How does Context Affect the Distribution of Software Maintainability Metrics?

Feng Zhang*, Audris Mockus†, Ying Zou‡, Foutse Khomh§, and Ahmed E. Hassan*

* School of Computing, Queen's University, Canada
† Department of Software, Avaya Labs Research, Basking Ridge, NJ 07920. USA
‡ Department of Electrical and Computer Engineering, Queen's University, Canada
§ SWAT, École Polytechnique de Montréal, Canada
feng@cs.queensu.ca, audris@avaya.com, ying.zou@queensu.ca, foutse.khomh@polymtl.ca, ahmed@cs.queensu.ca

***Abstract*—Software metrics have many uses, *e.g.*, defect prediction, effort estimation, and benchmarking an organization against peers and industry standards. In all these cases, metrics may depend on the context, such as the programming language. Here we aim to investigate if the distributions of commonly used metrics do, in fact, vary with six context factors: application domain, programming language, age, lifespan, the number of changes, and the number of downloads. For this preliminary study we select 320 nontrivial software systems from SourceForge. These software systems are randomly sampled from nine popular application domains of SourceForge. We calculate 39 metrics commonly used to assess software maintainability for each software system and use Kruskal Wallis test and Mann-Whitney U test to determine if there are significant differences among the distributions with respect to each of the six context factors. We use Cliff's delta to measure the magnitude of the differences and find that all six context factors affect the distribution of 20 metrics and the programming language factor affects 35 metrics. We also briefly discuss how each context factor may affect the distribution of metric values. We expect our results to help software benchmarking and other software engineering methods that rely on these commonly used metrics to be tailored to a particular context.**

***Index Terms*—context; context factor; metrics; static metrics; software maintainability; benchmark; sampling; mining software repositories; large scale.**

I. Introduction

The history of software metrics predates software engineering. The first active software metric is the number of lines of code (LOC) which was used in the mid 60's to assess the productivity of programmers [1]. Since then, a large number of metrics have been proposed [2], [3], [4], [5], and extensively used in software engineering activities, *e.g.*, defect prediction, effort estimation and software benchmarks. For example, software organizations use benchmarks as management tools to assess the quality of their software products in comparison to industry standards (*i.e.*, benchmarks). Since software systems are developed in different environments, for various purposes, and by teams with diverse organizational cultures, we believe that context factors, such as application domain, programming language, and the number of downloads, should be taken into account, when using metrics in software engineering activities (*e.g.*, [6]). It can be problematic [7] to apply metric-based benchmarks derived from one context to software systems in a different context, *e.g.*, applying benchmarks derived from small-size software systems to assess the maintainability of large-size software systems. COCOMO II[1] model supports a number of attribute settings (*e.g.*, the complexity of product) to fine tune the estimation of the cost and system size (*i.e.*, source lines of code). However, to the best of our knowledge, no study provides empirical evidence on how contexts affect such metric-based models. The context is overlooked in most existing approaches for building metric-based benchmarks [8], [9], [10], [11], [12], [13], [14].

This preliminary study aims to understand if the distributions of metrics do, in fact, vary with contexts. Considering the availability and understandability of context factors and their potential impact on the distribution of metric values, we decide to investigate seven context factors: application domain, programming language, age, lifespan, system size, the number of changes, and the number of downloads. Since system size strongly correlates to the number of changes, we only examine the number of changes of the two factors.

In this study, we select 320 nontrivial software systems from SourceForge[2]. These software systems are randomly sampled from nine popular application domains. For each software system, we calculate 39 metrics commonly used to assess software maintainability. To better understand the impact on different aspects of software maintainability, we further classify the 39 metrics into six categories (*i.e.*, complexity, coupling, cohesion, abstraction, encapsulation and documentation) based on Zou and Kontogiannis [15]'s work. We investigate the following two research questions:

RQ1: *What context factors impact the distribution of the values of software maintainability metrics?*
We find that all six context factors affect the distribution of the values of 51% of metrics (*i.e.*, 20 out of 39). The most influential context factor is the programming language, since it impacts the distribution of the values of 90% of metrics (*i.e.*, 35 out of 39).

[1] http://sunset.usc.edu/csse/research/COCOMOII
[2] http://www.sourceforge.net

RQ2: *What guidelines can we provide for benchmarking*[3] *software maintainability metrics?*
We suggest to group all software systems into 13 distinct groups using three context factors: application domain, programming language, and the number of changes, and consequently, we obtain 13 benchmarks.

The remainder of this paper is organized as follows. In Section II, we summarize the related work. We describe the experimental setup of our study in Section III and report our case study results in Section IV. Threats to validity of our work are discussed in Section V. We conclude and provide insights for future work in Section VI.

II. Related Work

In this section, we review previous attempts to build metric-based benchmarks. Such attempts are usually made up of a set of metrics with proposed thresholds and ranges.

A. Thresholds and ranges of metric values

Deriving appropriate thresholds and ranges of metrics is important to interpret software metrics [17]. For instance, McCabe [2] proposes a widely used complexity metric to measure software maintainability and testability, and further interprets the value of his metric in such a way: sub-functions with the metric value between 3 and 7 are well structured; sub-functions with the metric value beyond 10 are unmaintainable and untestable [2]. Lorenz and Kidd [3] propose thresholds and ranges for many object-oriented metrics, which are interpretable in their context. However, direct application of these thresholds and ranges without taking into account the contexts of systems might be problematic. Erni and Lewerentz [4] propose to consider mean and standard deviations based on the assumption that the metrics follow normal distributions. Yet many metrics follow power-law or log-normal distributions [18], [19]. Thus many researchers propose to derive thresholds and ranges based on statistical properties of metrics. For example, Benlarbi *et al.* [8] apply a linear regression analysis. Shatnawi [9] use a logistic regression analysis. Yoon *et al.* [10] use a k-means cluster algorithm. Herbold *et al.* [12] use machine learning techniques. Sánchez-González *et al.* [20] compare two techniques: ROC curves and the Bender method.

A recent attempt to build benchmarks is by Alves *et al.*[11]. They propose a framework to derive metric thresholds by considering metric distributions and source code scales, and select a set of software systems from a variety of contexts as measurement data. Baggen *et al.* [21] present several applications of Alves *et al.*[11]'s framework. Bakota *et al.* [13] propose a different approach using probabilities other than thresholds or ranges, and focus on aggregating low-level metrics to maintainability as described in ISO/IEC 9126 standard.

The aforementioned studies do not consider the potential impact by the contexts of software systems. The contexts can affect the effective values of various metrics [7]. As compliments to these studies, our work investigates how contexts impact the distribution of software metric values. We further provide guidelines to split software systems using contexts before applying these approaches (*e.g.*, [9], [10], [11], [12]) to derive thresholds and ranges of metric values.

