Comparing Closed-Source and Open-Source Code Static Measures

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

***Abstract*—Most software engineering empirical studies are based on the analysis of open-source code. The reason is that open-source code is readily available, while usually software development organizations do not give access to their code, not even when the purpose is research and the code itself will not be disclosed. As a consequence, the corpus of empirical knowledge is related almost exclusively to open-source software. This poses a quite important question: do the conclusions we draw from the analysis of open-source code apply to closed-source code as well? In this paper, a comparison of open-source and closed-source code is performed, to provide some preliminary answers to the question. Specifically, the goal of the paper is to evaluate whether static code measures from open-source code are similar to those obtained from closed-source code. To this end, an empirical study was performed, involving closed-source code from two organizations and open-source code from a few different projects. The most popular static code measures were collected using a commercial tool, and compared. The study shows that open-source code measures appear similar to the measures obtained from industrial closed-source code. However, we must note that the study reported here involved just a few industrial projects' measures. Therefore, replications of the work presented here would be very useful.**

Keywords-software code measures; static code measures; open-source code; closed-source code.

I. Introduction

Software development organizations make their code available to researchers very rarely. This is due to their need for preserving the competitive advantage deriving from code ownership. As a consequence, the great majority of the empirical studies involving source code analyze open-source code, which is freely available. The conclusions reached by these studies are expected to apply to all code, including industrial closed-source code. However, the generalizability of studies based on open-source software relies on the assumption that closed-source software is "similar" to open-source software. Specifically, it is expected that the measures of open-source code are representative of closed-source software as well.

This paper describes an empirical study that aims at verifying if and to what extent code measures of open- and closed-source projects are similar. To this end, we measured a set of industrial closed-source projects and a set of open-source projects and compared the resulting measures. While in a previous paper [1] we considered only method-level measures of Java code, in this paper we extend the analysis to class-level measures.

Based on our results, there are no major differences among the measures collected from industrial and open-source projects. The study reported here has the merit to provide some objective evidence that studying open-source projects as representative of closed-source projects is sound.

In this study, the investigation is limited to static code measures for Java projects. Specifically, we consider the code metrics that are most frequently used in the research literature and the software industry, which can be easily obtained via any state-of-the-art tool.

The paper is structured as follows. Section II describes the static code measures investigated in this study. Section III describes the empirical study; results are given in Sections IV and V for method and class measures, respectively Section VI discusses the results obtained by the study. 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

Since the first high-level programming languages were introduced, several measures were proposed, to represent the possibly relevant characteristics of code [2]. For instance, the size of a software module is usually measured in terms of Lines Of Code (LOC), while McCabe Complexity (also known as Cyclomatic Complexity) [3] 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 [4] were proposed to recognize poor software design. For instance, modules with high levels of coupling are supposed to be associated with difficult maintenance.

We have considered some of the most popular method-level measures (listed in Table I) and class-level measures (listed in Table II). All the measures mentioned in Tables I and II were collected via the SourceMeter tool [5]. Interested readers can find additional information concerning the definition and meaning of the selected metrics in the documentation of SourceMeter.

Halstead proposed several code metrics [6], based on the total number of occurrences of operators N_1, the total number of occurrences of operands N_2, the number of distinct operators η_1 and the number of distinct operands η_2. SourceMeter does not provide the individual measures of N_1, N_2, η_1 and η_2; instead, it provides Halstead Program Length (HPL), which is defined as $HPL = N_1 + N_2$, and Halstead Program Vocabulary (HPV), which is defined as $HPV = \eta_1 + \eta_2$. Halstead Volume (HVOL) is defined as $HVOL = (N_1 + N_2) * log_2(\eta_1 + \eta_2)$; Halstead Calculated Program Length (HCPL) is defined as

TABLE I
METHOD MEASURES COLLECTED VIA SOURCEMETER.

Metric name	Abbreviation
Lines of Code	LOC
Logical Lines of Code	LLOC
Halstead Program Length	HPL
Halstead Program Vocabulary	HPV
Halstead Calculated Program Length	HCPL
Halstead Volume	HVOL
Maintainability Index (Original version)	MI
McCabe's Cyclomatic Complexity	McCC

$HCPL = \eta_1 * log_2(\eta_1) + \eta_2 * log_2(\eta_2)$. McCabe's complexity (McCC) is used to indicate the complexity of a program, being the number of linearly independent paths through a program's source code [3]. The Maintainability Index (MI) [7] is defined as $MI = 171 - 5.2 * ln(HVOL) - 0.23 * (McCC) - 16.2 * ln(LLOC)$, where LLOC is the number of Logical LOC, i.e., the number of non-empty and non-comment code lines.

At the class level, we considered size metrics and the metrics proposed by Chidamber and Kemerer [4].

TABLE II
CLASS MEASURES COLLECTED VIA SOURCEMETER.

Metric name	Abbreviation
Lines of Code	LOC
Logical Lines of Code	LLOC
Coupling between Objects	CBO
Response for a Class	RFC
Weighted Methods per Class	WMC
Lack of Cohesion in Methods	LCOM5
Depth of Inheritance	DIT
Number of Children	NOC

Besides LOC and LLOC, which have the same meaning and definition as the corresponding metrics used for methods, the other class-level metrics are briefly described below [8].

WMC (Weighted methods per class) was defined as the sum of the complexities of its methods. As a measure of complexity SourceMeter assigns 1 to each method, hence WMC is equal to the number of methods in the class.

CBO (Coupling between object classes) represents the number of classes coupled to a given class (efferent couplings and afferent couplings) through method calls, field accesses, inheritance, arguments, return types, and exceptions.

RFC (Response for a Class) measures the number of different methods that can be executed when a method is invoked for that object.

LCOM (Lack of cohesion in methods) measures the lack of cohesion and computes into how many coherent classes the class could be split. The original definition of LCOM [4] was criticized because it depended on the number of methods in the considered class. A few alternative definitions were given. We use the one supported by SourceMeter, which is computed by taking a non-directed graph, where the nodes are the implemented local methods of the class and there is an edge between the two nodes if and only if a common (local or inherited) attribute or abstract method is used or a method invokes another. The value of the metric is the number of connected components in the graph not counting those, which contain only constructors, destructors, getters, or setters.

DIT (Depth of Inheritance Tree) provides for each class a measure of the inheritance levels from the hierarchy top class. In Java, where all classes inherit Object, the minimum value of DIT is 1.

NOC (Number of Children) measures the number of immediate descendants of the class.

III. THE EMPIRICAL STUDY

The empirical study involved closed-source and open-source Java programs. This code was measured, and the collected data were analyzed via well established 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 Sections IV and V.

A. The Dataset

As already mentioned, obtaining source code from software industries is not easy. Therefore, the closed-source code analyzed within the study is a convenience sample: it is the code that we were able to obtain from industrial developers. The open-source code analyzed within the study is the open-source code used within or together with the analyzed industrial projects. This guarantees a sort of "homogeneity" of code with respect to the required quality.

The open-source projects that supplied the code for the study are: Log4J [9], JCaptcha [10], Pdfbox [11], JasperReports (abbreviated JReports where necessary) [12], Hibernate [13].

