The Uncomfortable Discrepancies of Software Metric Thresholds and Reference Values in Literature

Eudes de Castro Lima, Antônio Maria P. de Resende
Department of Computer Science
Universidade Federal de Lavras (UFLA)
Lavras, Minas Gerais, Brasil
e-mail: comp.eudes@gmail.com, tonio@dcc.ufla.br

Timothy C. Lethbridge
School of Information Technology and Engineering
University of Ottawa
Ottawa, Canada
e-mail: tcl@eecs.uottawa.ca

***Abstract*— Software metrics perform a crucial role in the software industry because they provide measures needed to control software process and product, such as software quality, complexity, maintainability, and size. Measuring software allows one to diagnose whether the project is within expected norms or there is a deviation. However, many publications present metrics but omit thresholds or reference values that would give guidance about their ideal limits and range. Metrics might be used more frequently and effectively if they were accompanied by reliable reference values. We therefore present a Systematic Literature Review to find research that presents such reference values and thresholds. The keyword search phase of the systematic review generated 6.654 articles from IEEE Xplore, ACM Digital Library, Ei Compendex, SCOPUS, and Elsevier Science Direct. Further filtering narrowed this to only 19 articles actually disucssing thresholds and reference values. We present an analysis of these papers, including a comparison highlighting discrepancies in the reference values and thresholds. The results serve as a starting point to guide further research.**

Keywords- software metrics; software measures; thresholds; reference values; systematic literature review.

I. INTRODUCTION

In medicine, when a blood test is done, the values obtained are compared with their respective reference intervals printed beside the results. If there is any abnormality in the results then the doctor makes a diagnosis, defines the disease, and determines the type and dose of medicine the patient should take. There are reference values for most tests, allowing the diagnosis of patients. However, in software engineering, there is still a long journey to obtain these values and achieve maturity based on measures.

Software metrics perform an important role in the software industry because they provide measures for software features, such as maintainability, reusability, portability, readability, correctness, complexity and so on. These measures provide the software engineer, software architects and project managers the current state of the software. The measures allow diagnosing of projects, products and processes, and check whether the values of measures are within the expected norm or there is unexpected deviation.

Over the years, a variety of software metrics [1]-[11] and automated tools for measuring [12][13][14] have been proposed. However, despite the importance of software metrics, most have not been widely applied in industry [15][16]. It is believed that one reason is the lack of reference values and thresholds for most metrics [17].

A threshold defines a point that should (or not) be exceeded due to (un)desirable effects involved. A reference value or range gives objectives for what should be achieved or defines value sets classified qualitatively; for instance the classification could be bad, regular and good. In this paper, the term 'reference value' will be used for both in most of what follows, unless context requires otherwise.

In some cases, the reference values are known, but not widely accepted. This causes an uncertainty which, according to [16], inhibits the popularization of software metrics.

Reference values for metrics enable interpretation of the results of measurement. It is through comparing measures to reference values that software engineers can verify that the project, product and process meets a desired standard or, that the project is improving, worsening or stable.

Various authors [18]-[22] have proposed reference values for software metrics and techniques for deriving them. There are articles, such as [19][21][23], which provide benchmarks based on "experience" (tacit knowledge) without any statistical or technical analysis that supports the claim. However, since they were obtained in a specific context, published reference values tend not to be generalizable beyond the context of their inception.

In this work, the results of a systematic literature review (SLR) of software metrics are presented, focusing on reference values. The SLR selection process resulted in selection of 19 articles, out of 6.654 considered. In subsequent sections we summarize these articles and present the reference values cited or calculated in the articles . We discuss certain differences in metric interpretations. We also comment on the amount and type of software used to calculate and validate the reference values. We then present a comparison of the discrepancies among reference values proposed in those articles. Finally, we suggest future work that would promote improvements in software metrics and measurements.

The SLR methodology has proven very useful software engineering researchers. It provides a documented and repeatable process to identify the state of the art about some issues of researchers' interest.

The structure of this paper is as follows. Section II describes SLRs in general, the SLR construction process, the protocol used and the results obtained from this SLR. Section V presents the comparison analysis and discussion of the

 ISBN: 978-1-61208-498-5

articles as a group. Section VI presents the main conclusions obtained in this work, as well as contributions and future work.

II. SYSTEMATIC REVIEW PROCESS

A systematic literature review is an evidence-based technique originating in medicine and medical sciences [24]. This technique has been employed in several areas including software engineering.

An SLR involves several distinct activities [25]. In the literature, it is possible to find different suggestions for the number and order of activities undertaken in a systematic review. In [24][25][26] the authors present an SLR process consisting of three main phases: planning, execution and analysis of results. This section presents the application of the SLR, following the three-phase approach.

