS_L	-3	-24	-3	-4	-7	-35	-37	-11	-34	70	16	-19	32	-35	39	39	-28	14	4	1	-3	100	11	-28	52	37
NOT	-13	14	-13	4	10	10	9	-5	5	6	9	1	3	11	1	1	7	7	9	0	25	11	100	7	2	-9
RFC_L	-10	25	-10	-6	17	55	60	4	44	-30	-27	24	-25	53	-7	-9	44	-7	3	-2	3	-28	7	100	-34	-60
SHA_L	-4	-24	-4	-4	-26	-42	-47	-13	-43	58	8	-43	36	-44	29	31	-34	1	-15	-1	-18	52	2	-34	100	46
WMC_L	13	-34	13	7	-27	-82	-97	-6	-61	39	35	-32	31	-76	9	11	-60	8	-4	3	-5	37	-9	-60	46	100

Table 5 Kendall τ correlation values (in points) for metrics with rank normalization

Metric	ABC_R	CBO_R	CCC_R	$CHAM_R$	CON_R	DIT_R	ENT_R	ETF_R	$GZIP_R$	HOR_R	$LCOM_R$	LOC_R	LPC_R	LZW_R	MCC_R	$MCCS_R$	MNK_R	MTE_R	NIV_R	NOC_R	NOM_R	NOS_R	NOT	RFC_R	SHA_R	WMC_R
ABC_R	100	7	97	16	23	23	29	17	44	3	20	35	21	23	16	16	21	23	17	29	12	0	0	21	-2	-29
CBO_R	7	100	7	35	21	25	22	5	10	3	11	-9	18	16	7	6	17	14	11	22	1	-4	14	16	1	-24
CCC_R	97	7	100	16	23	23	29	17	43	3	20	35	21	22	16	16	21	23	17	29	13	0	0	20	-3	-29
$CHAM_R$	16	35	16	100	28	24	17	7	6	4	24	7	20	6	13	12	18	16	12	25	20	0	12	18	-1	-25
CON_R	23	21	23	28	100	44	19	9	9	9	32	14	30	8	22	22	31	38	37	47	19	11	15	31	0	-50
DIT_R	23	25	23	24	44	100	18	9	8	11	29	3	35	8	23	23	41	40	26	66	15	11	17	41	16	-73
ENT_R	29	22	29	17	19	18	100	9	43	11	15	17	21	58	21	21	14	17	21	20	6	5	9	14	2	-22
ETF_R	17	5	17	7	9	9	9	100	17	-1	8	15	6	9	3	3	8	8	7	11	5	-2	0	7	-4	-11
$GZIP_R$	44	10	43	6	9	8	43	17	100	3	5	30	10	54	12	12	6	8	10	9	-3	-3	2	6	-6	-9
HOR_R	3	3	3	4	9	11	11	-1	3	100	0	0	24	9	49	48	10	10	7	12	-4	67	17	10	46	-13
$LCOM_R$	20	11	20	24	32	29	15	8	5	0	100	23	19	2	10	9	23	34	26	33	52	6	23	23	-2	-33
LOC_R	35	-9	35	7	14	3	17	15	30	0	23	100	2	11	3	2	5	17	18	8	31	6	4	5	-19	-6
LPC_R	21	18	21	20	30	35	21	6	10	24	19	2	100	13	40	38	29	31	21	40	8	20	15	29	21	-43
LZW_R	23	16	22	6	8	8	58	9	54	9	2	11	13	100	16	16	5	8	10	9	-8	1	10	5	4	-9
MCC_R	16	7	16	13	22	23	21	3	12	49	10	3	40	16	100	90	20	20	17	26	0	40	12	20	34	-27
$MCCS_R$	16	6	16	12	22	23	21	3	12	48	9	2	38	16	90	100	19	20	17	26	0	39	12	20	34	-27
MNK_R	21	17	21	18	31	41	14	8	6	10	23	5	29	5	20	19	100	31	23	45	14	10	12	33	12	-49
MTE_R	23	14	23	16	38	40	17	8	8	10	34	17	31	8	20	20	31	100	54	46	28	14	17	32	7	-48
NIV_R	17	11	17	12	37	26	21	7	10	7	26	18	21	10	17	17	23	54	100	32	24	12	15	23	-5	-34
NOC_R	29	22	29	25	47	66	20	11	9	12	33	8	40	9	26	26	45	46	32	100	21	13	16	46	16	-86
NOM_R	12	1	13	20	19	15	6	5	-3	-4	52	31	8	-8	0	0	14	28	24	21	100	5	29	15	-3	-20
NOS_R	0	-4	0	0	11	11	5	-2	-3	67	6	6	20	1	40	39	10	14	12	13	5	100	22	10	40	-14
NOT	0	14	0	12	15	17	9	0	2	17	23	4	15	10	12	12	12	17	15	16	29	22	100	12	19	-17
RFC_R	21	16	20	18	31	41	14	7	6	10	23	5	29	5	20	20	33	32	23	46	15	10	12	100	12	-50
SHA_R	-2	1	-3	-1	0	16	2	-4	-6	46	-2	-19	21	4	34	34	12	7	-5	16	-3	40	19	12	100	-18
WMC_R	-29	-24	-29	-25	-50	-73	-22	-11	-9	-13	-33	-6	-43	-9	-27	-27	-49	-48	-34	-86	-20	-14	-17	-50	-18	100

associate an "internal clock" with each corpus. This clock ticks with every commit operation to the global repository. Instability is defined as the *median* number of clock ticks between two successive changes of a file. Note that each commit to the repository may change more than one file; we still treat this multi-file commits as a single tick.

Using the commits' clock rather than wall clock time is not only in adherence to the general principle of context-aware measurement. As it turns out, statistical significance of certain statistics improves (slightly) by switching from wall clock time to the commits clock. This improvement might be explained by the partial, sometimes even leisurely, engagement of many programmers to open source projects. Development effort is therefore more likely to be in bursts which may introduce bias to measurements that use wall clock time.