Because of confidentiality reasons, the names of the industrial projects that supplied the code to be measured are not given: these projects are named Industrial1, Industrial2, Industrial3 (abbreviated Ind1, Ind2 and Ind3 where necessary). Ind1 and Ind2 are client and contract management systems from a large service company, Ind3 is the back-end of a web application. All of the industrial projects aimed to develop software supporting the main business of the owner companies, i.e., none of the considered projects delivered a product to be sold on the market. Also, all projects were developed by external software houses on behalf of the owner companies. Because of confidentiality reasons, the code and the raw measures are not available.

Table III gives some descriptive statistics of the considered projects.

TABLE III
DESCRIPTIVE STATISTICS OF THE DATASETS.

	Number	LOC	LOC per file			
	of files	total	mean	sd	median	range
Ind1	1507	202299	134	268	91	[1–6851]
Ind2	280	56419	201	286	93	[3–2336]
Ind3	1323	250193	189	307	100	[6–3644]
Log4J	1067	126354	118	121	80	[20–1357]
JCaptcha	248	25292	102	99	75	[16–691]
Pdfbox	1215	252158	208	251	125	[21–2966]
JReports	3177	533008	168	285	89	[27–4398]
Hibernate	2392	236527	99	127	63	[9–2146]

Table III provides, for each analyzed project, the number of files, the total number of LOC, and the mean, standard deviation, median and range of the LOC per file.

B. The Method

The first phase of the study consisted in measuring the code. We used SourceMeter [5] to obtain the measures.

When analyzing the measures concerning methods, we decided to exclude from the study all the methods having unitary McCabe complexity, i.e., the methods that contain no decision points, since those methods would bias the results. In fact, these methods are quite numerous (since they include all the setters and getters) and very small (the excluded methods have mean and median LOC in the [3,6] range).

After removing the methods having unitary McCabe complexity, we got the dataset whose descriptive statistics are given in Table IV.

TABLE IV
DESCRIPTIVE STATISTICS OF THE DATASETS' METHODS, AFTER REMOVING METHODS WITH UNITARY MCCABE COMPLEXITY.

	Num. methods	LOC total	LOC per method mean	sd	median	range
Ind1	1342	32654	24	38	15	[3, 626]
Ind2	703	17099	24	25	16	[3, 197]
Ind3	3339	127170	38	61	21	[3, 1272]
Log4J	1729	29948	17	17	12	[3, 176]
JCaptcha	362	6386	18	15	13	[3, 100]
Pdfbox	3738	92679	25	26	16	[3, 380]
JReports	6815	180104	26	31	17	[3, 453]
Hibernate	2746	46505	17	15	12	[3, 221]

Table V gives the descriptive statistics of the datasets with respect to classes.

TABLE V
DESCRIPTIVE STATISTICS OF THE DATASETS' CLASSES.

	Num. classes	LOC total	LOC per class mean	sd	median	range
Ind1	1159	151726	131	293	65	[7, 6735]
Ind2	247	48261	195	273	88	[3, 2268]
Ind3	1389	222711	160	292	75	[2, 3639]
Log4J	1210	85242	70	93	39	[1, 1218]
JCaptcha	240	17941	75	93	50	[2, 648]
Pdfbox	1408	206029	146	225	72	[2, 2894]
JReports	2895	371364	128	262	55	[3, 4088]
Hibernate	2832	166082	59	95	32	[2, 2039]

Finally, we compared the collected measures. To this end, we provide a visual representation of the data via boxplots that describe the distributions, the mean and the median of the measures collected from each project. We also performed statistical analysis:

1) We performed a Kruskal-Wallis test for all the considered metrics, since the conditions for performing ANOVA tests did not hold. As a result, we obtained that, for all metrics, projects are not all equivalent with respect to the considered measure.
2) To explore in detail the differences among projects, we performed Wilcoxon rank sum tests for all project pairs, for all the considered metrics.
3) When a Wilcoxon rank sum test excluded that the measures are equivalent, we evaluated the effect size via Hedge's g.

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

IV. RESULTS OF METHOD-LEVEL MEASUREMENTS

This section reports the data collected from methods, grouped according to the type of property being measured.

A. Size Measures

Boxplots of LOC measures are given in Figure 1: for the sake of readability, Figure 1 provides also a view without outliers. The mean values are represented as blue diamonds.

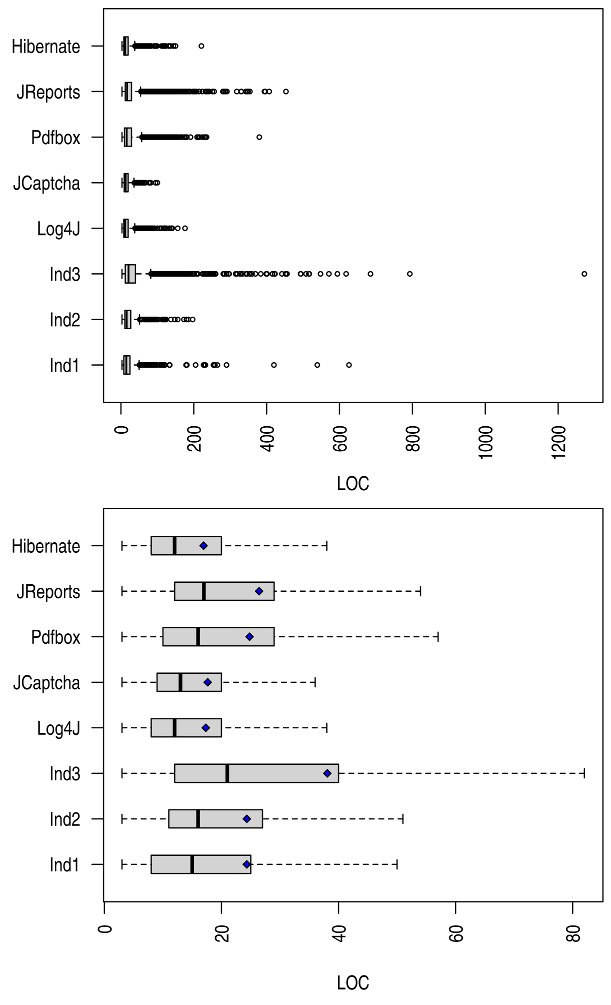

Figure 1. Boxplots of size (measured in LoC) distributions, with (top) and without (bottom) outliers.

The results of the Wilcoxon rank sum tests and Hedges's g evaluations concerning LOC are given in Table VI. Specifically, a cell includes symbol '=' if the Wilcoxon rank sum test could not exclude that the considered measures are equivalent; otherwise, a cell includes one of the symbols 'n,' 's,' 'm' for negligible, small and medium effect size, respectively (in no case a large effect size was found).