B. Context factors

Contexts of software systems are considered in Ferreira *et al.* [14]'s work. They propose to identify thresholds of six object-oriented software metrics using three context factors: application domain, software types and system size (in terms of the number of classes). However, they directly split all software systems using the context factors without examining whether the context factors affect the distribution of metric values or not, thus result in a high ratio of duplicated thresholds. To reduce duplications and maximize the samples of measurement software systems, a split is necessary only when a context factor impacts the distribution of metric values. Different from Ferreira *et al.* [14]'s study, we propose to split all software systems based on statistical significance and corresponding effect size. We study four additional context factors and 33 more metrics than their study.

Open source software systems are characterized by Capiluppi *et al.* [22] using 12 context factors: age, application domain, programming language, size (in terms of physical occupation), the number of developers, the number of users, modularity level, documentation level, popularity, status, success of project, and vitality. Considering not all software systems provide information for these factors, we decide to investigate five commonly available context factors: application domain, programing language, age, system size (we redefined it as the total number of lines), and the number of downloads (measured using average monthly downloads). Since software metrics are also affected by software evolution [23], we study two additional context factors: lifespan and the number of changes.

To the best of our knowledge, we are the first to propose detailed investigation of context factors when building metric-based benchmarks.

III. Case Study Setup

This section presents the design of our case study.

A. Factor Description

In this subsection, we briefly describe the definition and motivation of each factor.

1) Application Domain (AD): describes the type of software systems. In general, software systems designed as *frameworks* might contain more classes than other types of software systems.

2) Programming Language (PL): describes the nature of programming paradigms. Generally speaking, software systems written in Java might have deeper inheritance tree than C++, since C++ supports both object oriented programming and structural programming.

[3]A benchmark is generally defined as "a test or set of tests used to compare the performance of alternative tools or techniques" [16]. In this study, we refer to "benchmark" as a set of metric-based evaluations of software maintainability.

3) Age (AG): is the time duration after creating a software system. As software development techniques evolve fast, older software systems might be more difficult to maintain than newly created software systems.

4) Lifespan (LS): describes the time duration of development activities in the life of a software system. Software systems developed over a long period of time might be harder to maintain than software systems developed over a shorter time period.

5) System Size (SS): is the total lines of code of a software system. Small software systems might be easier to maintain than large ones.

6) Number of Changes (NC): describes the total number of commits made to a software system. It might be more difficult to maintain heavily-modified software systems than lightly-modified ones.

7) Number of Downloads (ND): describes the external quality of a software system. It is of interest to find if popular software systems have better maintainability than less popular ones. In this study, the number of downloads is measured using the average monthly downloads which were collected directly from SourceForge.

B. Data Collection

Data Source. We use the SourceForge data initially collected by Mockus [24]. There are some updates after that work, and the new data collection was finished on February 05, 2010. The dataset contains 154,762 software systems. However, we find 97,931 incomplete software systems which contain fewer than 41 files, and an empty CVS repository has 40 files. There are 56,833 nontrivial software systems in total from SourceForge. FLOSSMole [25] is another data source, from where we download descriptions (*i.e.*, application domain) of SourceForge software systems. Furthermore, we download latest application domain information[4] and monthly download data[5] of studied software systems directly from SourceForge.
Sampling. Investigating all the 56,833 software systems requires a large amount of computation resources. For example, the snapshots of our selected 320 software systems occupy about 8 GB hard drive, and the computed metrics take more than 15 GB hard drive. The average time for computing metrics of one software system is 6 minutes. The bottleneck is the slow disk I/O, since we intensively access disks (*e.g.*, to dump snapshots of source code, and to store metric values). Applying SSD or RAID storage or using RAM drive might eliminate this bottleneck. Yet our resource is limited to apply such solution at this moment. For a preliminary study, we perform stratified sampling of software systems by application domains to explore how context factors affect the distribution of metric values. The limitation of stratified sampling is discussed in Section V. Moreover, we plan to stratify by the remaining six factors in future. In this exploratory study, we pick nine popular application domains containing over 1,000 software systems. We conduct simple random sampling to select 100 software systems from each application domain and obtain 900 software systems in total. Yet there are only 824 different software systems, since a software system may be categorized into several application domains.

C. Factor Extraction

1) Application Domain (AD): We extract the application domain of each software system using the data collected in June 2008 by FLOSSMole [25]. We rank all application domains using the number of software systems and pick nine popular application domains: Build Tools, Code Generators, Communications, Frameworks, Games/Entertainment, Internet, Networking, Software Development (excluding Build Tools, Code Generators, and Frameworks), and System Administration. We replace sub-domains, if exist, by their parent application domain.

2) Programming Language (PL): In this study, we only investigate software systems that are mainly written in C, C++, C#, Java, or Pascal. For each software system, we dump the latest snapshot, and determine the main programing language by counting the total number of files per file type (*i.e.*, *.c, *.cpp, *.cxx, *.cc, *.cs, *.java, and *.pas).

3) Age (AG): For each software system, we compute the age using the date of the first CVS commit. In the sampled 824 software systems, the oldest software system[6] was created on November 02, 1996, and the latest software system[7] was created on May 28, 2008.

4) Lifespan (LS): For each software system, we compute the lifespan by computing the intervals between the first and the last CVS commits. The quantiles of lifespan in a unit of day in the sampled 824 software systems are: 0 (minimum), 51 (25%), 338 (median), 930 (75%), and 4,038 (maximum).

5) System Size (SS): For each software system, we count the total lines of code from the latest snapshot. The quantiles of the total lines of code in the sampled 824 software systems are: 0 (minimum), 1,124 (25%), 3,955 (median), 14,945 (75%), and 2,792,334 (maximum).

6) Number of Changes (NC): For each software system, we count the total number of commits from the whole history. The quantiles of the number of changes in the sampled 824 software systems are: 12 (minimum), 123 (25%), 413 (median), 1,142 (75%), and 94,853 (maximum).

7) Number of Downloads (ND): For each software system, we first sum up all the monthly downloads to get the total number of downloads, and search the first and the last month with at least one download to determine the downloading period. We divide the total downloads by the total number of months of the downloading period to obtain the average monthly downloads. The quantiles of the average monthly downloads in the sampled 824 software systems are: 0 (minimum), 0 (25%), 6 (median), 16 (75%), and 661,247 (maximum).

