An Extended Study of the Correlation of Cognitive Complexity-related Code Measures

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

Abstract—**Several measures have been proposed to represent various characteristics of code, such as size, complexity, cohesion, coupling, etc. These measures are deemed interesting because the internal characteristics they measure (which are not interesting *per se*) are believed to be correlated with external software qualities (like reliability, maintainability, etc.) that are definitely interesting for developers or users. Although many measures have been proposed for software code, new measures are continuously proposed. However, before starting using a new measure, we would like to ascertain that it is actually useful and that it provides some improvement with respect to well established measures that have been in use for a long time and whose merits have been widely evaluated. In 2018, a new code measure, named "Cognitive Complexity" was proposed. According to the proposers, this measure should correlate to code understandability much better than traditional code measures, such as McCabe Complexity, for instance. However, hardly any experimentation proved whether the "Cognitive Complexity" measure is better than other measures or not. Actually, it was not even verified whether the new measure provides different knowledge concerning code with respect to traditional measures. In this paper, we aim at evaluating experimentally to what extent the new measure is correlated with traditional measures. To this end, we measured the code from a set of open-source Java projects and derived models of "Cognitive Complexity" based on the traditional code measures yielded by a state-of-the-art code measurement tool. We found that fairly accurate models of "Cognitive Complexity" can be obtained using just a few traditional code measures. In this sense, the "Cognitive Complexity" measure does not appear to provide additional knowledge with respect to previously proposed measures.**

Keywords–Cognitive complexity; software code measures; McCabe complexity; cyclomatic complexity; Halstead measures; static code measures

I. Introduction

In [1], the correlation between "Cognitive Complexity," a measure proposed with the aim of representing the complexity of understanding code [2], and "traditional" measures was studied.

In fact, many measures of the internal characteristics of code, such as size, complexity, cohesion or coupling, have been proposed in the past (for instance, Chidamber and Kemerer proposed a suite of metrics that are suitable for representing the characteristics of object-oriented code [3]) and new ones are continuously proposed. However, code measures are of little interest *per se*, since they address internal properties of software. In general, developers, managers and users are more interested in external software qualities, like faultiness or maintainability. Therefore, it is necessary that internal property measures are correlated to some external property of interest. Such correlation makes it possible, among other things, to predict interesting external qualities, which are unknown, based on measures of internal code properties, which can be easily collected.

In 2018, a new code measure was proposed with the aim of representing the complexity of understanding code [2]. The measure is a code measure, which accounts exclusively for internal code properties. However, according to the author, it is expected to be strictly correlated with code understandability, which is an external code property. This measure is named "Cognitive Complexity," however, in the remainder of this paper we shall refer to this measure as "CoCo," to avoid confusion with the *concept* of cognitive complexity, i.e., the external property that CoCo is expected to measure.

Some initial work has been done to evaluate whether CoCo is actually correlated with code understandability [4]: preliminary results do not support the claim that CoCo is better correlated to code understandability than previously proposed measures.

At any rate, whatever the goal that a new code measure is supposed to help achieving, the new measure should provide some "knowledge" that existing code measures are not able to capture. If a new measure is so strongly correlated with other measures that the latters can be used to predict the new measure with good accuracy, it is unlikely that the new measure actually conveys any new knowledge.

CoCo is receiving some attention, probably because it is provided by `SonarQube`, which is a quite popular tool. Therefore, it is time to look for evidence that CoCo provides additional knowledge with respect to well established code measures. To this end, the following two research questions were addressed by a previous paper [1]:

RQ1 How strongly is CoCo correlated with each of the code measures that are commonly used in software development?

RQ2 Is it possible to build models that predict the value of CoCo based on the values of commonly used code measures? If so, how accurate are the predictions that can be achieved?

These research questions were addressed by analyzing the code from nine open source Java projects. It was found that CoCo appears strongly correlated to McCabe's complexity and slightly less strongly correlated to several other code measures. Several regression models of CoCo as a function of traditional measures were also found. Based on these findings, it was concluded that—at least for the considered software projects—CoCo does not appear to convey additional information with respect to traditional measures.

In this paper, we first report in some detail the work described in [1], then we look for further evidence that may confirm (or challenge) previous findings. To this end,

1) We selected two large open source Java projects, whose code we used in the following activities.
2) We evaluated the correlation between CoCo and traditional measures in the two selected open source Java projects.
3) We used two models found in [1] to predict the CoCo of the two projects' methods. We then evaluated the accuracy of the prediction.
4) Finally, we used machine learning techniques to verify whether it is possible to estimate CoCo based on traditional measures with an even greater accuracy than via regression.

The paper is structured as follows. Section II provides some background, by introducing CoCo and describing the traditional code measures used in this study. Section III describes the empirical study that was carried out to answer the research questions. Section IV discusses the results obtained by the original study [1] and answers the research questions. Section V describes a second empirical study, which provides further evidence that confirms previous findings. Section VI explores whether it is possible to uncover even stronger relationships by means of machine learning, or if the usage of ML just confirms the previous findings obtained via ordinary least squares regression. Section VII discusses the threats to the validity of the study. Section VIII accounts for related work. Finally, in Section IX some conclusions are drawn, and future work is outlined.

II. Code Measures

In this paper we deal with measures of the internal attributes of code. Internal attributes of code can be measured by looking at code alone, without considering software qualities (like faultiness, robustness, maintainability, etc.) that are externally perceivable.

Several measures for internal software attributes (e.g., size, structural complexity, cohesion, coupling) were proposed [5] to quantify the properties of software modules. These measures are interesting because they concern code properties that are believed to affect external software qualities (like faultiness or maintainability), which are interesting for developers and users.