TABLE VI
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR METHODS' LOC.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	s	s	n	n	n	s
Ind2	n	–	s	s	s	=	n	s
Ind3	s	s	–	s	s	s	s	s
Log4J	s	s	s	–	n	s	s	n
JCaptcha	n	s	s	n	–	s	s	n
Pdfbox	n	=	s	s	s	–	n	s
JReports	n	n	s	s	s	n	–	s
Hibernate	s	s	s	n	n	s	s	–

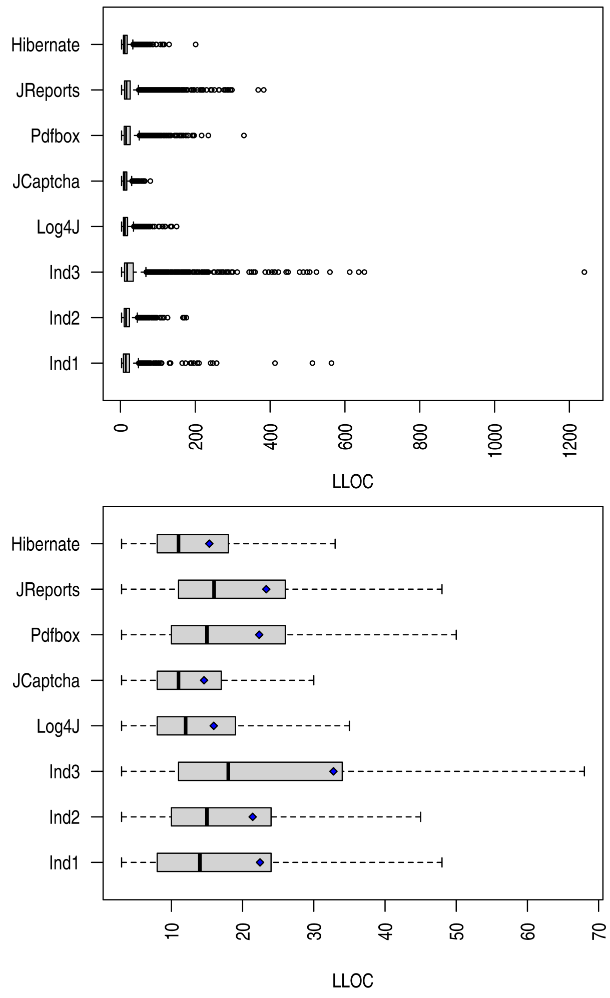

Figure 2. Boxplots of size (measured in LLoC) distributions, with (top) and without (bottom) outliers.

Boxplots of LLOC measures are given in Figure 2.

The results of the Wilcoxon rank sum tests and Hedges's g evaluations for LLOC measures are given in Table VII.

B. Complexity

Boxplots of McCabe cyclomatic complexity measures are given in Figure 3. For the sake of readability, Figure 4 provides the same data, excluding outliers.

The results of the Wilcoxon rank sum tests and Hedges's g evaluations are given in Table VIII.

TABLE VII
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR LLOC.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	s	s	s	n	n	s
Ind2	n	–	s	s	s	n	n	s
Ind3	s	s	–	s	s	s	s	s
Log4J	s	s	s	–	n	s	s	=
JCaptcha	s	s	s	n	–	s	s	n
Pdfbox	n	n	s	s	s	–	n	s
JReports	n	n	s	s	s	n	–	s
Hibernate	s	s	s	=	n	s	s	–

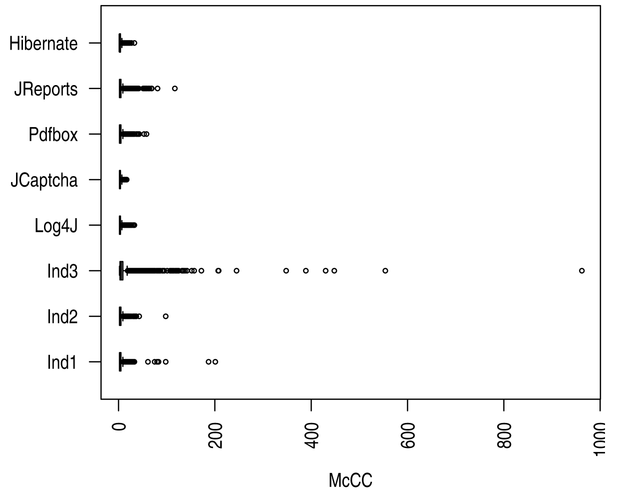

Figure 3. Boxplots of McCabe cyclomatic complexity distributions.

C. Maintainability

Maintainability is measured via the Maintainability Index (MI) [7].

Boxplots of MI measures are given in Figure 5. For the sake of readability, Figure 6 provides the same data, excluding outliers.

The results of the Wilcoxon rank sum tests and Hedges's g evaluations are given in Table IX.

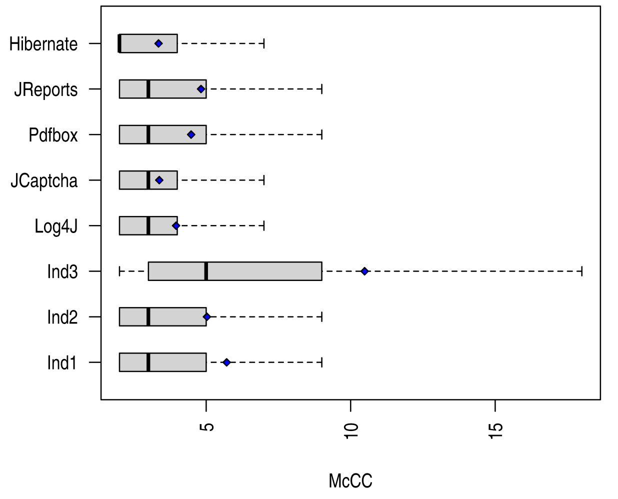

Figure 4. Boxplots of McCabe cyclomatic complexity distributions. Outliers omitted.

TABLE VIII
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR MCCABE COMPLEXITY.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	n	s	s	n	=	s
Ind2	n	–	s	s	s	=	n	s
Ind3	n	s	–	s	s	s	s	s
Log4J	s	s	s	–	n	n	n	s
JCaptcha	s	s	s	n	–	s	s	n
Pdfbox	n	=	s	n	s	–	n	s
JReports	=	n	s	n	s	n	–	s
Hibernate	s	s	s	s	n	s	s	–

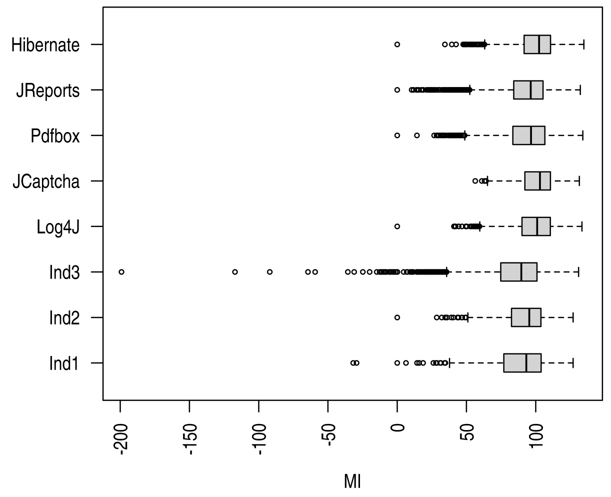

Figure 5. Boxplots of Maintainability Index MI distributions.

D. Halstead Measures

Halstead identified measurable properties of software in analogy with the measurable properties of matter [6]. Among these properties is the volume, measured via the Halstead Volume (HVOL). Boxplots of HVOL measures are given in Figure 7. For the sake of readability, Figure 8 provides the same data, excluding outliers. The results of the Wilcoxon rank sum tests and Hedges's g evaluations are given in Table X.