[4]http://sourceforge.net/projects/ProjectName (NOTE: the ProjectName needs to be substituted by the real project name, *e.g.*, a2ixlibrary)

[5]http://sourceforge.net/projects/ProjectName/files/stats/json?start_date=1990-01-01&end_date=2012-12-31 (NOTE: the ProjectName needs to be substituted by the real project name, *e.g.*, gusi)

[6]gusi, http://sourceforge.net/projects/gusi

[7]pic-gcc-library, http://sourceforge.net/projects/pic-gcc-library

D. Data Cleanliness

1) Lifespan (LS): The 25% quantile of the lifespan of the 824 software systems is 51 days. We suspect that software systems with lifespans less than the 25% quantile are never finished or are just used as prototypes. After manually checking such software systems, we find that most of them are incomplete with very few commits. We exclude such software systems from our study. The number of subject systems drops from 824 to 618.

2) System Size (SS): We manually check the software systems with lines of code less than the 25% quantile (*i.e.*, 1,124) of the 824 software systems. We find that most of such software systems are incomplete (*e.g.*, vcgenv[8]), or mainly written in other languages (*e.g.*, jajax[9] is written mainly in javascript). We exclude such software systems from our study. The number of subject systems drops from 618 to 506.

3) Programming Language (PL): We filter out software systems that are not mainly written in C, C++, C#, Java, or Pascal. The number of subject systems drops from 506 to 478.

4) Number of Downloads (ND): Some of the remaining 478 software systems have no downloads. It might because that such software systems are still incomplete to be used, or they are absolutely useless software. We exclude software systems without downloads from our study. The number of subject systems drops from 478 to 390.

5) Application Domain (AD): A software system might be categorized into several application domains. The combinations of multiple application domains are considered as different application domains from single application domains. The 75% quantile of the number of software systems of all single and combined application domains is seven. We exclude combined application domains which have less than seven software systems, and yield six combined application domains. The number of subject systems drops from 390 to 323.

The collection date of the application domain is not the same as the date of collecting source code. To verify whether the application domains of the 323 software systems remain the same as time elapses, we checked the latest application domain information directly downloaded from SourceForge in April 2013. We find that only three software systems (*i.e.*, cdstatus, g3d-cpp, and satyr[10]) have changed their application domains, indicating that application domains collected in June 2008 are adequate. The application domain of *g3d-cpp* is removed, and the application domains of *cdstatus* and *satyr* are changed to Audio/Video. We exclude the three software systems from our study. The number of subject systems drops from 323 to 320.

Refine Factors. The context factors like age, lifespan, system size, and the number of changes seem to be strongly related. Hence, we compute Spearman correlation among these context factors of the 320 software systems. As shown in Table I, system size strongly correlates to the number of changes. We choose to examine the number of changes only.

[8]http://sourceforge.net/projects/vcgenv

[9]http://sourceforge.net/projects/jajax

[10]http://sourceforge.net/projects/ProjectName (NOTE: the ProjectName needs to be substituted by the real project name, *e.g.*, cdstatus)

TABLE I: The Spearman correlations among four context factors: age, lifespan, system size, and the number of changes.

Context Factor	Lifespan	System Size	Number of Changes
Age	0.35	0.06	0.14
Lifespan		0.25	0.46
System Size			0.67

TABLE II: The number of software systems per group divided by each context factor.

Context Factor	Group	Number of Systems
Application Domain (AD)	G_{build}	31
	$G_{codegen}$	26
	G_{comm}	23
	G_{frame}	29
	G_{games}	49
	$G_{internet}$	19
	$G_{network}$	16
	G_{swdev}	41
	$G_{sysadmin}$	29
	$G_{build;codegen}$	14
	$G_{comm;internet}$	13
	$G_{comm;network}$	7
	$G_{games;internet}$	7
	$G_{internet;swdev}$	9
	$G_{swdev;sysadmin}$	7
Programming Language (PL)	G_{c}	57
	G_{cpp}	85
	$G_{c\#}$	18
	G_{java}	146
	G_{pascal}	14
Age (AG)	G_{lowAG}	80
	$G_{moderateAG}$	160
	G_{highAG}	80
Life Span (LS)	G_{lowLS}	80
	$G_{moderateLS}$	160
	G_{highLS}	80
Number of Changes (NC)	G_{lowNC}	80
	$G_{moderateNC}$	160
	G_{highNC}	80
Number of Downloads (ND)	G_{lowND}	90
	$G_{moderateND}$	150
	G_{highND}	80

Summary. When investigating the impact of application domain (respectively programming language) on the distribution of metric values, we break down the 320 software systems into 15 (respectively five) groups as shown in Table II. When investigating the impact of the other four context factors on the distribution of metric values, we divide the 320 software systems into three groups, respectively, in the following way: 1) low (below or at the 25% quantile); 2) moderate (above the 25% quantile and below or at the 75% quantile); and 3) high (above the 75% quantile). The 25% quantile of lifespan (respectively the number of changes and the number of monthly downloads) is: 287 days (respectively 364 and 6). The 75% quantile of lifespan (respectively the number of changes and the number of monthly downloads) is: 1,324 days (respectively 2,195 and 38). The detailed groups are shown in Table II.

E. Metrics Computation

In this study, we select 39 metrics related to the five quality attributes (*i.e.*, modularity, reusability, analyzability, modifiability, and testability) of software maintainability (as defined in ISO/IEC 25010). We further group the 39 metrics into six categories (*i.e.*, complexity, coupling, cohesion, abstraction, encapsulation, and documentation) based on Zou and Kontogiannis[15]'s work. These categories can measure different aspects of software maintainability. For example, low complexity indicates high analyzability and modifiability; low coupling improves analyzability and reusability; high cohesion increases modularity and modifiability; high abstraction enhances reusability; high encapsulation implies high modularity; and documentation might contribute to analyzability, modifiability, and reusability. The metrics and their categories are shown in Table III. Most metrics can be computed by a commercial tool, called Understand[11]. For the remaining metrics, we computed them by ourselves[12] using equations from the work of Aggarwal *et al.* [26].

F. Analysis Methods

For each factor, we divide all software systems into non-overlapping groups, as described in Section III-D.

Analysis method for RQ1: To study how context factors impact the distribution of metric values, we analyze the overall impact of each context factor. To examine the overall impact of a factor f on a metric m, we test the following null hypothesis for the grouping based on factor f.

$H0_1$: *there is no difference in the distributions of metric values among all groups*.

To compare the distribution of metric m values among all groups, we apply Kruskal Wallis test [32] using the 5% confident level (*i.e.*, p-value$<$0.05). The Kruskal Wallis test assesses whether two or more samples originate from the same distribution. It does not assume a normal distribution since it is a non-parametric statistical test. As we investigate six context factors and 39 metrics in total, we apply Bonferroni correction which adjusts the threshold p-value by dividing the number of tests (39×6=234 tests). If there is a statistically significant difference (*i.e.*, p-value is less than $0.05/234$=$2.14e$-04), we reject the null hypothesis and report that factor f impacts the distribution of metric m values.