Also, the median is preferred over the average for the following reason: Assume there are two files, F_1 and F_2, such that, F_1 incurred a rapid succession of changes in a short time span followed by a period of inactivity, while F_2 incurred the same number of changes, but more evenly spaced out. While the average of both kinds of change observations would be similar, the median of the former will be smaller than that of the latter, which is preferable as we consider the former is less stable than the latter.

Variants of our instability metric were used for many years as a measure of software quality (Gillies 2011; Mohagheghi et al. 2004; Fayad and Altman 2001).

5.2 Change complexity

The "change complexity" feature is a measure of the amount of work programmers spend in a committed modification of a file. Specifically, let the *complexity of a commit* (of an individual file) be the number of line operations (as defined by `git`) conducted on the file. Then, the *change complexity* of a file is defined simply as the *average* commit complexity computed over all commits to this file.

We see that instability and change complexity are complementary: The former is (an indirect) measure of the number of steps in the evolution of a software module, while the latter is a measure of the size of each such step.

5.3 Bugginess

"Bugginess", our third and final feature, is defined as follows: A certain revision of a file is considered "buggy" if the comment of an overriding commit suggests that this commit was a bug fix. Specifically, the "buggy" flag is raised if this message contained the word "`bug`", or the word "`fix`", in the variety of its forms (`fixes`, `fixing`, etc.), or a sequence of digits which may be a legal bug identifier. The "bugginess" of a file is simply the number of times this flag was raised for any of its revisions.

Clearly, the bugginess of a file is only an approximation of the "true" number of bugs found in it. Indeed, as Bird et al. (2009) show, real bugs may be missed by this feature, and, likewise, this value may at times include in it false positives. Yet, this practical means for approximating the number of bugs is generally accepted by the community.

5.4 Statistics of features' distribution

Table 6 summarizes the main statistics of features values in the corpora that constitute our dataset, and provides some aggregate values of these.

Table 6 Statistics of features values in all corpora

Id	Instability				Change Complexity				Bugginess(×100)			
	Mean	Std	Median	MAD	Mean	Std	Median	MAD	Mean	Std	Median	MAD
A	16147	19707	6794	6107	8.2	14.6	2.4	2.4	0.5	1.2	0.0	0.0
B	8254	7741	6453	3061	14.1	20.8	9.2	8.5	0.1	0.4	0.0	0.0
C	16318	18913	7214	6598	9.8	16.4	3.0	3.0	0.6	1.7	0.0	0.0
D	3172	2072	3090	1610	13.3	10.2	11.6	2.4	0.3	0.8	0.0	0.0
E	5058	6022	2823	2063	13.4	66.2	8.0	3.0	1.1	0.9	1.0	0.0
F	2235	3338	1124	864	9.6	13.3	6.0	3.7	2.2	4.5	1.0	1.0
G	6742	9348	3489	1591	6.9	15.6	2.4	2.2	0.6	0.9	0.0	0.0
H	4281	5311	1787	1424	28.7	198.3	4.2	4.2	0.5	1.4	0.0	0.0
I	1610	2693	518	342	17.1	132.7	7.6	5.4	1.3	2.7	1.0	1.0
J	1474	2774	319	285	16.5	51.1	9.2	6.6	1.4	3.0	1.0	1.0
K	3107	3942	1554	1268	13.1	24.2	4.3	4.3	0.5	1.1	0.0	0.0
L	4534	5067	1864	1742	9.4	33.7	4.0	4.0	0.5	1.7	0.0	0.0
M	432	841	294	221	8.0	11.1	6.5	4.1	8.5	20.5	3.0	3.0
N	970	1773	326	191	13.5	20.3	8.1	5.8	2.7	6.5	1.0	1.0
O	572	1418	220	135	10.4	14.3	7.3	4.0	3.4	5.7	2.0	2.0
P	3950	3242	2885	1886	14.2	79.7	2.6	2.6	0.0	0.3	0.0	0.0
Q	1115	1226	501	480	15.3	27.5	9.5	9.5	2.5	7.2	0.0	0.0
R	284	183	299	166	10.1	10.4	6.3	4.3	1.3	3.3	0.0	0.0
S	1581	1736	904	800	6.3	10.6	3.0	3.0	0.9	2.3	0.0	0.0
T	482	770	224	140	12.0	18.2	6.4	6.4	0.4	1.6	0.0	0.0
U	898	1220	410	292	10.0	14.4	4.5	4.5	1.1	2.5	0.0	0.0
V	498	427	495	294	6.1	8.7	3.0	3.0	0.6	1.1	0.0	0.0
W	623	602	374	348	8.6	11.7	4.8	4.3	1.0	1.9	0.0	0.0
X	783	850	400	360	4.8	8.0	1.0	1.0	0.5	1.1	0.0	0.0
Y	807	1043	409	289	9.9	19.2	4.7	4.2	1.3	1.4	1.0	1.0
Z	1094	1383	319	293	31.7	347.1	4.0	4.0	1.3	3.1	1.0	1.0
Total	87021	103642	45089	32850	321	1198	143	110	35	78	12	11
Average	3346	3986	1734	1263	12	46	5	4	1	3	0	0
STD	4243	4975	2082	1650	6	73	2	1	1	3	0	0
Median	1527	1922	509	420	10	17	4	4	0	1	0	0
MAD	1037	1236	287	282	3	6	1	1	0	0	0	0
Min	284	183	220	135	4	8	1	1	0	0	0	0
Max	16318	19707	7214	6598	31	347	11	9	8	20	3	3

The main thing to notice in Table 6, is not the concrete values in it, but rather the considerable variety of values of each of the features we study. There are two aspects to this variety:

First, feature values of files within the same corpus are spread out. To see this, compare the standard deviation to the mean. In almost all cases, the standard deviation is almost always greater than the mean; in some of the cases it is even two or more times greater.

Even in cases in which the standard deviation is not greater than the mean, the two values are in the same order of magnitude. This variety cannot be simply explained by outliers. Comparing the median with the median average deviation yields similar results (except that in some of the case, both the median and the MAD statistics are 0). We learn that even the median cannot serve as a good approximation of the "bulk" of the distribution of feature values.