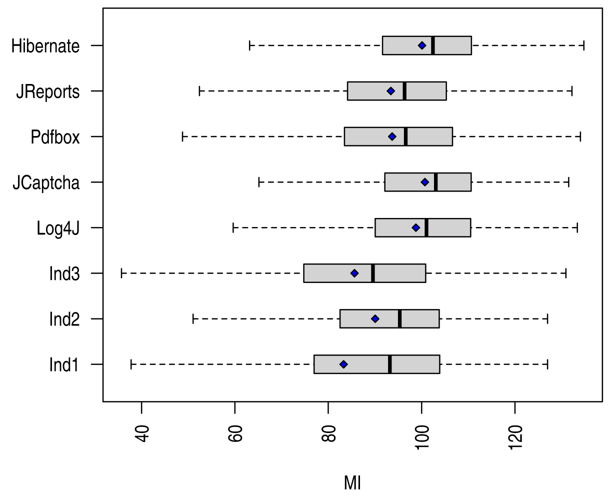

Figure 6. Boxplots of Maintainability Index MI distributions. Outliers omitted.

TABLE IX
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR THE MAINTAINABILITY INDEX (MI).

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	s	n	m	m	s	s	m
Ind2	s	–	n	s	m	n	n	m
Ind3	n	n	–	m	m	s	s	m
Log4J	m	s	m	–	n	s	s	n
JCaptcha	m	m	m	n	–	s	s	=
Pdfbox	s	n	s	s	s	–	n	s
JReports	s	n	s	s	s	n	–	s
Hibernate	m	m	m	n	=	s	s	–

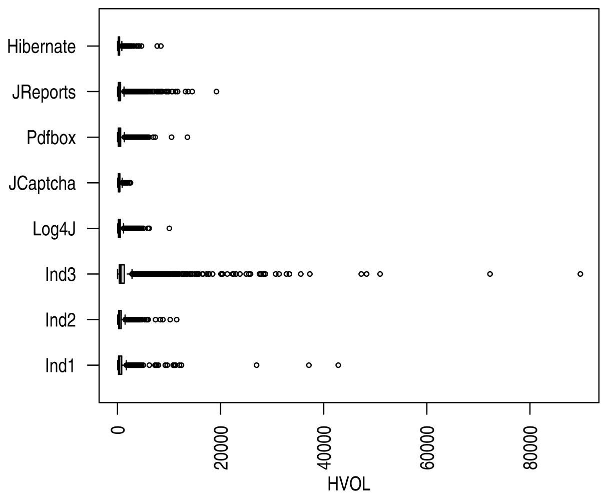

Figure 7. Halstead volume distributions.

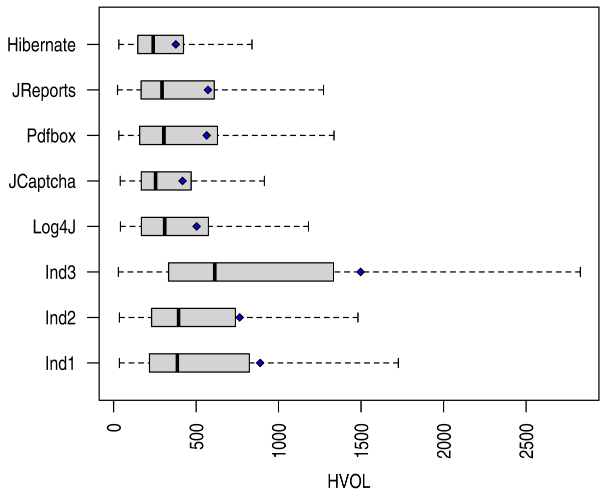

Figure 8. Halstead volume distributions. Outliers omitted.

Boxplots of Halstead Calculated Program Length (HCPL) measures are given in Figure 9. For the sake of readability, Figure 10 provides the same data, excluding outliers. The results of the Wilcoxon rank sum tests and Hedges's g evaluations are given in Table XI.

TABLE X
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR THE HALSTEAD VOLUME.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	=	n	s	s	s	s	s
Ind2	=	–	s	s	s	s	s	m
Ind3	n	s	–	s	s	s	s	s
Log4J	s	s	s	–	n	n	=	s
JCaptcha	s	s	s	n	–	n	n	n
Pdfbox	s	s	s	n	n	–	n	s
JReports	s	s	s	=	n	n	–	s
Hibernate	s	m	s	s	n	s	s	–

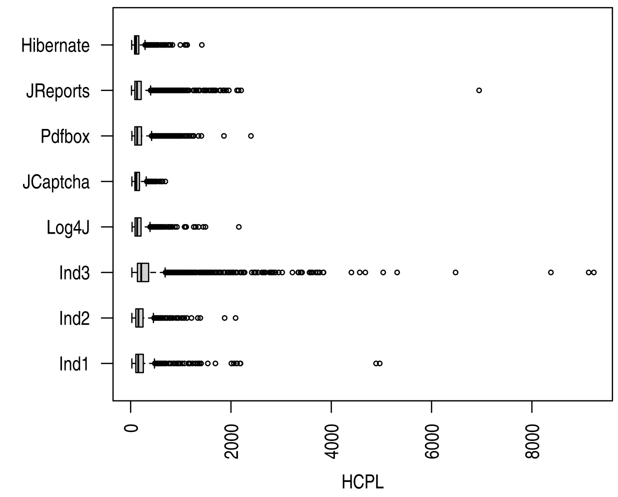

Figure 9. Halstead computed program length distributions.

V. RESULTS OF CLASS-LEVEL MEASUREMENTS

This section reports the data collected from classes, grouped according to the type of property being measured.

A. Size Measures

Boxplots of LOC and LLOC measures are given in Figure 11.

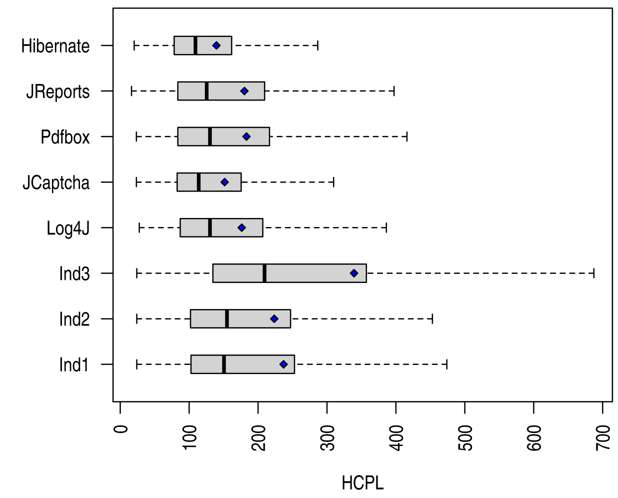

Figure 10. Halstead computed program length distributions. Outliers omitted.

TABLE XI
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR THE HALSTEAD COMPUTED PROGRAM LENGTH.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	=	s	s	s	s	s	m
Ind2	=	–	s	s	s	s	s	m
Ind3	s	s	–	s	s	s	s	m
Log4J	s	s	s	–	n	=	n	s
JCaptcha	s	s	s	n	–	n	n	n
Pdfbox	s	s	s	=	n	–	n	s
JReports	s	s	s	n	n	n	–	s
Hibernate	m	m	m	s	n	s	s	–

The results of the Wilcoxon rank sum tests and Hedges's g evaluations are given in Tables XII and XIII.