Analysis method for RQ2: To provide guidelines on how to group software systems for benchmarking maintainability metrics, we break down our analysis method into three steps: 1) for each impacting factor, we examine the impact in detail by comparing every pair of groups separated by the factor; 2) for any comparison exhibiting a statistically significant difference, we further compute the corresponding effect size to quantify the importance of the difference; and 3) we discuss how to use effect sizes to split software systems. We present the detailed steps as follows.

[11]http://www.scitools.com

[12]https://bitbucket.org/serap/contextstudy

TABLE III: List of metrics characterizing maintainability. The metrics are from three levels: project level (P), class level (C), and method level (M).

Category	Metric	Level
Complexity	Total Lines of Code (TLOC)	P
	Total Number of Files (TNF)	P
	Total Number of Classes (TNC)	P
	Total Number of Methods (TNM)	P
	Total Number of Statements (TNS)	P
	Class Lines of Code (CLOC)	C
	Number of local Methods (NOM) [27]	C
	Number of Instance Methods (NIM) [28]	C
	Number of Instance Variables (NIV) [28]	C
	Weighted Methods per Class (WMC) [29]	C
	Number of Method Parameters (NMP)	M
	McCabe Cyclomatic Complexity (CC) [2]	M
	Number of Possible Paths (NPATH) [28]	M
	Max Nesting Level (MNL) [28]	M
Coupling	Coupling Factor (CF) [30]	P
	Coupling Between Objects (CBO) [29]	C
	Information Flow Based Coupling (ICP) [5]	C
	Message Passing Coupling (MPC) [27]	C
	Response For a Class (RFC) [29]	C
	Number of Method Invocation (NMI)	M
	Number of Input Data (FANIN) [28]	M
	Number of Output Data (FANOUT) [28]	M
Cohesion	Lack of Cohesion in Methods (LCOM) [29]	C
	Tight Class Cohesion (TCC) [31]	C
	Loose Class Cohesion (LCC) [31]	C
	Information Flow Based Cohesion (ICH) [26]	C
Abstraction	Number of Abstract Classes/Interfaces (NACI)	P
	Method Inheritance Factor (MIF) [30]	P
	Number of Immediate Base Classes (IFANIN) [28]	C
	Number of Immediate Subclasses (NOC) [29]	C
	Depth of Inheritance Tree (DIT) [29]	C
Encapsulation	Ratio of Public Attributes (RPA)	C
	Ratio of Public Methods (RPM)	C
	Ratio of Static Attributes (RSA)	C
	Ratio of Static Methods (RSM)	C
Documentation	Comment of Lines per Class (CLC)	C
	Ratio Comments to Codes per Class (RCCC)	C
	Comment of Lines per Method (CLM)	M
	Ratio Comments to Codes per Method (RCCM)	M

1) Pairwise comparison of the distribution of metric values: To investigate the effects of factor f on metric m, we test the following null hypothesis for every pair of groups divided by factor f.

$H0_2$: *there is no difference in the distributions of metric values between the two groups of any pairs*.

To examine the difference in the distribution of the metric m values between every two groups, we apply Mann-Whitney U test [32] using the 5% confident level (*i.e.*, p-value$<$0.05). The Mann-Whitney U test assesses whether two independent distributions have equally large values. It does not assume a normal distribution since it is a non-parametric statistical test. As we conduct multiple tests, we also apply Bonferroni correction to the threshold p-value. If there is a statistically significant difference, we reject the null hypothesis and claim that factor f is important to metric m. To quantify the importance, we further compute the effect size.

TABLE IV: p-value of Kruskal Wallis test. Non statistically significant is denoted as n.s., which means p-value is not less than $2.14e$-04 (*i.e.*, $0.05/39/6$).

Category	Metric	Application Domain (AD)	Programming Language (PL)	Age (AG)	Lifespan (LS)	Number of Changes (NC)	Number of Downloads (ND)
Complexity	TLOC	n.s.	n.s.	n.s.	1.94e-05	< 2.2e-16	n.s.
	TNF	n.s.	1.11e-05	n.s.	5.97e-06	< 2.2e-16	n.s.
	TNC	3.41e-05	< 2.2e-16	n.s.	n.s.	8.05e-12	n.s.
	TNM	1.46e-04	< 2.2e-16	n.s.	n.s.	1.03e-11	n.s.
	TNS	n.s.	n.s.	n.s.	6.26e-06	< 2.2e-16	n.s.
	CLOC	< 2.2e-16	< 2.2e-16	< 2.2e-16	1.37e-14	n.s.	< 2.2e-16
	NOM	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16
	NIM	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16
	NIV	< 2.2e-16	< 2.2e-16	< 2.2e-16	5.27e-10	5.76e-08	5.27e-11
	WMC	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16
	NMP	< 2.2e-16	< 2.2e-16	< 2.2e-16	2.93e-07	< 2.2e-16	< 2.2e-16
	CC	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16
	NPATH	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16
	MNL	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	2.22e-12	< 2.2e-16
Coupling	CF	n.s.	n.s.	n.s.	9.55e-05	< 2.2e-16	n.s.
	CBO	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	n.s.	< 2.2e-16
	ICP	< 2.2e-16	< 2.2e-16	8.34e-11	< 2.2e-16	< 2.2e-16	n.s.
	MPC	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	1.50e-04
	RFC	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16
	NMI	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16
	FANIN	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16
	FANOUT	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16
Cohesion	LCOM	< 2.2e-16	< 2.2e-16	< 2.2e-16	5.36e-10	n.s.	6.34e-15
	TCC	< 2.2e-16	< 2.2e-16	1.11e-14	< 2.2e-16	8.87e-14	< 2.2e-16
	LCC	< 2.2e-16	< 2.2e-16	8.09e-15	< 2.2e-16	5.68e-14	< 2.2e-16
	ICH	< 2.2e-16	< 2.2e-16	n.s.	< 2.2e-16	5.60e-12	n.s.
Abstraction	NACI	6.30e-05	< 2.2e-16	n.s.	n.s.	5.78e-09	n.s.
	MIF	n.s.	< 2.2e-16	n.s.	n.s.	n.s.	n.s.
	IFANIN	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	4.75e-05	1.69e-04
	NOC	< 2.2e-16	< 2.2e-16	3.50e-08	8.20e-05	n.s.	1.95e-06
	DIT	< 2.2e-16	< 2.2e-16	< 2.2e-16	1.52e-14	n.s.	< 2.2e-16
Encapsulation	RPA	< 2.2e-16	n.s.	n.s.	n.s.	n.s.	n.s.
	RPM	< 2.2e-16	< 2.2e-16	< 2.2e-16	4.13e-06	< 2.2e-16	< 2.2e-16
	RSA	4.45e-16	3.26e-05	n.s.	3.38e-07	< 2.2e-16	4.85e-07
	RSM	< 2.2e-16	1.83e-08	1.41e-05	< 2.2e-16	n.s.	< 2.2e-16
Documentation	CLC	< 2.2e-16	< 2.2e-16	< 2.2e-16	4.80e-15	1.51e-11	< 2.2e-16
	RCCC	< 2.2e-16	< 2.2e-16	< 2.2e-16	n.s.	< 2.2e-16	< 2.2e-16
	CLM	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16
	RCCM	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16

TABLE V: Mapping Cohen's d to Cliff's δ.