Among other things, this variety means that feature values are not normally distributed, thereby making it difficult to apply many of the standard statistical tools. This variety also raises a difficulty in applying statistical tools such as linear regression which tacitly assumes that values are consistent with the assumption of normal distribution.

Second, there is a variety of feature values between the corpora, as can be seen in the summarizing rows of Table 6. Despite the smoothing effect of averaging many files within each corpus, we see that neither the mean nor the median in distinct corpora are close to each other. Very telling is the comparison of the last two lines in the table, which shows that the Max statistic is many times larger than the Min statistic. This second variety is to be expected. Software projects tend to have their own "development culture", which dictates among other things, the detail of accuracy of commit comments, frequency of commits, and size of these.

This variety is the reason why we always use rank (within each corpus) of feature values rather than their actual magnitude.

6 Metrics' validity

Having described the metrics and the features, we proceed to the study of the validity of the quality of metrics as features *predictors*. The statistical tool employed toward this end is Kendall τ correlation. As explained in Section 4, τ values have a natural interpretation as measures of predictive power.

We computed τ values for each metric and feature pair in each of the corpora. As it turns out, the distribution of the τ values in the different corpora for each such pair appears to be normal. (Concretely, it is not rejected by the Shapiro-Wilk test for normality with $p = 0.05$.) This phenomena is an indication that examination of the predictive value of metrics with τ values, reduces much of the variety between corpora, or, conversely, that the manner by which corpora are selected is statistically sound.

Tables 7 and 8 provide the mean and the standard deviation of these τ values.

A qualitative judgment suggested by the table is that the predictive powers of any of the metric with respect to three features are roughly the same. This phenomenon may be an indication that the three features pertain to the same underlying property, probably the development effort incurred in association with a source file. A closer inspection reveals that the prediction quality with respect to Bugginess is the least, and that it is at its best with respect to Change complexity.

We also see in Tables 7 and 8 that the majority of the metrics are valid at least in the $\tau \geq 0.1$ level, i.e., the prediction they offer is superior to the random prediction (in the sense suggested by (5) above). There are precisely five metrics whose validity with respect to the Change Complexity feature does not reach this threshold: CON, DIT, MNK, NOC and SHA. Curiously, these are the same five metrics identified (at the end of Section 4) whose correlation with NOT is the least.

As we shall see below, this is not a coincidence: a tightly fit linear regression model makes it possible to infer the validity of a metric from its correlation with NOT. But, before

Table 7 Mean and standard deviation across corpora of the predictive qualities of metrics (Kendall τ, points) and their mean Kendall τ correlation (in points) with NOT

Metric	Instability	Change Complexity	Bugginess	∼NOT
ABC3	0.11±0.11	0.13±0.13	0.10±0.11	0.54
CBO	0.15±0.16	0.14±0.13	0.12±0.14	0.70
CCC	0.11±0.11	0.13±0.13	0.10±0.11	0.54
CHAM	0.13±0.14	0.13±0.13	0.11±0.13	0.68
CON	0.05±0.09	0.06±0.09	0.03±0.08	0.27
DIT	0.00±0.09	0.00±0.09	0.02±0.08	0.00
ENT	0.15±0.15	0.14±0.15	0.12±0.13	0.70
ETF	0.14±0.15	0.16±0.16	0.12±0.13	0.74
GZIP	0.16±0.15	0.16±0.17	0.13±0.14	0.85
HOR	0.15±0.15	0.16±0.17	0.13±0.14	0.80
LCOM	0.09±0.10	0.12±0.12	0.08±0.11	0.56
LOC	0.15±0.15	0.16±0.17	0.12±0.13	0.89
LPC	0.12±0.11	0.12±0.13	0.11±0.11	0.56
LZW	0.16±0.16	0.16±0.17	0.13±0.14	0.94
MCC	0.15±0.15	0.15±0.17	0.13±0.14	0.70
MCCS	0.15±0.15	0.15±0.17	0.13±0.14	0.70
MNK	0.00±0.03	0.00±0.03	0.00±0.03	0.00
MTE	0.06±0.12	0.09±0.12	0.06±0.10	0.37
NIV	0.09±0.10	0.11±0.11	0.07±0.09	0.51
NOC	0.02±0.05	0.04±0.06	0.03±0.06	0.06
NOM	0.11±0.12	0.14±0.15	0.09±0.10	0.69
NOS	0.15±0.15	0.16±0.17	0.13±0.14	0.86
NOT	0.15±0.16	0.16±0.17	0.12±0.14	1.00
RFC	0.15±0.16	0.15±0.17	0.13±0.14	0.83
SHA1	0.00±0.02	0.00±0.03	0.00±0.03	0.00
WMC	0.15±0.16	0.15±0.17	0.13±0.14	0.86

proceeding further, we ask whether metric NOT is really the best predictor, or perhaps it is the case that we should seek a fit between metrics' validity and correlation with other metric.

Examining Tables 7 and 8 together we see, e.g., that the value of τ for the pair GZIP and Instability, is greater than that of the pair NOT and Instability (36.4 vs. 35.2). To judge whether this difference and other such differences are significant, we employ the *paired student's t-test*: For each feature f and for each metric μ, $\mu \neq$ NOT, the pairs submitted to the t-test are obtained by iterating over the corpora

$$\langle \tau_A(\mu, f), \tau_A(\mathsf{NOT}, f)\rangle, \ldots, \langle \tau_Z(\mu, f), \tau_Z(\mathsf{NOT}, f)\rangle. \tag{8}$$

For lack of space, we only provide a summary of the results. As expected, only a handful of the metrics perform better than NOT, i.e., win in the majority of the pairs in the array (8). These metrics are the nonsensical metric ETF as well as GZIP, LZW and the famous LOC metric. However, the differences between NOT and LOC and between NOT and ETF are never statistically significant.