Since CoCo is computed at the method level, in what follows, we consider only measures at the same granularity level, i.e., measures that are applicable to methods.

A. *"Traditional" Code Measures*

Since the first high-level programming languages were introduced, several measures were proposed, to represent the possibly relevant characteristics of code. For instance, the Lines Of Code (LOC) measure the size of a software module, while McCabe Complexity (also known as Cyclomatic Complexity) [6] was proposed to represent the "complexity" of code, with the idea that high levels of complexity characterize code that is difficult to test and maintain. The object-oriented measures by Chidamber and Kemerer [3] were proposed to recognize poor software design: for instance, modules with high levels of coupling are supposed to be associated with difficult maintenance.

In this paper, we are interested in evaluating the correlation between CoCo and traditional measures. Since CoCo is defined at the method level, here we consider only traditional measures addressing methods; measures defined to represent the properties of classes or other code structures are ignored.

We used SourceMeter [7] to collect code measures. The collected method-level measures are listed in Table I.

Here we provide just a brief description of the collected measures; readers can find complete specifications and additional information in the documentation of SourceMeter.

The measures listed in Table I include Halstead measures [8], several maintainability indexes, including the original one [9], McCabe complexity, measures of the nesting level (i.e., how deeply are code control structures included in each other), logical lines of code (which are counted excluding blank lines, comment-only lines, etc.).

TABLE I
THE MEASURES COLLECTED VIA SOURCEMETER.

Metric name	Abbreviation
Halstead Calculated Program Length	HCPL
Halstead Difficulty	HDIF
Halstead Effort	HEFF
Halstead Number of Delivered Bugs	HNDB
Halstead Program Length	HPL
Halstead Program Vocabulary	HPV
Halstead Time Required to Program	HTRP
Halstead Volume	HVOL
Maintainability Index (Microsoft version)	MIMS
Maintainability Index (Original version)	MI
Maintainability Index (SEI version)	MISEI
Maintainability Index (SourceMeter version)	MISM
McCabe's Cyclomatic Complexity	McCC
Nesting Level	NL
Nesting Level Else-If	NLE
Logical Lines of Code	LLOC
Number of Statements	NOS

B. *The "Cognitive Complexity" Measure*

In 2017, SonarSource introduced Cognitive Complexity [2] as a new measure for the understandability of any given piece of code. This new measure was named "Cognitive Complexity" because its authors assumed that the measure was suitable to represent the cognitive complexity of understanding code. To this end, CoCo was proposed with the aim *"to remedy*

Cyclomatic Complexity's shortcomings and produce a measurement that more accurately reflects the relative difficulty of understanding, and therefore of maintaining methods, classes, and applications" [2].

Rather than a direct measure, CoCo is an indicator, which takes into account several aspects of code. Like McCabe's complexity, it takes into account decision points (conditional statements, loops, switch statements, etc.), but, unlike McCabe's complexity, CoCo gives them a weight equal to their nesting level plus 1. So, for instance, in the following code fragment

```
void firstMethod() {
  if (condition1)
    for (int i = 0; i < 10; i++)
      while (condition2) { x+=a[i]; }
}
```

the `if` statement at nesting level 0 has weight 1, the `for` statement at nesting level 1 has weight 2, and the `while` statement at nesting level 2 has weight 3; accordingly CoCo= $1+2+3=6$. The same code has McCabe complexity = 4 (3 decision points plus one).

Consider instead the following code fragment, in which the control structures are not nested.

```
void secondMethod() {
  if (condition1) { x=0; }
  for (int i = 0; i < 10; i++) { x+=2*i; }
  while (condition2) { x=x/2; }
}
```

This code has CoCo = 3, while its McCabe complexity is still 4. It is thus apparent that nested structures increase CoCo, while they have no effect on McCabe complexity.

CoCo also accounts for Boolean predicates (while McCabe's complexity does not): a Boolean predicate contributes to CoCo depending on the number of its sub-sequences of logical operators. For instance, consider the following code fragment, where `a, b, c, d, e, f` are Boolean variables

```
void thirdMethod() {
  if (a && b && c || d || e && f) { ... }
}
```

Predicate `a && b && c || d || e && f` contains three sub-sequences with the same logical operators, i.e., `a && b && c`, `c || d || e`, and `e && f`, so it adds 3 to the value of CoCo.

Other aspects of code contribute to increment CoCo, but they are much less frequent than those described above. For a complete description of CoCo, see the definition [2].

III. The Empirical Study

The empirical study involved a set of open-source Java programs. The Java code was measured, and the collected data were analyzed via well consolidated statistical methods. The dataset is described in Section III-A, while the measurement and analysis methods are described in Section III-B. The results we obtained are reported in Section III-C.

A. The Dataset

The code to be analyzed within the study was a convenience sample: data whose code was already available from previous studies concerning completely different topics was used. In practice, this amount to a random choice.

The projects that supplied the code for the study are listed in Table II, where some descriptive statistics for the most relevant measures are also given (for space reasons, statistics are given only for a subset of representative measures). Methods having CoCo=0 (i.e., with no decidion points, no complex boolean expressions, etc.) or NOS=0 (i.e., having no statements) are clearly uninteresting, therefore their data were excluded, so Table II does not account for such methods. Overall, the initial dataset included data from 13,922 methods. The dataset is available on demand for replication purposes.

B. The Method

The first phase of the study consisted in measuring the code. We used SourceMeter to obtain the traditional measures listed in Table I, and a self-constructed tool to measure CoCo. The data from the two tools were joined, thus obtaining a single dataset with 8,214 data points.