Cohen's Standard	Cohen's d	Pct. of Non-overlap	Cliff's δ
small	0.20	14.7%	0.147
medium	0.50	33.0%	0.330
large	0.80	47.4%	0.474

2) Quantifying the importance of the difference: We apply Cliff's δ as effect size [33] to quantify the importance of the difference, since Cliff's δ is reported [33] to be more robust and reliable than Cohen's d [34]. As Cliff's δ estimates non-parametric effect sizes, it makes no assumptions of a particular distribution. Cliff's δ represents the degree of overlap between two sample distributions [33]. It ranges from -1 (if all selected values in the first group are larger than the second group) to +1 (if all selected values in the first group are smaller than the second group). It is zero when two sample distributions are identical [35].

3) Interpreting the effect sizes: Cohen's d is mapped to Cliff's δ via the percentage of non-overlap as shown in Table V [33]. Cohen [36] states that a medium effect size represents a difference likely to be visible to a careful observer, while a large effect is noticeably larger than medium. In this study, we choose the large effect size as the threshold. If the effect size is large, we suggest that software systems should be split into different groups based on factor f when benchmarking metric m. Otherwise, all software systems can be put in the same group along with factor f.

IV. Case Study Results

In this section, we report and discuss the results of our study.

RQ1: **What context factors impact the distribution of the values of software maintainability metrics?**

Motivations. This question is preliminary to the other question. It determines the number of pairwise tests in the other question. In this research question, we determine if each factor impacts the distribution of each metric values, and should be considered in pairwise comparison.

Approach. To address this research question, we examine each factor individually. For each factor, we divide all software systems into non-overlapping groups as described in Section III-D. On all groups divided by a single factor, we test the null hypothesis $H0_1$ from Section III-F using Kruskal Wallis test with the 5% confident level (*i.e.*, p-value$<$ $2.14e$-04 after Bonferroni correction). If the difference is statistically significant, we reject the null hypothesis $H0_1$ and state that the corresponding factor has an overall impact on the distribution of the values of the corresponding metric.

Findings. We present p-value of Kruskal Wallis test in Table IV. For each factor, statistically significant results indicate the impacting factors. In general, 51% of metrics (*i.e.*, 20 out of 39) are impacted by all six factors. On the other hand, programming language, application domain, and lifespan are three most important factors since they impact over 80% of metrics (*i.e.*, 35, 34, and 33 out of 39, respectively). Moreover, the number of changes, age, and the number of downloads affect more than 70% of metrics (*i.e.*, 31, 28 and 28 out of 39, respectively).

Overall, we conclude that all six factors impact the distribution of the maintainability metric values. The programming language is the most influential factor, since it affects 90% of metrics (*i.e.*, 35 out of 39). In the next research question, we examine types (of programming language, application domain) and levels (of lifespan, the number of changes, age, the number of downloads) in more detail to determine what factors should be considered when benchmarking software maintainability.

RQ2: What guidelines can we provide for benchmarking software maintainability metrics?

Motivations. In **RQ1**, we found that each of the six factors impact the values of at least 75% of metrics. However, considering all six factors when benchmarking software maintainability can result in a number of small groups, and can increase the possibility of duplicated benchmarks. To effectively build benchmarks, we suggest to follow three steps: a) separate software systems into distinct groups to ensure each group contains only systems that share a similar context; b) apply existing approaches (*e.g.*, [14], [21]) to build benchmarks of each group; c) for a given software system, determine which groups it belongs to and apply corresponding benchmarks to evaluate its maintainability. The results of several benchmarks can be aggregated when evaluating the maintainability of software. In this research question, we aim to find the factors that impact the distribution of the maintainability metric values. Such factors can affect the derivation of the thresholds/ranges of the corresponding metrics. Moreover, we provide guidelines in splitting software systems into distinct groups for building benchmarks to measure software maintainability.

Approach. To address each research question, we divide all software systems into non-overlapping groups by each factor, respectively (as described in Section III-D). If examining all possible interactions of all six context factors, the number of groups will be 6,075 ($= 15 \times 5 \times 3 \times 3 \times 3 \times 3$). However, the number of our subject systems is 320, then a large number of groups might be empty. Therefore, interactions of all six context factors are not investigated in this study.

TABLE VI: List of the threshold p-values after Bonferroni correction. The number of pairwise tests required for each context factor is determined by $C(n, 2)$, which denotes the number of 2-combinations from a given set S of n elements. The number of groups by factors AD, PL, AG, LS, NC, and ND are: 15, 5, 3, 3, 3, and 3, respectively. Hence, the number of pairwise tests are: $C(15, 2) = 105$, $C(5, 2) = 10$, $C(3, 2) = 3$, $C(3, 2) = 3$, $C(3, 2) = 3$, and $C(3, 2) = 3$, respectively.