Table 8 Mean and standard deviation across corpora of the predictive qualities of metrics (Spearman, points) and their mean Kendall τ correlation (in points) with NOT

Metric	Instability	Change Complexity	Bugginess	∼NOT
ABC3	0.16±0.15	0.18±0.18	0.13±0.14	0.72
CBO	0.21±0.22	0.19±0.19	0.15±0.18	0.86
CCC	0.16±0.15	0.18±0.18	0.12±0.14	0.72
CHAM	0.18±0.18	0.18±0.18	0.14±0.16	0.84
CON	0.06±0.11	0.08±0.11	0.04±0.10	0.35
DIT	0.00±0.11	0.00±0.11	0.02±0.09	0.00
ENT	0.22±0.21	0.20±0.21	0.16±0.17	0.87
ETF	0.20±0.21	0.22±0.23	0.16±0.17	0.91
GZIP	0.23±0.22	0.22±0.23	0.17±0.18	0.97
HOR	0.22±0.21	0.22±0.23	0.16±0.17	0.94
LCOM	0.13±0.14	0.16±0.17	0.11±0.13	0.72
LOC	0.22±0.21	0.23±0.23	0.16±0.17	0.98
LPC	0.15±0.14	0.15±0.17	0.13±0.13	0.70
LZW	0.22±0.22	0.23±0.24	0.17±0.18	0.99
MCC	0.20±0.20	0.20±0.22	0.16±0.17	0.85
MCCS	0.20±0.20	0.20±0.22	0.16±0.17	0.85
MNK	0.00±0.05	0.01±0.04	0.00±0.04	0.00
MTE	0.07±0.16	0.12±0.17	0.07±0.11	0.49
NIV	0.12±0.14	0.15±0.16	0.09±0.11	0.66
NOC	0.03±0.06	0.04±0.07	0.04±0.07	0.08
NOM	0.16±0.17	0.19±0.20	0.12±0.13	0.86
NOS	0.21±0.21	0.22±0.23	0.16±0.17	0.97
NOT	0.22±0.22	0.22±0.24	0.16±0.18	1.00
RFC	0.21±0.21	0.21±0.23	0.16±0.18	0.95
SHA1	0.00±0.03	0.00±0.04	0.00±0.04	0.00
WMC	0.21±0.22	0.21±0.23	0.16±0.18	0.96

For some of the features, there is a tiny statistically significant advantage of GZIP and LZW over NOT, but this advantage is not preserved across all the features, and it is volatile with respect to small changes in the definitions of the metrics. However, about half of these "wins" lack statistical significance, Thus the validity of NOT is as good as any of the competitors. Our slight preference of NOT is justified by its simplicity and robustness.

We note finally that the behavior of the DIT metric is somewhat peculiar. In agreement with previous research (Basili et al. 1996; Gyimothy et al. 2005; Subramanyam and Krishnan 2003) we find fluctuations of this metric. Its correlation with quality may be positive, negative or zero in different corpora. Detailed discussion of this metric is beyond the scope of this paper.

6.1 The importance of using multiple corpora

Previous studies have used only a handful of projects, and separated the analysis of these projects. We offer an alternative in the form of using aggregating all corpora values to a

single value. Both of these—the use of multiple projects and aggregating the results—are very important to the validity of our results.

Due to the idiosyncratic nature of software projects, and the relatively high standard devotion of the validity of the metrics, it is not particularly difficult to find a combination of a metric μ and a corpus c, such that μ is independent of NOT, and valid.

This provides an explanation of some of the previous validations of some of the metrics, e.g., DIT, that our empirical results have shown to be invalid. Suppose that about 5% of all corpora exhibit such said "positive" results. Paired with publication bias—the bias of researchers to publish "positive" results, and discard negative ones—this may lead to a skewing of the results towards areas that are not representative of the great majority of software projects.

7 Isolating size as a feature explainer

We now turn to presenting the main contribution of this work, the LMC phenomenon—a linear relation tying together the correlation of a metric with size and its validity.

Figure 1 is a visualization of LMC for the instability feature.

The X-axis in the figure, spanning the range of $-1 \leq \tau \leq 1$ is correlation with the size metric. The Y-axis is the correlation with instability. Each of the data points in the represents a metric, drawn from the our 76 metrics suite, including the original metrics, their linear-scaled version, and, their rank-scaled version. The Y-coordinate of the point

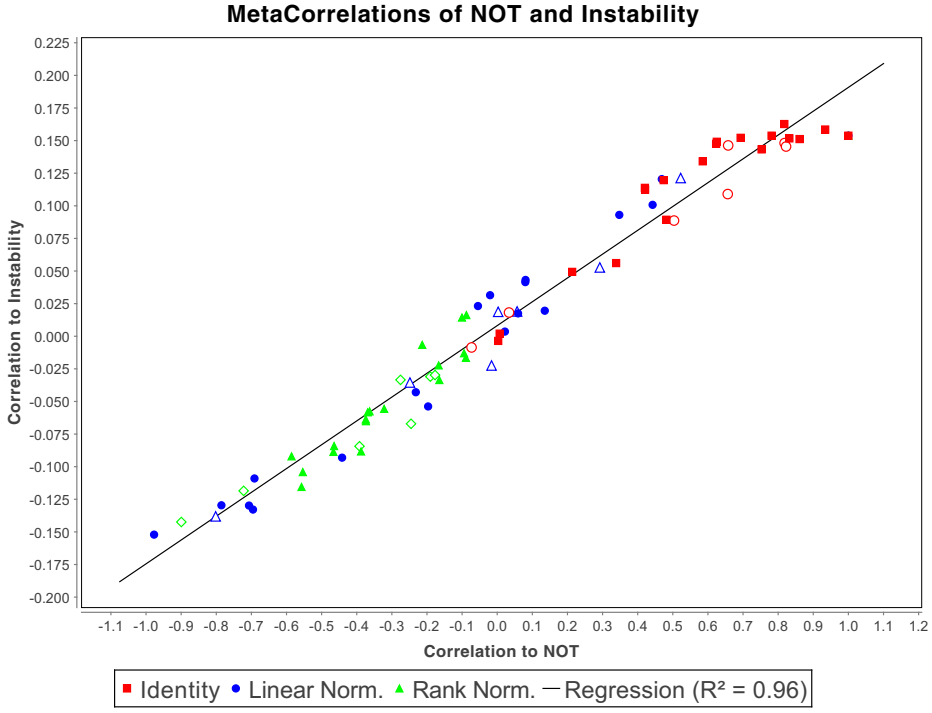

Fig. 1 Correlation with Instability vs. correlation with NOT

is the correlation of the metric with the instability feature; its Y-coordinate is the metric's correlation with the NOT metric.