The second step consisted in selecting the data for the study. We excluded from the study all the methods having CoCo <5, since those methods would bias the results, because of 'built-in' relationships. For instance a piece of code having CoCo = 0 also has McCabe complexity = 1; similarly, CoCo = 1 implies that McCabe complexity = 2 for all but a few very peculiar cases, etc. In addition, low-complexity methods are of little interest: CoCo is meant to represent the complexity of understanding code, and CoCo is less than 5 for methods that are so simple that understanding them is hardly an issue. Therefore, by excluding only methods having CoCo < 5 we are sure to exclude only 'non-interesting' code.

We also excluded methods having CoCo > 50, because our dataset contains too few methods having CoCo > 50 to support reliable statistical analysis. Besides, CoCo > 50 indicates exceedingly complex methods; in practice, it is hardly useful knowing if, say, CoCo = 60 or CoCo = 70, just like it is hardly useful knowing that McCabe's complexity is 60 or 70. In these cases, we just have "too complex" methods.

After removing the exceedingly simple or complex methods, we got a dataset including 3,610 data points, definitely enough to perform significant statistical analysis. In this dataset the mean value of CoCo is 12, while the median is 9.

The third step consisted in performing statistical analysis. We started by studying the correlation between CoCo and each one of the other code measures. Since the data are not normally distributed, we used non-parametric tests, namely we computed Kendall's rank correlation coefficient τ [10] and Spearman's rank correlation coefficient ρ [11]. Since the correlation analysis gave encouraging results, we proceeded to evaluate correlations via both linear and non-linear correlation

TABLE II
DESCRIPTIVE STATISTICS OF THE DATASETS.

Project	num. methods	Measure	mean	st.dev.	median	min	max
hibernate	2532	CoCo	3.1	4.3	2.0	1	79
		HPV	32.3	17.1	28.0	0	211
		MI	100.3	14.7	102.2	0	135
		McCC	3.3	2.4	2.0	1	33
		NLE	1.3	0.8	1.0	0	7
		LLOC	15.2	12.3	12.0	3	201
jcaptcha	317	CoCo	3.3	4.0	2.0	1	34
		HPV	35.0	18.4	29.0	10	120
		MI	100.3	14.0	102.5	56	132
		McCC	3.5	2.2	3.0	2	18
		NLE	1.3	0.8	1.0	0	5
		LLOC	14.6	10.6	11.0	3	80
jjwt	205	CoCo	4.0	7.2	2.0	1	84
		HPV	30.6	22.9	28.0	0	280
		MI	101.7	20.6	104.0	0	135
		McCC	4.3	4.6	3.0	2	46
		NLE	1.3	0.8	1.0	0	4
		LLOC	13.5	14.9	11.0	3	169
json_ iterator	379	CoCo	5.6	8.7	3.0	1	73
		HPV	38.3	21.1	32.0	14	145
		MI	96.4	15.3	99.0	45	131
		McCC	4.6	3.9	3.0	1	28
		NLE	1.6	1.0	1.0	0	7
		LLOC	18.0	15.1	13.0	3	110
JSON-java	260	CoCo	5.7	15.8	2.0	1	203
		HPV	41.0	36.9	31.5	11	413
		MI	95.7	18.2	97.4	32	133
		McCC	5.0	5.8	3.0	2	50
		NLE	1.5	1.1	1.0	0	7
		LLOC	21.5	26.5	13.0	3	255
log4j	798	CoCo	4.6	6.4	2.0	1	61
		HPV	36.6	21.4	30.0	8	163
		MI	98.1	15.2	100.4	44	135
		McCC	4.1	3.4	3.0	1	34
		NLE	1.6	1.0	1.0	0	8
		LLOC	16.9	13.4	12.0	3	115
netty-socketio	136	CoCo	4.4	5.5	3.0	1	37
		HPV	33.7	20.0	28.0	0	122
		MI	97.7	20.8	101.4	0	132
		McCC	4.1	2.8	3.0	1	19
		NLE	1.6	0.9	1.0	0	5
		LLOC	15.0	12.3	11.0	3	84
pdfbox	3587	CoCo	5.2	8.2	2.0	1	118
		HPV	39.3	25.7	32.0	0	326
		MI	93.7	17.2	96.4	0	128
		McCC	4.5	4.5	3.0	1	58
		NLE	1.6	1.1	1.0	0	10
		LLOC	22.3	21.8	15.0	3	330
jasper reports	6415	CoCo	5.6	10.1	3.0	1	186
		HPV	38.7	28.5	31.0	0	740
		MI	93.4	18.1	96.5	0	132
		McCC	4.9	5.6	3.0	1	117
		NLE	1.6	1.1	1.0	0	10
		LLOC	23.5	26.0	15.0	3	383

analysis. Namely, we performed ordinary least squares (OLS) linear regression analysis and OLS regression analysis after log-log transformation of data. In both cases, we identified and excluded outliers based on Cook's distance [12].

In all the performed analysis, we considered the results significant at the usual $\alpha = 0.05$ level.

C. Results of the Study

The results of Kendall's and Spearman's correlation tests are given in Table III. All the reported results are statistically significant, with p-values well below 0.001.

TABLE III
RESULTS OF CORRELATION TEST.