Metric	Number of Pairwise Tests by Each Factor						Total Number of Pairwise	Corrected Threshold
	AD	PL	AG	LS	NC	ND	Tests	p-value
TLOC	0	0	0	3	3	0	6	$8.33e-03$
TNF	0	10	0	3	3	0	16	$3.13e-03$
TNC	105	10	0	0	3	0	118	$4.24e-04$
TNM	105	10	0	0	3	0	118	$4.24e-04$
TNS	0	0	0	3	3	0	6	$8.33e-03$
CLOC	105	10	3	3	0	3	124	$4.03e-04$
NOM	105	10	3	3	3	3	127	$3.94e-04$
NIM	105	10	3	3	3	3	127	$3.94e-04$
NIV	105	10	3	3	3	3	127	$3.94e-04$
WMC	105	10	3	3	3	3	127	$3.94e-04$
NMP	105	10	3	3	3	3	127	$3.94e-04$
CC	105	10	3	3	3	3	127	$3.94e-04$
NPATH	105	10	3	3	3	3	127	$3.94e-04$
MNL	105	10	3	3	3	3	127	$3.94e-04$
CF	0	0	0	3	3	0	6	$8.33e-03$
CBO	105	10	3	3	0	3	124	$4.03e-04$
ICP	105	10	3	3	3	0	124	$4.03e-04$
MPC	105	10	3	3	3	3	127	$3.94e-04$
RFC	105	10	3	3	3	3	127	$3.94e-04$
NMI	105	10	3	3	3	3	127	$3.94e-04$
FANIN	105	10	3	3	3	3	127	$3.94e-04$
FANOUT	105	10	3	3	3	3	127	$3.94e-04$
LCOM	105	10	3	3	0	3	124	$4.03e-04$
TCC	105	10	3	3	3	3	127	$3.94e-04$
LCC	105	10	3	3	3	3	127	$3.94e-04$
ICH	105	10	0	3	3	0	121	$4.13e-04$
NACI	105	10	0	0	3	0	118	$4.24e-04$
MIF	0	10	0	0	0	0	10	$5.00e-03$
IFANIN	105	10	3	3	3	3	127	$3.94e-04$
NOC	105	10	3	3	0	3	124	$4.03e-04$
DIT	105	10	3	3	0	3	124	$4.03e-04$
RPA	105	0	0	0	0	0	105	$4.76e-04$
RPM	105	10	3	3	3	3	127	$3.94e-04$
RSA	105	10	0	3	3	3	124	$4.03e-04$
RSM	105	10	3	3	0	3	124	$4.03e-04$
CLC	105	10	3	3	3	3	127	$3.94e-04$
RCCC	105	10	3	0	3	3	124	$4.03e-04$
CLM	105	10	3	3	3	3	127	$3.94e-04$
RCCM	105	10	3	3	3	3	127	$3.94e-04$

For each pair of groups divided by a factor f, we test the null hypothesis $H0_2$ as discussed in Section III-F using Mann-Whitney U test with the 5% confident level. We apply Bonferroni correction to adjust the threshold p-value based on the findings of **RQ1**. The corrected threshold p-values are shown in Table VI. If the difference is statistically significant, we reject the null hypothesis $H0_2$ and further compute the Cliff's δ effect size to determine the importance of the factor.

If the Cliff's δ effect size is large, we conclude that the

TABLE VII: Cliff's δ and p-value of Mann-Whitney U test of every statistically significant different pairs of groups divided by factors. (investigation of *comlexity* metrics).

Metric	Factor	Group1	Group2	Cliff's δ
TLOC	NC	G_{lowNC}	G_{highNC}	0.498
TNF	NC	G_{lowNC}	$G_{moderateNC}$	0.573
			G_{highNC}	0.639
		$G_{moderateNC}$	G_{highNC}	0.513
TNC	AD	G_{frame}	$G_{network}$	-0.519
			$G_{comm;network}$	-0.759
	PL	G_c	$G_{c\#}$	0.596
			G_{java}	0.667
		G_{pascal}	G_{cpp}	0.560
			$G_{c\#}$	0.729
			G_{java}	0.885
	NC	G_{lowNC}	$G_{moderateNC}$	0.476
			G_{highNC}	0.552
TNM	AD	G_{frame}	$G_{network}$	-0.599
	PL	G_c	$G_{c\#}$	0.614
			G_{java}	0.591
		G_{pascal}	G_{cpp}	0.683
			$G_{c\#}$	0.758
			G_{java}	0.832
	NC	G_{lowNC}	G_{highNC}	0.560
TNS	NC	G_{lowNC}	G_{highNC}	0.541

TABLE VIII: Cliff's δ and p-value of Mann-Whitney U test of every statistically significant different pairs of groups divided by factors. (investigation of *coupling* metrics).

Metric	Factor	Group1	Group2	Cliff's δ
CBO	AD	$G_{comm;network}$	$G_{build;codegen}$	0.482
	PL	G_{pascal}	$G_{c\#}$	0.486
			G_{java}	0.550
RFC	*AD*	$G_{comm;network}$	$G_{network}$	0.524
			$G_{internet}$	0.578
			$G_{sysadmin}$	0.492
			$G_{codegen}$	0.764
			G_{frame}	0.620
			G_{build}	0.703
			G_{swdev}	0.847
			$G_{games;internet}$	0.643
			$G_{internet;swdev}$	0.647
			$G_{comm;internet}$	0.642
		$G_{build;codegen}$	$G_{comm;network}$	-0.923
			$G_{swdev;sysadmin}$	-0.531
	PL	$G_{c\#}$	G_{java}	0.666
CF	NC	G_{lowNC}	G_{highNC}	-0.554
NMI	PL	G_{java}	$G_{c\#}$	-0.516
			G_{pascal}	-0.516

corresponding factor f has a large impact on the distribution of the corresponding metric m. Hence, factor f should be considered when benchmarking metric m.

Findings. To better understand the impact of context factors on different aspects of software maintainability, we report our findings along the six categories of metrics: complexity, coupling, cohesion, abstraction, encapsulation and documentation.

1) Complexity.

As shown in Table VII, the factor impacting TLOC, TNF, and TNS is the number of changes. The factors impacting TNC and TNM are application domain, programming language, and the number of changes. Overall, the distributions of metric values in the *complexity* category are strongly impacted by three context factors: application domain, programming language, and the number of changes.

TABLE IX: Cliff's δ and p-value of Mann-Whitney U test of every statistically significant different pairs of groups divided by factors. (investigation of *cohesion* metrics).

Metric	Factor	Group1	Group2	Cliff's δ
LCOM	AD	$G_{network}$	$G_{comm;network}$	0.552

TABLE X: Cliff's δ and p-value of Mann-Whitney U test of every statistically significant different pairs of groups divided by factors. (investigation of *abstraction* metrics).

Metric	Factor	Group1	Group2	Cliff's δ
NACI	PL	G_{java}	G_{pascal}	-0.773
IFANIN	AD	$G_{comm;network}$	$G_{network}$	0.794
			$G_{internet}$	0.751
			$G_{sysadmin}$	0.657
			G_{comm}	0.776
			$G_{codegen}$	0.780
			G_{frame}	0.748
			G_{build}	0.738
			G_{swdev}	0.736
			G_{games}	0.598
			$G_{games;internet}$	0.620
			$G_{swdev;sysadmin}$	0.828
			$G_{internet;swdev}$	0.704
			$G_{comm;internet}$	0.745
			$G_{build;codegen}$	0.824
		G_{games}	$G_{swdev;sysadmin}$	0.486
	PL	G_{java}	G_{cpp}	-0.514
			G_{pascal}	-0.708
DIT	AD	$G_{comm;network}$	$G_{network}$	0.820
			$G_{internet}$	0.861
			$G_{sysadmin}$	0.772
			G_{comm}	0.907
			$G_{codegen}$	0.954
			G_{frame}	0.899
			G_{build}	0.870
			G_{swdev}	0.962
			G_{games}	0.839
			$G_{games;internet}$	0.746
			$G_{swdev;sysadmin}$	0.910
			$G_{internet;swdev}$	0.915
			$G_{comm;internet}$	0.910
		$G_{build;codegen}$	$G_{sysadmin}$	-0.549
			$G_{comm;network}$	-0.983
			$G_{games;internet}$	-0.517
MIF	PL	G_{java}	G_{cpp}	-0.777
			$G_{c\#}$	-0.849
			G_{pascal}	-0.666
		G_{cpp}	$G_{c\#}$	0.657

2) Coupling.