The linear regression line, represent the IPP value of a metric. The *vertical* distance is the residual metric value (1), i.e., the metric's predictive value minus the predictive value inferred from the regression line.

Figures 1, 2 and 3 plot validity against correlation with NOT, along with a linear regression line. In addition to the plain version of the metric, these figures also includes the linear-scaled version of the metrics (see (6)), as well as their rank-scaled version (see (7)), for a total of $26 + 25 + 25 = 76$ metrics.

Linear regression fitting was carried out with respect to all these 76 metrics; however, the r values reported in the legend of Figs. 1, 2 and 3 are computed from this common regression line to four sets of metrics: the 26 plain metrics, the identity transformation, the 25 metrics obtained by linear scaling, the 25 metrics obtained by rank scaling, and the union of these three sets.

The results presented in Figs. 1, 2 and 3 revolve around the NOT metric in two distinct ways:

– *Correlation:* correlation to size is understood with NOT taken as size.
– *Scaling:* scaling with respect to size is understood as dividing by the value or the rank of NOT.

The next two sub-sections investigate how dependent are the results in these two particular applications of the term "size"

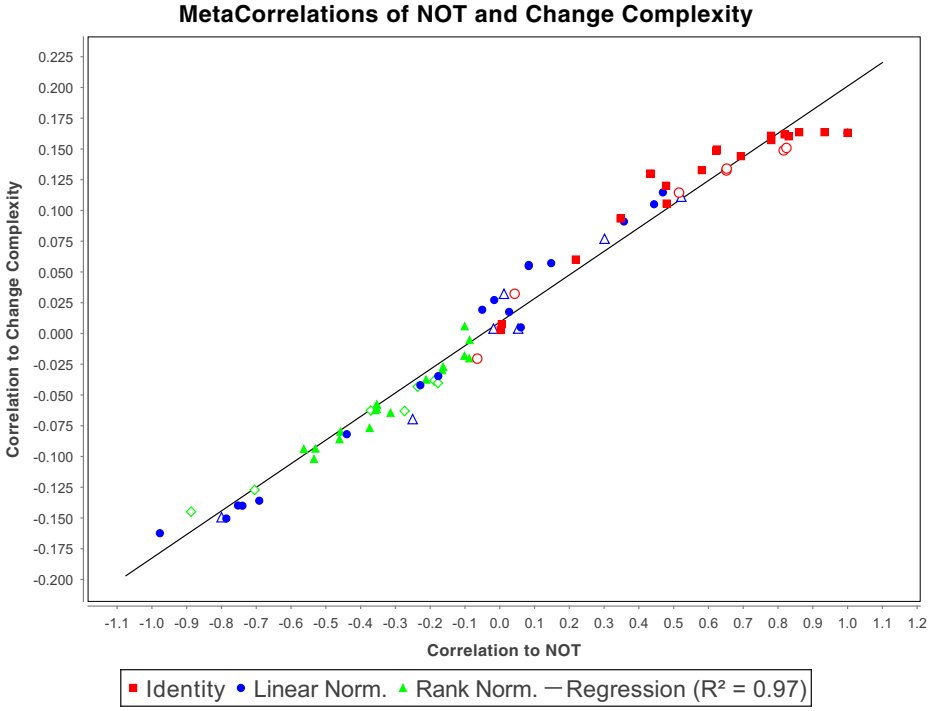

Fig. 2 Correlation with Change Complexity vs. correlation with NOT

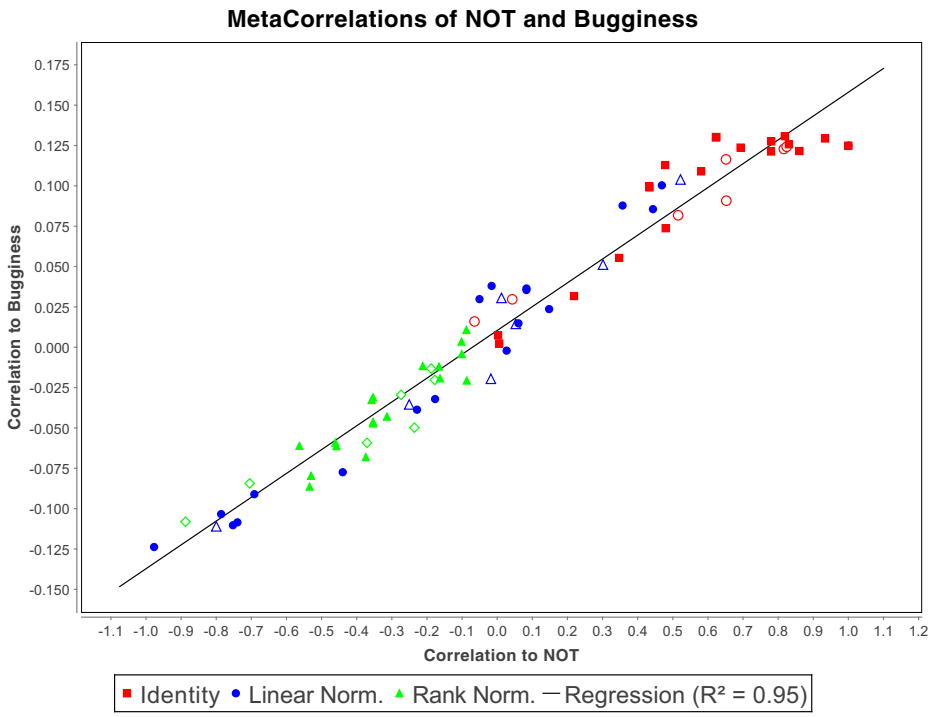

Fig. 3 Correlation with Bugginess vs. correlation with NOT

7.1 Correlation with metrics other than NOT

The numbers in Table 9 reveal the impact of variation in interpretation of "size" for the purpose of computing "correlation with size".