Measure	τ	ρ
HCPL	0.45	0.62
HDIF	0.38	0.52
HEFF	0.47	0.63
HNDB	0.47	0.63
HPL	0.50	0.67
HPV	0.46	0.62
HTRP	0.47	0.63
HVOL	0.50	0.66
MI	-0.56	-0.73
MIMS	-0.56	-0.73
MISEI	-0.41	-0.57
MISM	-0.41	-0.57
McCC	0.71	0.85
NL	0.50	0.61
NLE	0.50	0.60
LLOC	0.55	0.72
NOS	0.52	0.68

After the evaluation of correlations between CoCo and other measures, we proceeded to building regression models. We obtained 65 statistically significant models after log-log transformation of measures. Table IV provides a summary of the most accurate models we found. For each model, the adjusted R^2 determination coefficient is given (obtained after excluding outliers). We also give a few indicators of the accuracy of the models (computed including outliers): MAR is the mean of absolute residuals (i.e., the average absolute prediction error), MMRE is the mean magnitude of relative errors, while MdMRE is the median magnitude of relative errors. MMRE and MdMRE are considered biased indicators: we report them here only as a complement to MAR, which we considered the indicator of accuracy to be taken into account [13].

Note that in addition to the measures listed in Table I, we used also MCC/LLOC, i.e., McCabe's complexity density.

IV. DISCUSSION OF THE RESULTS FROM THE EMPIRICAL STUDY

The results of the correlation tests given in Table III show that CoCo is correlated with all the traditional code measures we considered. Specifically, CoCo is strongly correlated with McCabe's complexity: this is quite noticeable, considering that CoCo was proposed to improve McCabe's complexity.

We can thus answer RQ1 as follows:
Our study shows medium to strong correlations between CoCo and each of the commonly used code measures that we considered. Specifically, CoCo appears most strongly correlated with McCabe's complexity.

The results given in Table IV let us answer RQ2 as follows:
Our study shows that it possible to build models that predict the value of CoCo based on commonly used measures, as well as using Halstead measures and maintainability indexes. Many of the obtained models feature quite good accuracy.

Noticeably, the independent variables that support the most accurate models are McCabe's complexity, the nesting level and the number of logical lines of code. This is hardly surprising, given that elements of MCC and NLE are used in the definition of CoCo. As to LLOC, it is clear that the longer the code, the more decision points it contains (on average),

TABLE IV
MODELS FOUND.

Measures	adjusted R^2	MAR	MMRE	MdMRE
MI, NL	0.81	3.60	0.28	0.20
MIMS, NL	0.81	3.60	0.28	0.20
NLE, LLOC	0.79	3.08	0.25	0.20
HCPL, MI, NLE	0.84	2.96	0.24	0.18
HCPL, MIMS, NLE	0.84	2.96	0.24	0.18
HCPL, NLE, LLOC	0.81	3.04	0.25	0.20
HDIF, MI, NL	0.82	3.65	0.28	0.19
HDIF, MI, NLE	0.84	2.96	0.24	0.19
HDIF, MIMS, NL	0.82	3.65	0.28	0.19
HDIF, MIMS, NLE	0.84	2.96	0.24	0.19
HEFF, MI, NL	0.82	3.72	0.28	0.20
HEFF, MI, NLE	0.84	3.01	0.24	0.19
HEFF, MIMS, NL	0.82	3.72	0.28	0.20
HEFF, MIMS, NLE	0.84	3.01	0.24	0.19
HNDB, MI, NL	0.82	3.72	0.28	0.20
HNDB, MI, NLE	0.84	3.01	0.24	0.19
HNDB, MIMS, NL	0.82	3.72	0.28	0.20
HNDB, MIMS, NLE	0.84	3.01	0.24	0.19
HPL, MI, NLE	0.84	3.03	0.24	0.19
HPL, MIMS, NLE	0.84	3.03	0.24	0.19
HPL, NLE, LLOC	0.82	3.03	0.25	0.20
HPV, MI, NL	0.82	3.77	0.28	0.20
HPV, MI, NLE	0.84	2.95	0.24	0.18
HPV, MIMS, NL	0.82	3.77	0.28	0.20
HPV, MIMS, NLE	0.84	2.95	0.24	0.18
HTRP, MI, NL	0.82	3.72	0.28	0.20
HTRP, MI, NLE	0.84	3.01	0.24	0.19
HTRP, MIMS, NL	0.82	3.72	0.28	0.20
HTRP, MIMS, NLE	0.84	3.01	0.24	0.19
HVOL, MI, NLE	0.84	3.04	0.24	0.19
HVOL, MIMS, NLE	0.84	3.04	0.24	0.19
HVOL, NLE, LLOC	0.82	3.03	0.25	0.20
MI, MIMS, NLE	0.81	3.59	0.26	0.19
MI, NL, NLE	0.81	2.89	0.23	0.18
MI, NLE, LLOC	0.83	3.25	0.25	0.19
MIMS, NL, NLE	0.81	2.89	0.23	0.18
MIMS, NLE, LLOC	0.83	3.25	0.25	0.19
McCC, NLE, LLOC	0.95	1.77	0.15	0.11
McCC, NLE, MCC/LLOC	0.95	1.77	0.15	0.11
NL, NLE, LLOC	0.78	2.99	0.24	0.20
NLE, LLOC, MCC/LLOC	0.95	1.77	0.15	0.11

TABLE V
DESCRIPTIVE STATISTICS OF THE NEW DATASETS.

Project	num. methods	Measure	Mean	st.dev.	Median	Min	Max
ant	3505	CoCo	4.73	7.60	2	1	107
		HPV	34.22	23.07	27	0	188
		MI	101.20	17.91	104	0	136
		McCC	4.40	4.30	3	1	53
		NLE	1.44	1.06	1	0	9
		LLOC	16.19	16.30	11	3	162
tomcat	8050	CoCo	6.47	14.44	3	1	413
		HPV	39.64	30.42	31	0	651
		MI	96.75	18.76	100	-14	150
		McCC	5.12	6.77	3	1	154
		NLE	1.69	1.12	1	0	13
		LLOC	19.78	24.59	12	1	612

hence we can expect also LLOC to contribute to CoCo. In fact, the relationship between CoCo and lines of code was already observed [14].