As shown in Table VIII, the factors impacting CBO and RFC are application domain and programming language. The factor impacting CF (respectively NMI) is the number of changes (respectively programming language). Overall, the distributions of metric values in the *coupling* category are strongly impacted by three context factors: application domain, programming language, and the number of changes.

TABLE XI: Cliff's δ and p-value of Mann-Whitney U test of every statistically significant different pairs of groups divided by factors. (investigation of *encapsulation* metrics).

Metric	Factor	Group1	Group2	Cliff's δ
RPA	AD	G_{build}	$G_{internet}$	0.483
			$G_{codegen}$	0.558
			G_{frame}	0.619
			G_{swdev}	0.576
			G_{games}	0.682
			$G_{swdev;sysadmin}$	0.543
			$G_{internet;swdev}$	0.550
			$G_{comm;internet}$	0.505
			$G_{build;codegen}$	0.527
RSM	AD	$G_{comm;network}$	$G_{comm;internet}$	0.710

TABLE XII: Cliff's δ and p-value of Mann-Whitney U test of every statistically significant different pairs of groups divided by factors. (investigation of *documentation* metrics).

Metric	Factor	Group1	Group2	Cliff's δ
RCCC	AD	$G_{build;codegen}$	$G_{network}$	-0.513
	PL	G_{java}	G_{pascal}	-0.611

3) Cohesion.

As shown in Table IX, the factor impacting LCOM is application domain. Overall, the distributions of metric values in the *cohesion* category are strongly impacted by application domain only.

4) Abstraction.

As shown in Table X, the factor impacting NACI and MIF is programming langauge. The factor impating DIT is application domain. The factors impacting IFANIN are application domain and programming langauge. Overall, the distributions of metric values in the *abstraction* category are strongly impacted by application domain and programming language.

5) Encapsulation.

As shown in Table XI, the factor impacting RPA and RSM is application domain. Overall, the distributions of metric values in the *encapsulation* category are strongly impacted by application domain only.

6) Documentation.

As shown in Table XII, the factors impacting RCCC are application domain and programming language. Overall, the distributions of metric values in the *documentation* category are strongly impacted by application domain and programming language.

Guidelines for Benchmarking Maintainability Metrics.

Based on our findings, application domain, programming language, and the number of changes strongly impact the distribution of maintainability metric values.

When benchmarking the 39 metrics, we suggest to partition software systems into 13 groups: 1) five groups along application domain (*i.e.*, G_{build}, G_{games}, G_{frame}, $G_{build;codegen}$, and $G_{comm;network}$); 2) five groups along programming language (*i.e.*, G_c, G_{cpp}, $G_{c\#}$, G_{java}, and G_{pascal}); and 3) three groups along the number of changes (*i.e.*, G_{lowNC}, $G_{moderateNC}$, and G_{highNC}). When benchmarking metrics from a particular category, we provide detailed suggestions in Table XIII. Moreover, our approach can be applied to other software metrics and other software systems for generating guidelines on building benchmarks of such software metrics.

TABLE XIII: Guidelines on partitioning software systems when building metric based benchmarks.

Metric Category	Factor	Group
Complexity	AD	G_{frame} and others
	PL	G_c, G_{pascal} and others
	NC	G_{lowNC}, $G_{moderateNC}$, and G_{highNC}
Coupling	AD	$G_{comm;network}$, $G_{build;codegen}$, and others
	PL	G_{pascal}, G_{java}, and others
	NC	G_{lowNC}, $G_{moderateNC}$, and G_{highNC}
Cohesion	AD	$G_{comm;network}$, and others
Abstraction	AD	$G_{comm;network}$, G_{games}, $G_{build;codegen}$, and others
	PL	G_{java}, G_{cpp}, and others
Encapsulation	AD	G_{build}, $G_{comm;network}$, and others
Documentation	AD	$G_{build;codegen}$, and others
	PL	G_{java}, and others

V. THREATS TO VALIDITY

We now discuss the threats to validity of our study following common guidelines provided in [37].

Threats to conclusion validity concern the relation between the treatment and the outcome. Our conclusion validity threats are mainly due to sampling errors. Since stratified sampling was performed only along application domain, sampled software systems may not well represent along other five factors. Some differences along these factors, thus, may not be detected, and the detected differences are likely to be only a subset of differences. We plan to stratify along other factors.

Threats to internal validity concern our selection of subject systems and analysis methods. We randomly sample 320 software systems from SourceForge, some of the findings might be specific to software systems hosted on SourceForge. Future studies should consider using software systems from other hosts, and even commercial software systems.

Threats to external validity concern the possibility to generalize our results. Some of the findings might not be directly applicable to different software systems. Yet our approach can be applied to find guidelines for benchmarking maintainability of different open source and commercial software systems.

Threats to reliability validity concern the possibility of replicating this study. We attempt to provide all the necessary details to replicate our study. SourceForge is publicly available to obtain the same data. We make our data and R script available[13] as well.

VI. CONCLUSION

In this work, we perform a large scale empirical study to investigate how the six context factors affect the distribution

[13]https://bitbucket.org/serap/contextstudy

of maintainability metric values. We apply statistical methods (*i.e.*, Kruskal Wallis test, Mann-Whitney U test and Cliff's δ effect size) to analyze 320 software systems, and provide empirical evidence of the impact of context factors on the distribution of maintainability metric values. Our results show that all six context factors impact the distribution of the values of 51% of metrics. The most influential factors are application domain, programming language, and the number of changes. Based on our findings, we further provide guidelines on how to group software systems according the six context factors. We expect our findings to help software benchmarking and other software engineering methods using the 39 software maintainability metrics.

In the future, we plan to extend our study using more software systems from SourceForge, GoogleCode, and GitHub, and to perform stratified sampling along all six context factors. Moreover, we want to derive the thresholds and ranges of metric values based on our findings and provide benchmarks to measure maintainability. We also want to verify whether our findings can help sample representative software systems for empirical studies.