TABLE XII
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR CLASS LOC.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	s	s	n	n	n	s
Ind2	n	–	s	s	s	=	n	s
Ind3	s	s	–	s	s	s	s	s
Log4J	s	s	s	–	n	s	s	n
JCaptcha	n	s	s	n	–	s	s	n
Pdfbox	n	=	s	s	s	–	n	s
JReports	n	n	s	s	s	n	–	s
Hibernate	s	s	s	n	n	s	s	–

TABLE XIII
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR CLASS LLOC.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	s	s	s	n	n	s
Ind2	n	–	s	s	s	n	n	s
Ind3	s	s	–	s	s	s	s	s
Log4J	s	s	s	–	n	s	s	=
JCaptcha	s	s	s	n	–	s	s	n
Pdfbox	n	n	s	s	s	–	n	s
JReports	n	n	s	s	s	n	–	s
Hibernate	s	s	s	=	n	s	s	–

B. Coupling Measures

Boxplots of CBO measures are given in Figures 12 and 13. The results of the Wilcoxon rank sum tests and Hedges's g evaluations concerning CBO measures are given in Table XIV.

TABLE XIV
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR CBO.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	s	n	n	s	s	s
Ind2	n	–	s	n	n	n	s	n
Ind3	s	s	–	s	s	s	n	s
Log4J	n	n	s	–	n	s	s	n
JCaptcha	n	n	s	n	–	s	s	s
Pdfbox	s	n	s	s	s	–	n	n
JReports	s	s	n	s	s	n	–	s
Hibernate	s	n	s	n	s	n	s	–

C. Response for a Class

Boxplots of RFC measures are given in Figures 14 and 15. The results of the Wilcoxon rank sum tests and Hedges's g evaluations concerning RFC measures are given in Table XV.

D. Results for WMC

Boxplots of WMC measures are given in Figures 16 and 17. The results of the Wilcoxon rank sum tests and Hedges's g evaluations concerning WMC measures are given in Table XVI.

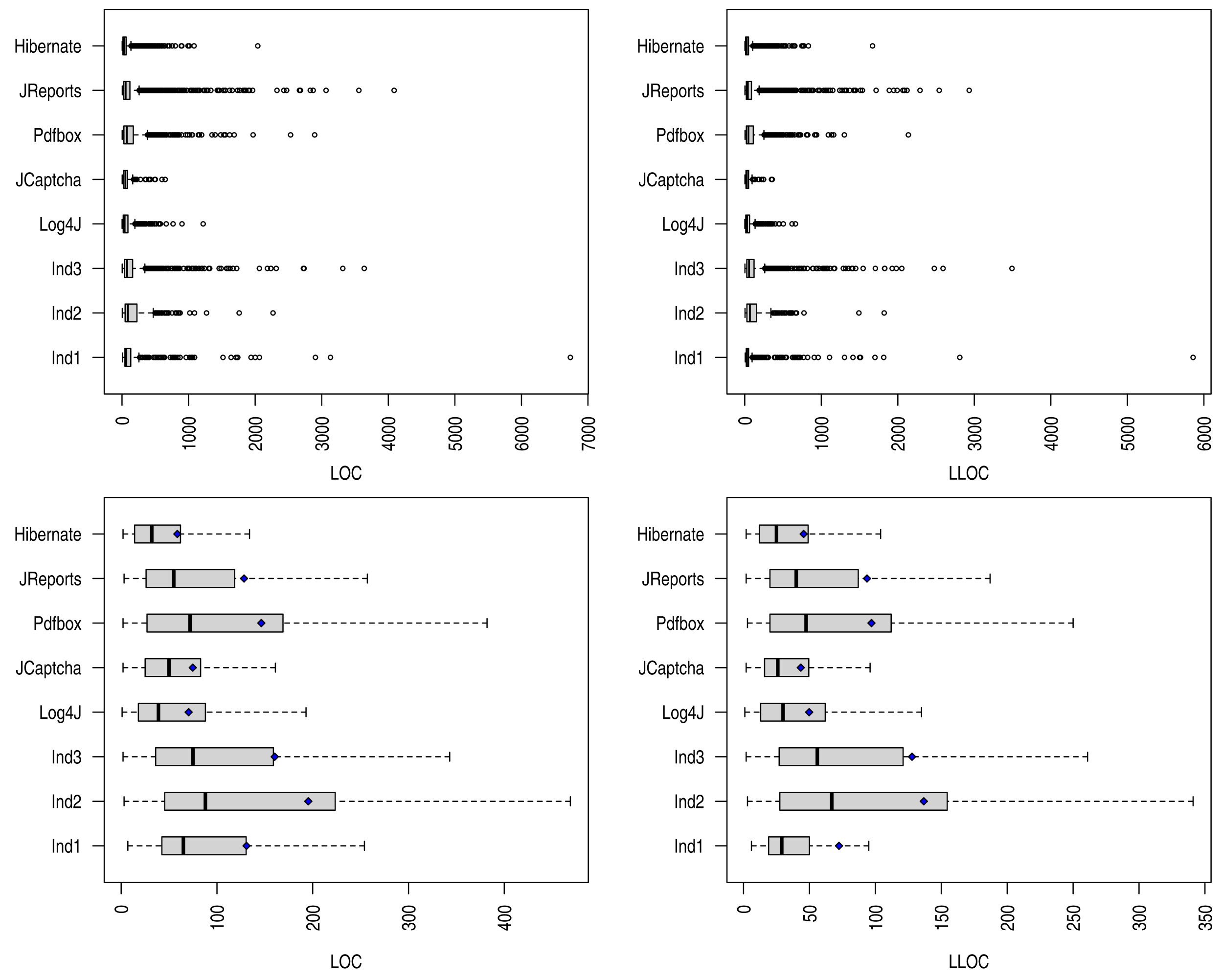

Figure 11. Boxplots of LoC (left) and LLOC (right) distributions, with (top) and without (bottom) outliers.

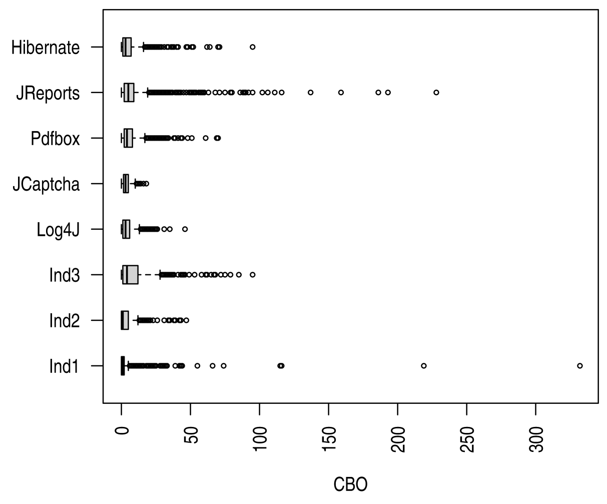

Figure 12. Boxplots of CBO distributions.