In conclusion, our study shows that CoCo does not seem to convey more knowledge than sets of properly chosen traditional code measures, like MCC, NLE and LLOC.

V. EXPERIMENTAL VERIFICATION OF FORMER RESULTS

In this section we report the results of a second empirical study, which provides further evidence that confirms the findings given above.

A. The verification dataset

To verify the results from [1] we selected two large open source Java projects, namely `ant 1.10.12` and `tomcat 10.0.0-M10`. The descriptive statistics of the two projects' code are given in Table V.

B. Verifying correlation of CoCo with traditional measures

The first verification activity we carried out consisted in testing the correlation between CoCo and traditional measures. To this end, we computed Kendall's rank correlation coefficient τ [10] and Spearman's rank correlation coefficient ρ [11], as was done in [1].

TABLE VI
RESULTS OF THE NEW CORRELATION TESTS.

Measure	τ	ρ
HCPL	0.47	0.63
HDIF	0.41	0.56
HEFF	0.49	0.66
HNDB	0.49	0.66
HPL	0.52	0.69
HPV	0.47	0.64
HTRP	0.49	0.66
HVOL	0.52	0.69
MI	−0.58	−0.75
MIMS	−0.58	−0.75
MISEI	−0.43	−0.58
MISM	−0.43	−0.58
McCC	0.71	0.86
NL	0.50	0.61
NLE	0.50	0.61
LLOC	0.58	0.75
NOS	0.55	0.72

The results we obtained are given in Table VI. In all cases, the p-value was less than 10^{-3}.

The results in Table VI fully confirm the previous results reported in Table III. Specifically, in `ant` and `tomcat`, the correlation between CoCo and traditional measures appears just a bit stronger. However, the differences in both τ and ρ are so small that they fully confirm the reliability of the correlation coefficients given in [1].

C. Evaluation of the accuracy of CoCo models

As we mentioned in Section III-C, several models of CoCo as a function of traditional measures were found. A selection is given in Table IV, where accuracy indications obtained via classical 10-time 10-fold cross-validation are also reported.

We can now use the new dataset containing measures from `ant` and `tomcat` to test the accuracy of those models. If we achieve accurate predictions, that means that the models obtained from the original dataset represent a fairly general relationship between CoCo and traditional measures.

The results from the original analysis suggest that the following two models are the most accurate ones:

$$CoCo = 0.6408 McCC^{0.8105} NLE^{0.6404} LLOC^{0.1552} \quad (1)$$

$$CoCo = 0.6515 \left(\frac{McCC}{LLOC}\right)^{0.8440} NLE^{0.6392} LLOC^{0.9651} \quad (2)$$

So, we used models (1) and (2) to estimate the CoCo of ant and tomcats methods, based on McCC, NLE and LLOC of those applications' methods.

When using model (1) we obtained the absolute error illustrated by the boxplot in Figure 1 (outliers not shown). The blue diamond represents the MAR.

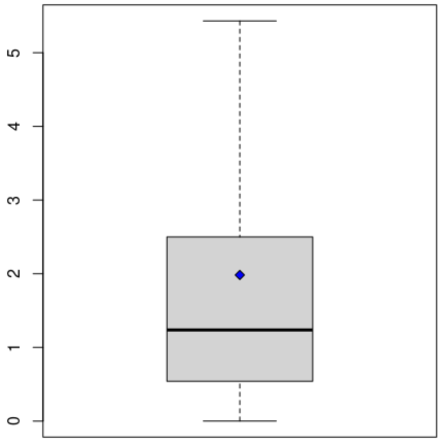

Fig. 1. Distribution of absolute errors of estimates obtained via (1), without outliers.

Figure 2 shows the distribution of absolute relative errors (including outliers). It can be seen that the greatest majority of estimates is within 5% of the actual CoCo.

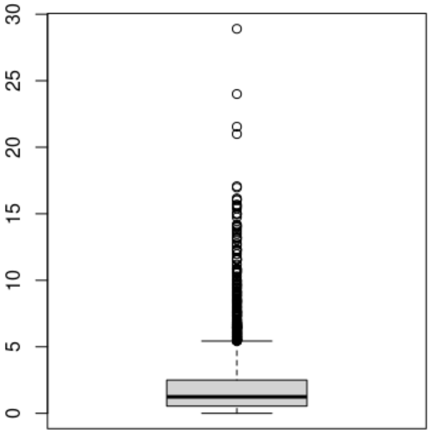

Fig. 2. Distribution of absolute relative errors of estimates obtained via (1), with outliers.

Figure 3 compares actual CoCo values with estimates obtained via (1). The blue straight line represents the perfect prediction.

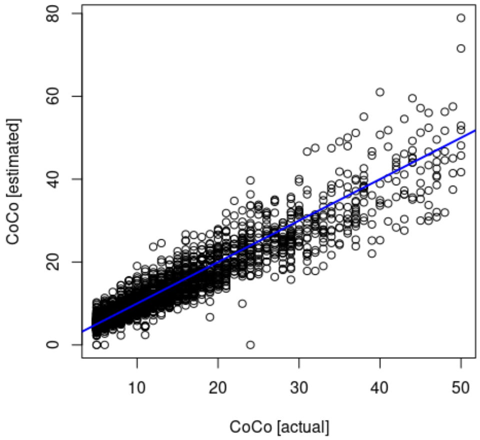

Fig. 3. Comparison of actual CoCo with estimates obtained via (1).