Table 9 Pearson r values of the metacorrelation between validity and correlation to the explaining metrics for each of our features and transformations

Metric	Feature	Identity	Linear Scaling	Rank Scaling	Global Metric Suite
NOT	Instability	0.95	0.98	0.98	0.98
	Change Complexity	0.96	0.98	0.98	0.99
	Bugginess	0.93	0.97	0.97	0.98
GZIP	Instability	0.97	0.99	0.99	0.99
	Change Complexity	0.96	0.99	0.99	0.99
	Bugginess	0.94	0.98	0.98	0.98
MCC	Instability	0.90	0.97	0.97	0.96
	Change Complexity	0.88	0.95	0.95	0.96
	Bugginess	0.91	0.98	0.98	0.97
NOC	Instability	0.31	0.10	0.10	0.03
	Change Complexity	0.26	0.17	0.17	0.10
	Bugginess	0.27	0.15	0.15	0.08

The first group of three rows in the table (those marked with NOT) are a summary of the r coefficients of Figs. 1, 2 and 3. This group is the baseline, to which we compare the three other groups, in which the correlation was with respect to a more or less accurate approximation of the concept "size".

In the second group, metacorrelation was computed with respect to the GZIP metric. Comparing this group to the first we see that r values are as good or slightly better when measured with respect correlation with GZIP. This might mean that, as argued by some authors, that the size of the compressed code is more indicative of its complexity than the size of uncompressed code.

In contrast, the third rows group in Table 9 employs correlation with MCC; the fourth, with NOC. The metacorrelation values are evidently not as good as with NOT.

It is interesting to make a visual comparison of the regressions in Fig. 4 (correlation with instability vs. correlation with GZIP) and Fig. 5 (correlation with instability vs. correlation with NOC). The LMC phenomenon is evident in Fig. 4 (just as it is evident in Fig. 1, which relies on correlation with NOT), and is clearly not present in Fig. 5.

7.2 Scaling with metrics other than NOT

As mentioned, another aspect of using NOT as a measure of size, is scaling. In 4 we used linear and rank scaling to remove the dependence of metrics on size. In this section we explore scaling with other "size" metrics, LOC and LZW.

Comparing Tables 10 and 11 to Table 9 shows that the results are very similar regardless of the measure of size used for scaling.

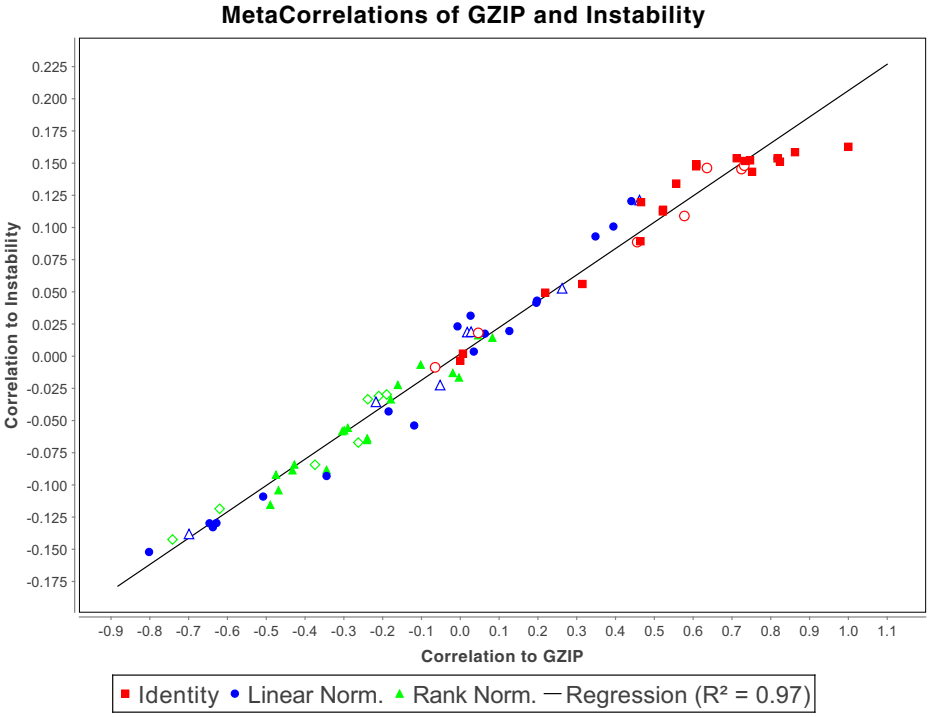

Fig. 4 Correlation with instability *vs.* correlation with GZIP

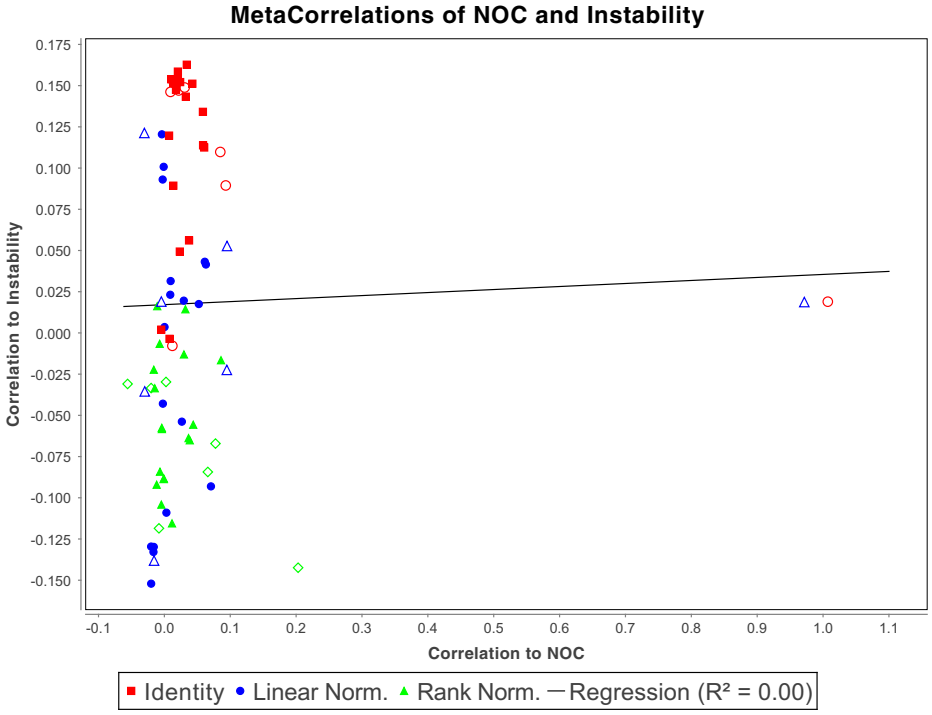

Fig. 5 Correlation with instability *vs.* correlation with NOC

7.3 Examining the C&K suite in depth

We applied the student t-test to measure the residuals of the C&K metrics compared to the residuals of the other metrics in our suite. The results for each metric are shown in Table 12.