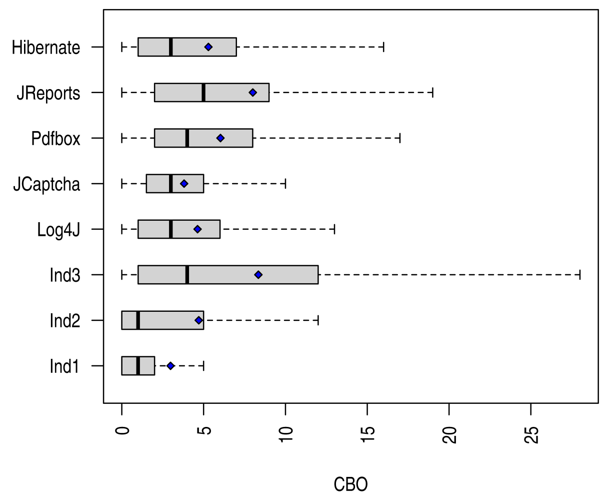

Figure 13. Boxplots of CBO distributions. Outliers omitted.

E. Results for Class Cohesion

Class cohesion was measured via the LCOM5 metric. Boxplots of LCOM5 measures are given in Figures 18 and 19.

The results of the Wilcoxon rank sum tests and Hedges's g evaluations concerning LCOM5 measures are given in

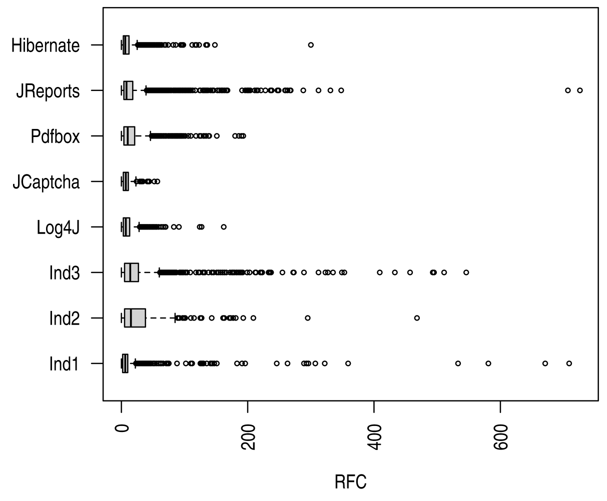

Figure 14. Boxplots of RFC distributions.

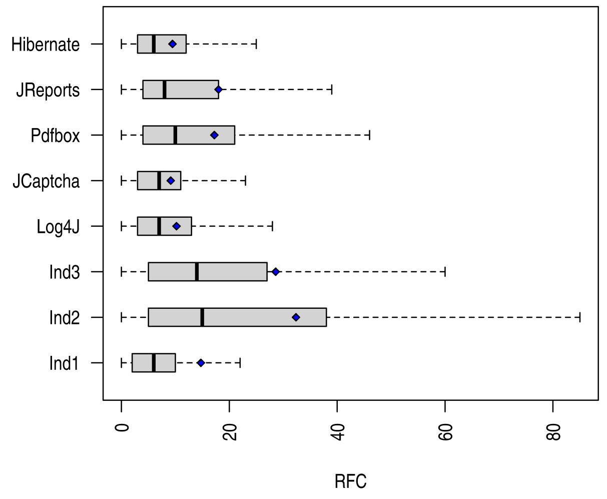

Figure 15. Boxplots of RFC distributions. Outliers omitted.

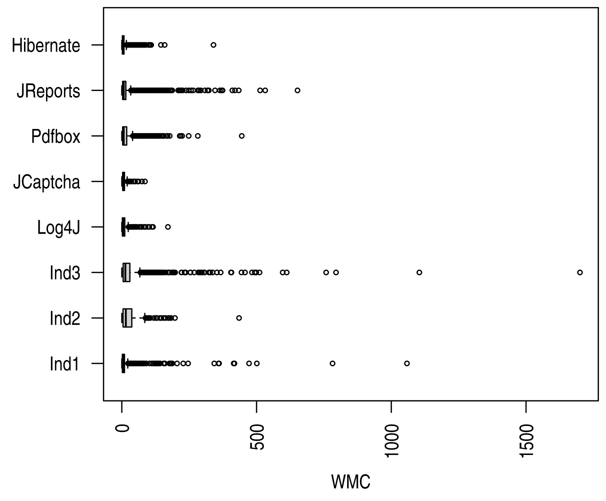

Figure 16. Boxplots of WMC distributions.

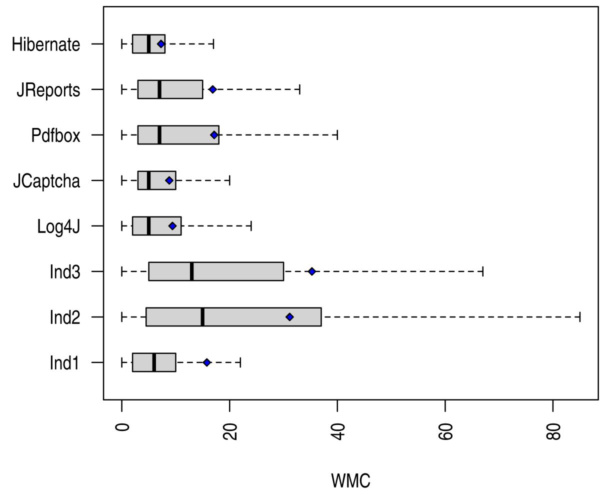

Figure 17. Boxplots of WMC distributions. Outliers omitted.

Table XVII.

F. Results for the Depth of Inheritance

Boxplots of DIT measures are given in Figures 20 and 21. The results of the Wilcoxon rank sum tests and Hedges's g evaluations concerning DIT measures are given in Table XVIII.

G. Results for the Number of Children

Boxplots of NOC measures are given in Figures 22 and 23. The results of the Wilcoxon rank sum tests and Hedges's g evaluations concerning NOC measures are given in Table XIX.

VI. Discussion

Figure 1 shows that the set of chosen projects are quite homogeneous with respect to size, all projects having the great majority of methods no longer than 200 LOC. This homogeneity is confirmed by the effect size evaluations given

TABLE XV
Wilcoxon rank sum test and Hedges's g results for RFC.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	s	s	n	n	n	n	n
Ind2	s	–	n	l	m	m	s	l
Ind3	s	n	–	s	s	s	s	m
Log4J	n	l	s	–	=	s	s	n
JCaptcha	n	m	s	=	–	s	s	n
Pdfbox	n	m	s	s	s	–	n	s
JReports	n	s	s	s	s	n	–	s
Hibernate	n	l	m	n	n	s	s	–

TABLE XVI
Wilcoxon rank sum test and Hedges's g results for WMC.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	s	s	n	n	n	n	s
Ind2	s	–	=	l	m	s	s	l
Ind3	s	=	–	s	s	s	s	m
Log4J	n	l	s	–	=	s	s	n
JCaptcha	n	m	s	=	–	s	s	n
Pdfbox	n	s	s	s	s	–	n	s
JReports	n	s	s	s	s	n	–	s
Hibernate	s	l	m	n	n	s	s	–

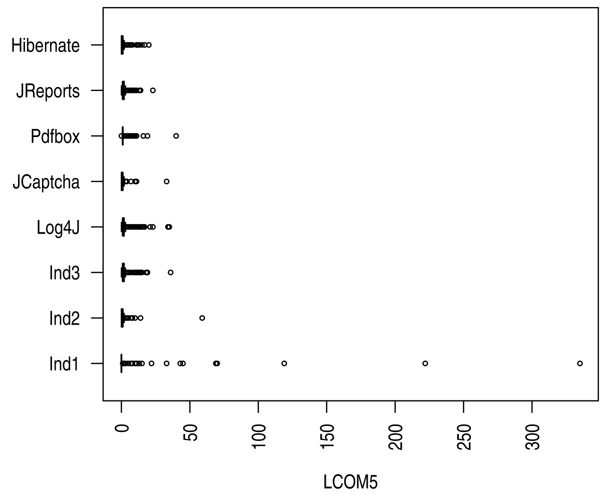

Figure 18. Boxplots of LCOM5 distributions.