When using model (2) we obtained the absolute error illustrated by the boxplot in Figure 5 (outliers not shown).

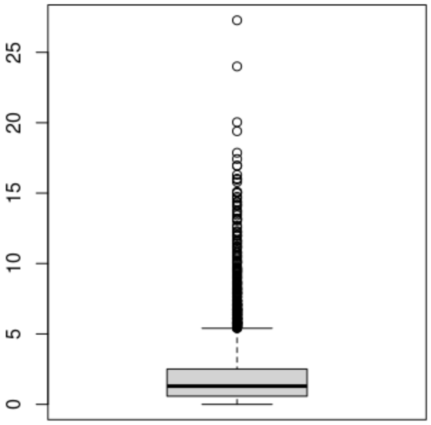

Fig. 4. Distribution of absolute relative errors of estimates obtained via (2), with outliers.

Figure 4 shows the distribution of absolute relative errors (including outliers). It can be seen that also in this case the greatest majority of estimates is within 5% of the actual CoCo.

Figure 6 compares actual CoCo values with estimates obtained via (2).

Overall, the evaluation of models (1) and (2) via the ant and tomcat dataset yielded results extremely close to those obtained with the 10-times 10-fold cross validation, as shown in Table VII, where column "10-times 10-fold Xval" reports the data already given in Table IV, concerning the accuracy evaluated on the original dataset, while column "ant and tomcat prediction" provides the accuracy indicators for the predictions obtained applying (1) and (2) to the ant and tomcat dataset.

In conclusion, the models of CoCo confirm that there is a strong correlation between CoCo and traditional measures, and that it is possible to get a quite accurate estimate of CoCo based on models that have traditional measures as independent variables.

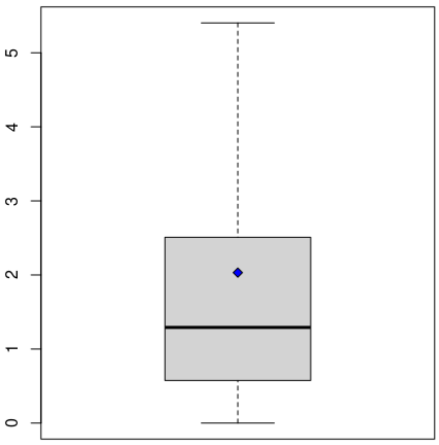

Fig. 5. Distribution of absolute errors of estimates obtained via (2), without outliers.

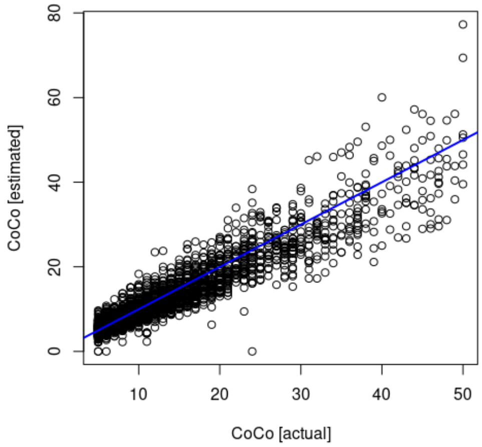

Fig. 6. Comparison of actual CoCo with estimates obtained via (2).

VI. CoCo ESTIMATION USING MACHINE LEARNING

Previous sections showed that CoCo does not seem to add much information with respect to traditional measures (especially McCabe complexity, NLE and logical LOC). In this section we explore whether it is possible to uncover even stronger relationships by means of machine learning (ML), or if the usage of ML just confirms the previous findings obtained via ordinary least squares (OLS) regression.

We proceeded through the following steps:

1) We used the original dataset to build a model of CoCo vs. McCC, NLE and LLOC. To this end, we used Support Vector Regression (SVR) with radial kernel. The computations were carried out via the R language and programming environment [15], using the `e1071` library. Parameters γ, ϵ and cost were trained to minimize the MAR (Mean Absolute Residual) via repeated application of a 5-fold cross validation sampling method (to this end, the `tuning` function of the `e1071` library was used).
2) We used the resulting model to estimate CoCo for each method of `ant` and `tomcat`.
3) We evaluated the accuracy of the obtained estimates, and compared then with the estimates obtained via OLS regression (as described in Section V-C).

TABLE VII
ACCURACY OF MODELS (1) AND (2).

	10-times 10-fold Xval			`ant` and `tomcat` prediction		
model	MAR	MMRE	MdMRE	MAR	MMRE	MdMRE
(1)	1.77	0.15	0.11	1.98	0.16	0.13
(2)	1.77	0.15	0.11	2.03	0.16	0.14

Figure 7 shows the distribution of estimation errors (without outliers). It is easy to see that the greatest majority of estimates is quite correct; namely over 50% of the estimation errors are in [-1, +1] range.

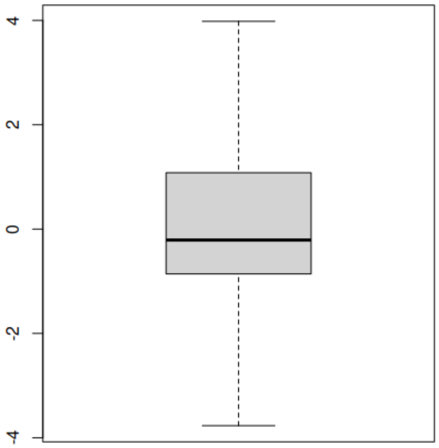

Fig. 7. Distribution of errors of estimates obtained via ML (no outliers).