Table 10 Pearson's r values of the metacorrelation between validity and correlation to the explaining metrics for each of our features and transformations, when using LOC as a measure of size

Metric	Feature	Identity	Linear Scaling	Rank Scaling	Global Metric Suite
LOC	Instability	0.96	0.98	0.98	0.99
	Change Complexity	0.96	0.98	0.98	0.99
	Bugginess	0.94	0.97	0.97	0.98
GZIP	Instability	0.97	0.99	0.99	0.99
	Change Complexity	0.96	0.99	0.99	0.99
	Bugginess	0.94	0.98	0.98	0.98
MCC	Instability	0.90	0.96	0.96	0.96
	Change Complexity	0.88	0.95	0.95	0.95
	Bugginess	0.91	0.97	0.97	0.97
NOC	Instability	0.31	0.03	0.03	0.01
	Change Complexity	0.26	0.09	0.09	0.07
	Bugginess	0.27	0.09	0.09	0.06

Table 11 Pearson's r values of the metacorrelation between validity and correlation to the explaining metrics for each of our features and transformations, when using LZW as a measure of size

Metric	Feature	Identity	Linear Scaling	Rank Scaling	Global Metric Suite
LZW	Instability	0.96	0.98	0.98	0.99
	Change Complexity	0.96	0.98	0.98	0.99
	Bugginess	0.94	0.97	0.97	0.98
GZIP	Instability	0.97	0.99	0.99	0.99
	Change Complexity	0.96	0.99	0.99	0.99
	Bugginess	0.94	0.98	0.98	0.98
MCC	Instability	0.90	0.96	0.96	0.96
	Change Complexity	0.88	0.95	0.95	0.95
	Bugginess	0.91	0.97	0.97	0.97
NOC	Instability	0.31	0.03	0.03	0.01
	Change Complexity	0.26	0.09	0.09	0.07
	Bugginess	0.27	0.09	0.09	0.06

We conclude that that the C&K metrics do not divert significantly from the pattern of the linear correlation between validity and correlation to size.

This particular result casts doubts on the validity of the C&K suite, which is the most widely cited object-oriented suite, as they too are not exempt from the LMC effect.

8 Threats to validity

> Let them therefore give us two bullocks; and let them choose one bullock for themselves, and cut it in pieces, and lay it on wood, and put no fire under: and we will dress the other bullock, and lay it on wood, and put no fire under.
>
> And call ye on the name of your gods, and we will call on the name of the Lord: and the God that answereth by fire, let him be God. And all the people answered and said, It is well spoken.
>
> IKings(18:23-24)

1. *Dataset and Metrics.* The unique nature of open source software, where a diverse group of often unrelated developers create code independently[13] that is later merged together (or not, as sometimes the development batch is rejected), can bias the results. Further research using closed-source artifacts may yield different results.

   Instead, we offer valuable information to the engineer who wishes to gauge the health of his open-source software artifacts. While one may be tempted to use a myriad of software metrics, it seems none offer any information that is not the result of size.
2. *Features used.* Three different features, each with its unique rationale, were examined. All three showed similar results. The features were motivated by similar features used in the literature, however the precise formulation is different, for the usual reasons: ability to measure and the classification of boundary cases. Our results may therefore be scrutinized by a more faithful implementation of classical features.

[13] i.e., with no meetings with other team-members and no guidance.

Table 12 The p-values of the student's t-test measure the difference in residuals from the regression line between C&K and the other metrics

Feature	Identity	Linear Scaling	Rank Scaling	Global Metric Suite
Instability	0.80	0.56	0.38	0.96
Change Complexity	0.78	0.52	0.37	0.97
Bugginess	0.77	0.52	0.37	0.96

A wider variety of "deeper" features is also a valid research direction. Features such as *reusability* (Caldiera and Basili 1991; Mohagheghi et al. 2004) and *maintainability* (Li and Henry 1993; Coleman et al. 1994) have been recognized as critical external qualities of software, and might be worth the great challenge of large scale measuring. An argument against the effort is the low plausibility of metrics having impact on "deep" features without showing their impact on the evolution of the software as measured here.

Conversely, We argue that that the features we used offer their own merits. Even if they they do not directly evaluate the software product, they are essential in evaluating the cost of producing it. We also argue that the three features are straightforward to implement on the spectrum of `git` repositories on the web.

3. *Correlation of features to size.* One might argue that the correlation of size to features is trivial: larger files are the focus of more work and therefore more bugs. While this is obviously true, we note that we didn't find evidence of any other factor. Further research may be needed to see how different features behave when normalized to the size of the module.

9 Conclusions

Our main result here it that,

Metrics are only as valid as their correlation to size.

Of course, and as usual with empirical research of this sort, the generality of this claim is limited to the scope of this research, including the particular programming language, the sort of projects in our sample, the kind of features we studied, and the set of metrics we used.

Even though we show that the C&K metrics fall on our linear regression line, in the same fashion as other metrics, we do not claim that this contradicts previous results in the empirical validation of software metrics, e.g., that of Basili et al. (1996). Our suggestion is that future empirical research of software metrics address the linear regression line suggested here. It is not enough to simply correlate code metric values with external features, such as maintainability. Rather, one has to also take into consideration that all correlations are simply a product of the dependence of metrics to size. Therefore, one has to either decouple the metrics from size or show that the external features themselves are not correlated with size.