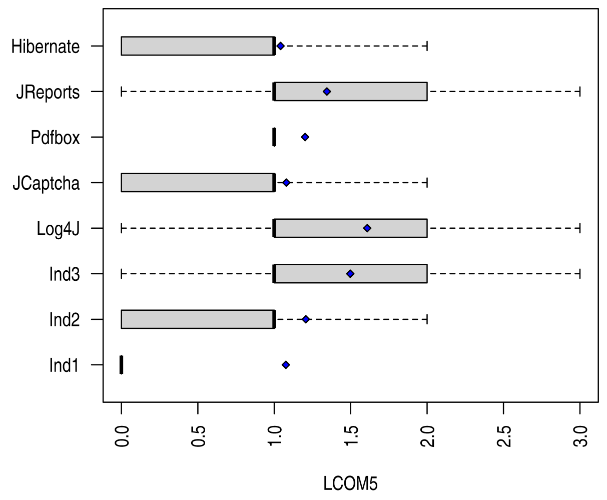

Figure 19. Boxplots of LCOM5 distributions. Outliers omitted.

in Tables VI and XII: only negligible and small effect sizes were found, both at the method and class level.

Similarly, the great majority of methods have McCabe complexity not greater than 5 for all projects, with the only exception of Industrial3 (see Figure 4). However, also in project Industrial3, only outliers have alarmingly high McCabe complexity. As for LOC, the effect size is at most small, indicating substantial equivalence of the projects' complexity measures.

TABLE XVII
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR LCOM5.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	n	n	n	n	n	n
Ind2	n	–	n	n	n	n	n	n
Ind3	n	n	–	n	n	n	=	s
Log4J	n	n	n	–	s	s	n	s
JCaptcha	n	n	n	s	–	n	n	n
Pdfbox	n	n	n	s	n	–	n	n
JReports	n	n	=	n	n	n	–	s
Hibernate	n	n	s	s	n	n	s	–

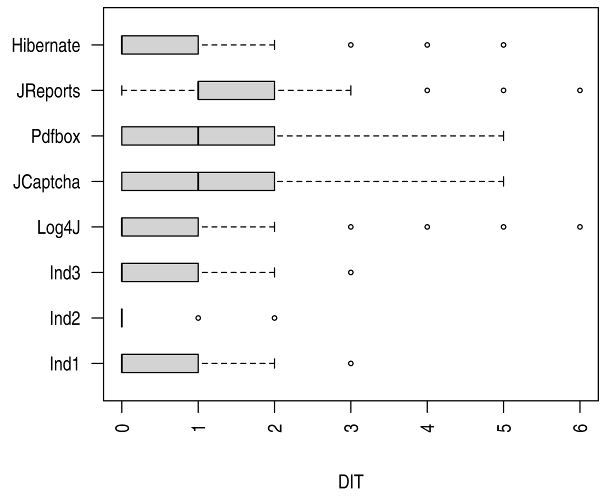

Figure 20. Boxplots of DIT distributions.

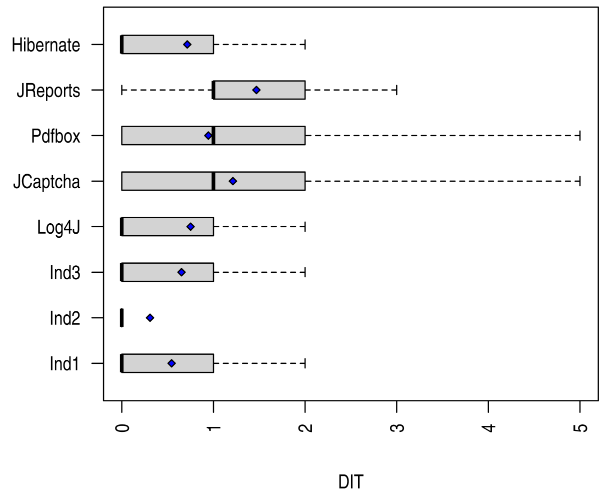

Figure 21. Boxplots of DIT distributions. Outliers omitted.

Concerning the Maintainability Index, Figure 5 shows that Industrial1 and Industrial3 are the only projects that include methods with negative MI; specifically, Industrial3 has several methods with negative MI, some with alarmingly low values. So, even though the situation excluding outliers (Figure 6) seems to indicate a rather homogeneous situation, industrial projects appear to be less maintainable then open-source projects in several cases: according to Table IX, in 8 out of 15 comparisons involving a closed-source and an open-source

TABLE XVIII
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR DIT.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	s	n	n	m	s	m	n
Ind2	s	–	s	s	l	m	l	s
Ind3	n	s	–	n	m	s	m	n
Log4J	n	s	n	–	s	n	m	n
JCaptcha	m	l	m	s	–	s	s	m
Pdfbox	s	m	s	n	s	–	s	s
JReports	m	l	m	m	s	s	–	m
Hibernate	n	s	n	n	m	s	m	–

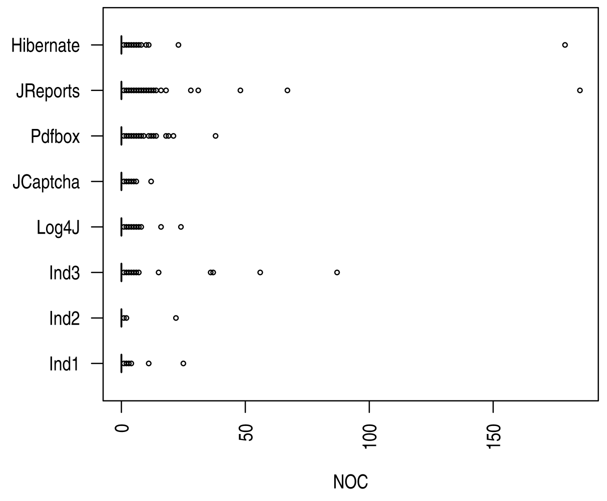

Figure 22. Boxplots of NOC distributions.

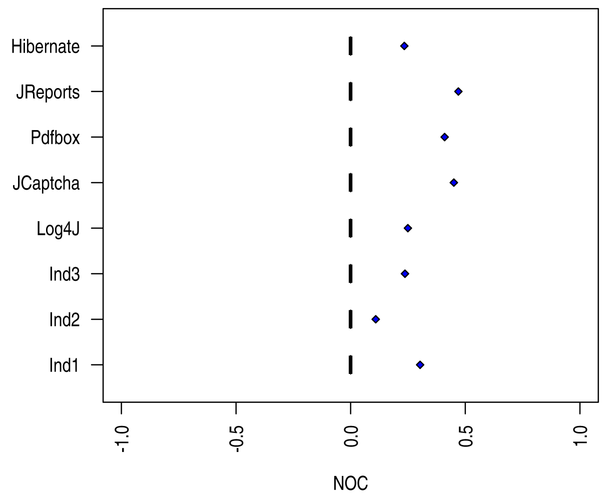

Figure 23. Boxplots of NOC distributions. Outliers omitted.

project, the effect size was medium. Instead, comparisons involving only open-source projects and comparisons involving only closed-source projects revealed at most small effect size.