Figure 8 shows the distribution of absolute estimation errors (without outliers). The blue diamond represents the MAR. It is easy to see that over 75% of the estimation errors have magnitude less than 2.5.

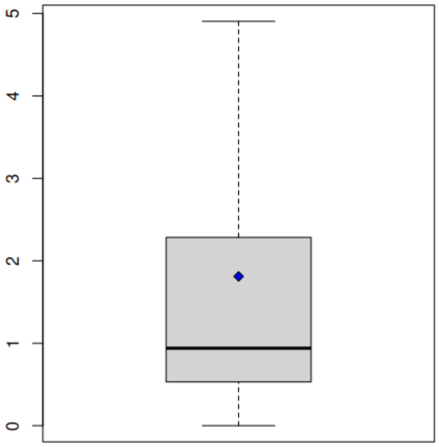

Fig. 8. Distribution of absolute errors of estimates obtained via ML (no outliers).

Figure 9 compares the estimates with the actual CoCo values. In can be seen that most estimates are very close to the corresponding actual values. However, in a few cases, the estimates are relatively far from the actual value.

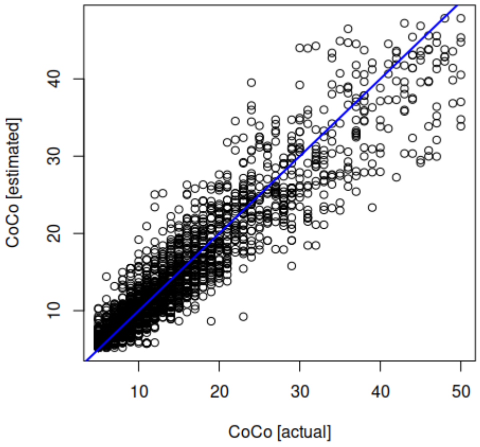

Fig. 9. Comparison of actual CoCo measures and estimates.

The summary of accuracy indicators is:

- MAR= 1.8
- MMRE= 0.145
- MdMRE= 0.115

So, the accuracy of ML models is quite close to the accuracy of OLS models, as is apparent by comparing the values above with those in Table VII.

VII. Threats to Validity

Concerning the application of traditional measures, we used a state-of-the-art tool (SourceMeter), which is widely used and mature, therefore we do not see any threat on this side. CoCo was measured using an ad-hoc tool that was built based on the specifications of CoCo [2]. This tool was thoroughly tested using `SonarQube` [16] as a reference, therefore we are reasonably sure that it provides correct measures. However, when joining the data from SourceMeter with the data from our tool, we were not able to always match methods identifiers, because the two tools reported slightly different descriptions of methods' names, parameters, etc. We just dropped the methods' data for which no sure match could be found: in this way, we lost less than 2% of the measures. Since the lost measures depend on characteristics that have nothing to do with the properties of code being measured, they can be considered a random subset, which can hardly affect the outcomes of the study.

Concerning the external validity of the study, as with most empirical studies in the Software Engineering area, we cannot be completely sure about the generalizability of results. However, the dataset used was large enough, and the selected software projects represent a reasonable variety of application types. In addition, the verification performed using a new dataset fully confirmed the original results [1]. Also the usage of SVR to evaluate the correlation of CoCo with traditional measure confirmed the original findings. We can thus conclude that the presented results appear reliable and reasonably general.

VIII. Related Work

Campbell performed an investigation of the developers' reaction to the introduction of CoCo in the measurement and analysis tool `SonarCloud` [17]. In an analysis of 22 open-source projects, she assessed whether a development team "accepted" the measure, based on whether they fixed code areas indicated by the tool as characterized by high CoCo. Around 77% of developers expressed acceptance of the measure.

An objective validation of the CoCo measure was performed by Muñoz Barón et al. [4]. They retrieved data sets from published studies that measured the understandability of source code from the perspective of human developers. They collected the data concerning various aspects of understandability, as well as the code snippets used in the experiments. They used `SonarQube` [16] to obtain the CoCo measure for each source code snippet. Then, they computed the correlation of CoCo with the measures of various aspects of understandability. Muñoz Barón et al. computed the correlation between CoCo and various aspects of understandability for each of the 10 experiments reported in the selected papers, as well as a summary obtained via meta-analysis. Muñoz Barón et al. concluded that CoCo correlates moderately with some of the considered understandability aspects.

The paper mentioned above dealt with evaluating the effectiveness of CoCo (a measure of internal code properties) as an indicator of understandability (an external code property). To our knowledge, nobody performed an analysis dealing with how internal code properties only are correlated with CoCo.

Nonetheless, CoCo has been used in some evaluations. CoCo is provided by `SonarQube` [16] together with many other measures and indicators, so some researchers that used `SonarQube` to collect code measures ended up using CoCo together with other measures. Among the papers that have used CoCo are the following.

Kozik et al. [14] developed a framework for analyzing software quality dependence on code measures and other data. Using the framework they found that CoCo affects the analyzability and adaptability of code.

Papadopoulos et al. [18] investigated the interrelation between design time quality metrics and runtime quality metrics, such as cache misses, memory accesses, memory footprint and CPU cycles. Papadopoulos et al. observed a trade-off between performance/energy consumption and cognitive complexity. However, having used CoCo as the only design time quality metric, it is unknown whether the same kind of trade-off would be observed with respect to other design-time metrics, like McCabe's complexity, for instance. Our study suggests that this doubt is well funded, i.e., a trade-off involving performance/energy consumption and design-time metrics like McCabe's complexity could very well exist.