REFERENCES

[1] N. E. Fenton and M. Neil, "Software metrics: roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 357–370.

[2] T. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering (TSE)*, vol. SE-2, no. 4, pp. 308 – 320, dec. 1976.

[3] M. Lorenz and J. Kidd, *Object-oriented software metrics: a practical guide*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994.

[4] K. Erni and C. Lewerentz, "Applying design-metrics to object-oriented frameworks," in *Proceedings of the 3rd International Software Metrics Symposium (METRICS'96)*, mar 1996, pp. 64 –74.

[5] L. C. Briand, J. Wüst, S. V. Ikonomovski, and H. Lounis, "Investigating quality factors in object-oriented designs: an industrial case study," in *Proceedings of the 21st International Conference on Software Engineering (ICSE'99)*. New York, NY, USA: ACM, 1999, pp. 345–354.

[6] L. B. L. De Souza and M. D. A. Maia, "Do software categories impact coupling metrics?" in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR'13)*. Piscataway, NJ, USA: IEEE Press, 2013, pp. 217–220.

[7] T. Dybå, D. I. Sjøberg, and D. S. Cruzes, "What works for whom, where, when, and why?: on the role of context in empirical software engineering," in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'12)*. New York, NY, USA: ACM, 2012, pp. 19–28.

[8] S. Benlarbi, K. El Emam, N. Goel, and S. Rai, "Thresholds for object-oriented measures," in *Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE'00)*, 2000, pp. 24 –38.

[9] R. Shatnawi, "A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems," *IEEE Transactions on Software Engineering (TSE)*, vol. 36, no. 2, pp. 216 –225, mar 2010.

[10] K.-A. Yoon, O.-S. Kwon, and D.-H. Bae, "An approach to outlier detection of software measurement data using the k-means clustering method," in *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM'07)*, sept. 2007, pp. 443 –445.

[11] T. Alves, C. Ypma, and J. Visser, "Deriving metric thresholds from benchmark data," in *Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM'10)*, Sept. 2010, pp. 1 –10.

[12] S. Herbold, J. Grabowski, and S. Waack, "Calculation and optimization of thresholds for sets of software metrics," *Journal of Empirical Software Engineering*, vol. 16, no. 6, pp. 812–841, Dec. 2011.

[13] T. Bakota, P. Hegedus, P. Kortvelyesi, R. Ferenc, and T. Gyimothy, "A probabilistic software quality model," in *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM'11)*, 2011, pp. 243–252.

[14] K. A. M. Ferreira, M. A. S. Bigonha, R. S. Bigonha, L. F. O. Mendes, and H. C. Almeida, "Identifying thresholds for object-oriented software metrics," *Journal of Systems and Software*, vol. 85, no. 2, pp. 244–257, Feb. 2012.

[15] Y. Zou and K. Kontogiannis, "Migration to object oriented platforms: a state transformation approach," in *Proceedings of the 18th International Conference on Software Maintenance (ICSM'02)*, 2002, pp. 530 – 539.

[16] S. E. Sim, S. Easterbrook, and R. C. Holt, "Using benchmarking to advance research: a challenge to software engineering," in *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 74–83.

[17] M. Lanza, R. Marinescu, and S. Ducasse, *Object-Oriented Metrics in Practice*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[18] P. Louridas, D. Spinellis, and V. Vlachos, "Power laws in software," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 18, no. 1, pp. 2:1–2:26, Oct. 2008.

[19] I. Herraiz, D. M. Germán, and A. E. Hassan, "On the distribution of source code file sizes," in *Proceedings of the 6th International Conference on Software and Data Technologies (ICSOFT'11)*, 2011, pp. 5–14.

[20] L. Sánchez-González, F. García, F. Ruiz, and J. Mendling, "A study of the effectiveness of two threshold definition techniques," in *EASE*, 2012, pp. 197–205.

[21] R. Baggen, J. Correia, K. Schill, and J. Visser, "Standardized code quality benchmarking for improving software maintainability," *Software Quality Journal*, vol. 20, pp. 287–307, 2012.

[22] A. Capiluppi, P. Lago, and M. Morisio, "Characteristics of open source projects," in *Proceedings of the 7th European Conference on Software Maintenance and Reengineering (CSMR'03)*, Mar. 2003, pp. 317 – 327.

[23] K. Johari and A. Kaur, "Effect of software evolution on software metrics: an open source case study," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 5, pp. 1–8, Sep. 2011.

[24] A. Mockus, "Amassing and indexing a large sample of version control systems: Towards the census of public source code history," in *6th IEEE International Working Conference on Mining Software Repositories*, ser. MSR'09, may 2009, pp. 11 –20.

[25] J. Howison, M. Conklin, and K. Crowston, "Flossmole: A collaborative repository for floss research data and analyses," *International Journal of Information Technology and Web Engineering*, vol. 1, pp. 17–26, 07/2006 2006.

[26] K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical study of object-oriented metrics," *Journal of Object Technology*, vol. 5, no. 8, pp. 149–173, 2006.

[27] B. Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*, ser. Prentice-Hall Object-Oriented Series. Prentice Hall, 1996.

[28] Scitools, "Metrics computed by understand," `http://www.scitools.com/documents/metricsList.php`.

[29] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering (TSE)*, vol. 20, no. 6, pp. 476 –493, jun 1994.

[30] R. Harrison, S. Counsell, and R. Nithi, "An evaluation of the mood set of object-oriented software metrics," *IEEE Transactions on Software Engineering (TSE)*, vol. 24, no. 6, pp. 491 –496, jun 1998.

[31] L. C. Briand, J. W. Daly, and J. K. Wüst, "A unified framework for coupling measurement in object-oriented systems," *IEEE Transactions on Software Engineering (TSE)*, vol. 25, no. 1, pp. 91–121, Jan. 1999.

[32] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures, Fourth Edition*. Chapman & Hall/CRC, Jan. 2007.

[33] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohen's d for evaluating group differences on the nsse and other surveys?" in *Annual Meeting of the Florida Association of Institutional Research*, February 2006, pp. 1–33.

[34] J. Cohen, *Statistical power analysis for the behavioral sciences : Jacob Cohen.*, 2nd ed. Lawrence Erlbaum, Jan. 1988.

[35] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions." *Psychological Bulletin*, vol. 114, no. 3, pp. 494–509, Nov. 1993.

[36] J. Cohen, "A power primer." *Psychological Bulletin*, vol. 112, no. 1, pp. 155–159, 1992.

[37] R. K. Yin, *Case Study Research: Design and Methods - Third Edition*, 3rd ed. SAGE Publications, 2002.