Finally, we can see that all projects are fairly homogeneous with respect to Halstead volume (Figures 7 and 8 and Table X). Similar considerations apply for Halstead Computed Program Length (HCPL), with medium effect size differentiating industrial projects only with respect to Hibernate (Table XI).

In conclusion, we can observe that the analyzed open-source and closed-source code appear sufficiently similar, as far as method-level metrics are concerned.

Halstead volume and program length appear homogeneous through open-source and closed source programs (Tables X and X), with the exception of Hibernate, which appears "smaller" than the closed-source industrial programs.

Concerning coupling (CBO) and cohesion (LCOM5), no difference could be spot between open-source and closed-source programs.

RFC and WMC appear larger in Ind2 and Ind3, with medium effect size in just a few cases (Tables XV and XVI).

Finally, the situation appear quite homogeneous also when inheritance-related metrics (DIT and NOC) are concerned. There are several medium effect size values in Table XVIII concerning DIT, but we have to consider that all the DIT values are small: as shown in Figure 21, the great majority of classes has DIT not grater than 2. Only Ind2 seems to make very little usage of inheritance (its values of both DIT and NOC are definitely small).

Overall, neither for classes nor for methods there are remarkable differences between open-source and closed-source programs.

TABLE XIX
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR NOC.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	n	n	n	n	=	n
Ind2	n	–	n	n	s	n	n	n
Ind3	n	n	–	n	n	n	n	n
Log4J	n	n	n	–	n	n	n	n
JCaptcha	n	s	n	n	–	n	n	n
Pdfbox	n	n	n	n	n	–	n	n
JReports	=	n	n	n	n	n	–	n
Hibernate	n	n	n	n	n	n	n	–

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.

A risk with the type of work presented here is that the code that companies are willing to provide to researchers might differ from the code they would not provide. This is possibly due to the desire to "hide" low-quality code. In our case, it is not so: the closed-source code being measured is the complete code being used to build production applications and is representative of the companies' software in general.

Concerning the external validity of the study, as with most Software Engineering empirical studies, we cannot claim that the obtained results are generalizable. Specifically, the limited number of considered projects calls for replications of this study, involving more industrial closed-source code projects.

VIII. RELATED WORK

Open-source projects have been compared with closed-source ones multiple times, but usually with respect to external perceivable qualities. In fact, many of the published papers aimed at answering questions like "Should I use this open-source software product or this closed-source one?" These papers considered issues like reliability, speed and effectiveness of defect removal, evolution, security, etc.

Bachmann and Bernstein [14] surveyed five open source projects and one closed source project to evaluate the quality and characteristics of data from bug tracking databases and version control system log files. Among other things, they discovered a poor quality in the link rate between bugs and commits.

The debate on the security of open-source software compared to that of closed-source software have produced several studies. This is due not only to the relevance of the problem, but also to the fact that security issues concerning closed-source software are publicly available, even when the source code is not.

Schryen and Kadura [15] analyzed and compared published vulnerabilities of eight open-source software and nine closed-source software packages. They provided an empirical analysis of vulnerabilities in terms of mean time between vulnerability disclosures, the development of disclosure over time, and the severity of vulnerabilities.

Schryen [16] also investigated empirically the patching behavior of software vendors/communities of widely deployed open-source and closed-source software packages. He found that it is not the particular software development style that determines patching behavior, but rather the policy of the particular software vendor.

Paulson et al. [17] compared open- and closed-source projects to investigate the hypotheses that open-source software grows more quickly, that creativity is more prevalent in open-source software, that open-source projects succeed because of their simplicity, that defects are found and fixed more rapidly in open-source projects.

As opposed to the papers mentioned above, here a fairly systematic comparison of code measures is proposed. Previously, MacCormack et al. compared the structure of an open-source system (Linux) an a closed-source system (Mozilla) [18]. With respect to our work, they evaluated just one code property (modularity) for a single pair of products.

A comparison based on code metrics involving multiple open-source and closed-source projects [19] was performed from a different point of view and using different techniques: the authors modified the Least Absolute Deviations technique where, instead of comparing metrics data to an ideal distribution, metrics from two programs are compared directly to each other via a data binning technique.

IX. Conclusions

Open-source projects provide the code used in many empirical studies. The applicability of the results of these studies to software projects in general, i.e., including closed-source projects, would be questionable, if open-source code were not representative of closed-source code as well.

To address this issue, a comparison of open-source and closed-source code was performed. Specifically, static code measures from five open-source projects were compared to those obtained from three closed-source projects. The study—which addressed only Java code—shows that some of the most well-known static code measures appear similar in open-source and in industrial closed-source products.

However, we recall that the study reported here involved just a few industrial projects' measures, because getting access to industrial code is not easy. Hence, the presented analysis should be regarded as a preliminary results, which needs replications before it can be considered valid in general.

References

[1] L. Lavazza, "A comparison of closed-source and open-source code static measures," in ICSEA 2024-The Nineteenth International Conference on Software Engineering Advances. IARIA, 2024, pp. 1–6.

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

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

[4] 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.

[5] "SourceMeter," https://www.sourcemeter.com/, [retrieved August, 2024].

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

[7] K. D. Welker, P. W. Oman, and G. G. Atkinson, "Development and application of an automated source code maintainability index," Journal of Software Maintenance: Research and Practice, vol. 9, no. 3, 1997, pp. 127–159.

[8] Diomidis Spinellis, "ckjm – A Tool for Calculating Chidamber and Kemerer Java Metrics," https://www.spinellis.gr/sw/ckjm/doc/indexw.html.

[9] "Log4j."

[10] "Jcaptcha," https://jcaptcha.sourceforge.net/.

[11] "Apache pdfbox," https://pdfbox.apache.org/.

[12] "Jasperreports."

[13] "Hibernate," https://hibernate.org/.

[14] A. Bachmann and A. Bernstein, "Software process data quality and characteristics: a historical view on open and closed source projects," in Proceedings of the Joint int. ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops, 2009, pp. 119–128.

[15] G. Schryen, "Security of open source and closed source software: An empirical comparison of published vulnerabilities," AMCIS 2009 Proceedings, 2009, p. 387.

[16] ——, "A comprehensive and comparative analysis of the patching behavior of open source and closed source software vendors," in 2009 Fifth International Conference on IT Security Incident Management and IT Forensics. IEEE, 2009, pp. 153–168.

[17] J. W. Paulson, G. Succi, and A. Eberlein, "An empirical study of open-source and closed-source software products," IEEE transactions on software engineering, vol. 30, no. 4, 2004, pp. 246–256.

[18] A. MacCormack, J. Rusnak, and C. Y. Baldwin, "Exploring the structure of complex software designs: An empirical study of open source and proprietary code," Management Science, vol. 52, no. 7, 2006, pp. 1015–1030.

[19] B. Robinson and P. Francis, "Improving industrial adoption of software engineering research: a comparison of open and closed source software," in Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2010, pp. 1–10.