Crespo et al. [19] used both the Cognitive complexity rate (defined as CoCo/LOC) and the Cyclomatic complexity rate (defined as McCabe complexity/LOC) as part of an assessment strategy concerning technical debt in an educational context. They found that the Cognitive complexity rate and the Cyclomatic complexity rate provide the same results, or lack of results, actually. Given the strong correlation that we observed between CoCo and McCabe's complexity, the result by Crespo et al. is not surprising.

IX. Conclusions

The "Cognitive Complexity" measure (CoCo throughout the paper) was introduced with the aim of improving the ability to detect code that is difficult to understand and maintain [2]. Rather than a direct measure, CoCo is an indicator, whose definition accounts for a few characteristics of source code. Among these characteristics are the number of decision points (e.g., if, for, while and switch statements) and the level of nesting of control statements.

When CoCo was proposed, no evaluations were published concerning the relationship between CoCo and traditional measures that directly address the aforementioned characteristics of code. In this paper, we have reported about empirical studies aiming at evaluating the correlation between CoCo and several traditional measures, including those addressing the same characteristics of code taken into account by CoCo. To this end, we measured a few open source projects' code, obtaining the measures of 3,610 methods. We then performed statistical analysis using both correlation tests (namely, Kendall's and Spearman's rank correlation coefficients), regression analysis and machine learning.

We found that CoCo appears strongly correlated to McCabe's complexity and slightly less strongly correlated to several other code measures. We found several regression models of CoCo as a function of traditional measures. Not surprisingly, one of the most accurate models involves McCabe's complexity, NLE (Nesting Level Else-If) and LLOC (the number of logical lines of code) as independent variables. Considering that the most accurate models have MAR=1.7, while the mean CoCo is 12, we may conclude that—at least for the considered software projects—CoCo does not appear to convey additional information with respect to traditional measures.

Cross-dataset validation confirmed the initial results, as did the models obtained using Support Vector Regression.

In conclusion, the study reported here casts the doubt that CoCo does not provide appreciable new knowledge with respect to the measures of code that are traditionally associated with the notion of complexity.

Concerning future work, it can be noticed that the work reported here concerns exclusively relationships among internal measures. It could be interesting to evaluate how well the studied internal measures (CoCo and traditional complexity and size measure) correlate with external qualities. Specifically, we plan to repeat previous studies [20], [21] using CoCo together with (or alternatively to) other code measures.

Acknowledgment

The work reported here was partly supported by Fondo per la Ricerca di Ateneo, Università degli Studi dell'Insubria. The author thanks Anatoliy Roshka for developing the tool that was used to measure CoCo.

References

[1] L. Lavazza, "An Empirical Study of the Correlation of Cognitive Complexity-related Code Measures," in Proceedings of The Sixteenth International Conference on Software Engineering Advances – ICSEA, 2021.

[2] G. A. Campbell, "Cognitive complexity - a new way of measuring understandability," https://www.sonarsource.com/docs/CognitiveComplexity.pdf, 2018, [Online; accessed 7-September-2021].

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

[4] M. M. Barón, M. Wyrich, and S. Wagner, "An empirical validation of cognitive complexity as a measure of source code understandability," in Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2020, pp. 1–12.

[5] N. Fenton and J. Bieman, Software metrics: a rigorous and practical approach. CRC press, 2014.

[6] T. J. McCabe, "A complexity measure," IEEE Transactions on software Engineering, no. 4, 1976, pp. 308–320.

[7] "SourceMeter," https://www.sourcemeter.com/, [Online; accessed 7-September-2021].

[8] M. H. Halstead, Elements of software science. Elsevier North-Holland, 1977.

[9] P. Oman and J. Hagemeister, "Metrics for assessing a software system's maintainability," in Proceedings Conference on Software Maintenance 1992. IEEE Computer Society, 1992, pp. 337–338.

[10] M. G. Kendall, "Rank and product-moment correlation," Biometrika, 1949, pp. 177–193.

[11] C. Spearman, "The proof and measurement of association between two things," The American journal of psychology, vol. 100, no. 3/4, 1987, pp. 441–471.

[12] R. D. Cook, "Detection of influential observation in linear regression," Technometrics.

[13] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," Information and Software Technology, vol. 54, no. 8, 2012, pp. 820–827.

[14] R. Kozik, M. Choraś, D. Puchalski, and R. Renk, "Q-rapids framework for advanced data analysis to improve rapid software development," Journal of Ambient Intelligence and Humanized Computing, vol. 10, no. 5, 2019, pp. 1927–1936.

[15] R core team, "R: a language and environment for statistical computing," 2015.

[16] "SonarQube," https://www.sonarqube.org/, [Online; accessed 7-September-2021].

[17] G. A. Campbell, "Cognitive complexity: An overview and evaluation," in Proceedings of the 2018 International Conference on Technical Debt, 2018, pp. 57–58.

[18] L. Papadopoulos, C. Marantos, G. Digkas, A. Ampatzoglou, A. Chatzigeorgiou, and D. Soudris, "Interrelations between software quality metrics, performance and energy consumption in embedded applications," in Proceedings of the 21st International Workshop on software and compilers for embedded systems, 2018, pp. 62–65.

[19] Y. Crespo, A. Gonzalez-Escribano, and M. Piattini, "Carrot and stick approaches revisited when managing technical debt in an educational context," arXiv preprint arXiv:2104.08993, 2021.

[20] V. Del Bianco, L. Lavazza, S. Morasca, D. Taibi, and D. Tosi, "An investigation of the users' perception of OSS quality," in IFIP International Conference on Open Source Systems. Springer, 2010, pp. 15–28.

[21] ——, "The QualiSPo approach to OSS product quality evaluation," in Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development, 2010, pp. 23–28.