Additionally, we would argue that empirical researchers further validate their results on an array of projects, rather than focusing on a single or handful software artifacts. This step is crucial in order to avoid the idiosyncratic effects of the projects undergoing evaluation, as discussed in Section 6.1.

We believe that finding an external feature which is both independent of size and makes a good predictors of external qualities, remains an open question. This research failed to find any.

References

Alan O, Catal C (2011) Thresholds based outlier detection approach for mining class outliers: an empirical case study on software measurement datasets. Expert Syst Appl 38(4):3440–3445

Allamanis M, Charles S (2013) Mining source code repositories at massive scale using language modeling. In: The 10th working conference on mining software repositories. IEEE, pp 207–216

Arnold K, Gosling J (1996) The JAVA programming language. The JAVA Series. Addison-Wesley, Reading MA

Basili VR, Briand LC, Melo WL (1996) A validation of object-oriented design metrics as quality indicators. IEEE Trans Softw Eng 22(10):751–761

Baxter G, Frean M, Noble J, Rickerby M, Smith H, Visser M, Melton H, Tempero E (2006) Understanding the shape of JAVA software. In: Tarr PL, Cook WR (eds) Proceedings of the 21st Ann. Conf. on OO Prog. Sys., Lang., & Appl. (OOPSLA'06). ACM, Portland, Oregon

Bird C, Bachmann A, Aune E, Duffy J, Bernstein A, Filkov V, Devanbu P (2009) Fair and balanced?: bias in bug-fix datasets. In: Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, pp 121–130. ACM

Caldiera G, Basili VR (1991) Identifying and qualifying reusable software components. IEEE Comp. pp 61–70

Chidamber SR, Kemerer CF (1994) A metrics suite for object oriented design. IEEE Trans Softw Eng 20(6):476–493

Coleman D, Ash D, Lowther B, Oman P (1994) Using metrics to evaluate software system maintainability. Computer 27(8):44–49

Deutsch LP (1996) Gzip file format specification version 4.3. RFC #1952

Eastlake D, Jones P (2001) US secure hash algorithm 1 (SHA1)

El Emam K, Benlarbi S, Goel N, Rai SN (2001) The confounding effect of class size on the validity of object-oriented metrics. IEEE Trans Softw Eng 27(7):630–650

Fayad ME, Altman A (2001) Thinking objectively: an introduction to software stability. Commun ACM 44(9):95

Fenton N, Bieman J (2014) Software metrics: a rigorous and practical approach, 3rd edn. CRC Press, Inc., Boca Raton, FL

Gil J, Itai A, Jul E (1998) The complexity of type analysis of object oriented programs, vol 1445. Springer, Brussels, Belgium

Gillies A (2011) Software quality: theory and management. Lulu.com

Gyimothy T, Ferenc R, Siket I (2005) Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans Softw Eng 31(10):897–910

Halstead MH (1977) Elements of software science. Elsevier, NY

Harrison WA, Magel KI (1981) A complexity measure based on nesting level. ACM Sigplan Notices 16(3):63–74

Henderson-Sellers B (1996) Software metrics. Prentice-Hall

Herraiz I, Gonzalez-Barahona JM, Robles G (2007) Towards a theoretical model for software growth. In: Proceedings of the 4th international workshop on mining software repositories. IEEE Computer Society, p 21

Hindle A, Godfrey M, Holt R (2008) Reading beside the lines: Indentation as a proxy for complexity metric. In: The 16th IEEE international conference on program comprehension (ICPC'08), pp 133–142

Jbara A, Feitelson DG (2014) On the effect of code regularity on comprehension. In: ICPC, pp 189–200

Khoshgoftaar TM, Munson JC (1990) Predicting software development errors using software complexity metrics. IEEE J Sel Areas Commun 8(2):253–261

Li HF, Cheung WK (1987) An empirical study of software metrics. IEEE Trans Softw Eng 13(6):697–708

Li W, Henry S (1993) Object-oriented metrics that predict maintainability. Sys Soft 23:111–122

Lorenz M, Kidd J (1994) Object-oriented software metrics: a practical guide. Prentice-Hall

McCabe TJ (1976) A complexity measure. IEEE Trans Softw Eng 2(4):308–320

Meyer B (1988) Object-oriented software construction. International Series in Computer Science. Prentice-Hall

Mohagheghi P, Conradi R, Killi OM, Schwarz H (2004) An empirical study of software reuse vs. defect-density and stability. In: Proceedings of the 26th international conference on software engineering (ICSE'04). IEEE Computer Society Press, Edinburgh, Scotland, United Kingdom, pp 282–291

Olague HM, Etzkorn LH, Gholston S, Quattlebaum S (2007) Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. IEEE Trans Softw Eng 33(6):402–419

Piwowarski P (1982) A nesting level complexity measure. ACM Sigplan Notices 17(9):44–50

Shepperd M (1988) A critique of cyclomatic complexity as a software metric. Softw Eng J 3(2):30–36

Subramanyam R, Krishnan M (2003) Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects. IEEE Trans Softw Eng 29(4):297–310. doi:10.1109/TSE.2003.1191795

Zhou Y, Leung H, Xu B (2009) Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness. IEEE Trans Software Eng 35(5):607–623

Ziv J, Lempel A (1978) Compression of individual sequences via variable-rate coding. IEEE Trans Inf Theory 24(5):530–536

Yossi Gil is an Associate Professor at Computer Science Department at the Technion. He also served as a visiting researcher at IBM research centers, Google, and at the Department of Computer Science at the University of British Columbia. He holds a B.Sc, M.Sc. and a Ph.D from the Hebrew University of Jerusalem.

His research interests span a wide range of topics, such as PRAM computation, lower bounds, algorithms, pattern analysis, and objectoriented programming. He earned a 10 year best paper award on his work on Design Patterns (together with Eden and Yehudai) and Visual Modelling (with Kent). His poems granted him the "Ofer Lider" prize for literary work among scientists. Among his eclectic projects is the 8086 assembly program "Terse", the marvel 4K full screen editor and the independent solution of the quartic polynomial equation.

Gal Lalouche is a PhD student in the computer science faculty in the Technion.