1. Planning

This section presents the planning phase.

- *Objectives:* To perform a survey of scientific papers that discusses software metrics that have ranges or specific reference values associated with them.
- *Research questions:* What software metrics have values or ranges of reference assigned to them? What values or ranges have been identified in the literature?
- *Keywords:* The following keywords were adopted: Software metric, measure, measuring, threshold, reference value, value, range, limit.

  Search string: the search string was compiled from the keywords, linking them logically: (software) AND (metric OR metrics OR measure OR measures OR measuring) AND ("reference value" OR "reference values" OR ranges OR thresholds OR limits OR range OR threshold OR limit).
- *Search method sources:* Web sites of virtual scientific libraries.
- *List of research sources:* IEEE Xplore (http://ieeexplore.ieee.org), Elsevier Science Direct (www.sciencedirect.com), Scopus (www.scopus.com), ACM Digital Library (http://dl.acm.org), and EiCompendex (www.engineeringvillage2.org).
- *Types of articles:* Papers considered are those relating to software metrics, including comparisons and analyzes.
- *Language of articles:* The articles must be in English.
- *Criteria for inclusion or exclusion of articles:* Articles should: i) Be available for download as full papers; ii) Provide reference values for software metrics; and iii) Have been published between the years 1990 and 2015.

It is known that the search string used can return a lot of articles or limit the results as well. So, in this investigation, the results expected are papers that contain the words present in search string. A string search containing the name of a specific metrics was not used. For instance, Depth Inheritance Tree, Response for Classes, Coupling Between Objects, Number of Children, Weighted Methods per Class, LCOM and others could be inserted in the search string. Considering the amount of metrics, the length of the search string, the volume of articles that need to be retrieved and the data need to be processed, the work must be separated for each metric.

2. Execution

The execution was divided into four steps, as suggested in [27] called initial selection, primary selection, secondary selection, and obtaining and evaluation of scientific papers.

- *Initial selection (obtaining of articles):* Searches are conducted in databases defined in the protocol; then the results are summarized according to previously established criteria. This process is iterative, i.e. the search can be readjusted and run again, if the results are not reasonable.
- *Primary selection:* This is the first filtering of the results. Usually the Title and Keywords of articles are read to verify compliance with the criteria for inclusion and exclusion.
- *Secondary selection:* This is the second filtering of the results. This step aims to eliminate irrelevant results by reading the abstracts and conclusions of the articles, and checking compliance with the criteria for inclusion and exclusion.
- *Results organization:* The results are tabulated in a way that favors a quick visual analysis.

Step 1 – Initial selection

The initial selection was conducted by searching in the databases mentioned above. Filters were carried out during searching activity to restrict the results according to year (the period between 1990 and 2015), language (English), and discipline (computer science and/or software engineering). Scientific articles were searched for using our search strings applied to titles, abstracts and keywords.

Because of the characteristics of the search engines for some databases, the search strings defined in the protocol required slight change, but their semantics were retained. In some situations, it was necessary to include the query string parameters. For instance, in the ACM Digital Library it was necessary to divide the search string into three, to obtain a plausible result for analysis. The searches were performed on November 10 and 26, 2014.

Table I presents the exact search strings used in the SLR for each database.

As a result of that search, 6.654 scientific articles were found, as shown in the second column of Table II. The tool JabRef version 2.7.2 [28] was used to manage the list of articles.

TABLE I. SEARCH STRINGS USED IN PRIMARY SELECTION

Databases	Search strings	
IEEE Xplore	((software) AND (metric OR metrics OR measure OR measures OR measuring) AND ("reference value" OR "reference values" OR ranges OR thresholds OR limits OR range OR threshold OR limit))	
Elsevier Science Direct	pub-date > 1989 and TITLE-ABS-KEY((software) AND (metric OR metrics OR measure OR measures OR measuring) AND ("reference value" OR "reference values" OR ranges OR thresholds OR limits OR range OR threshold OR limit))[All Sources(Computer Science)]	
Ei Compendex	(((((software) AND (metric OR metrics OR measure OR measures OR measuring) AND ("reference value" OR "reference values" OR ranges OR thresholds OR limits OR range OR threshold OR limit)) WN KY) AND (({computer software} OR {software engineering}) WN CV)) AND (((english) WN LA) AND (1990-2015) WN YR))	
Scopus	TITLE-ABS-KEY((software) AND (metric OR metrics OR measure OR measures OR measuring) AND ("reference value" OR "reference values" OR ranges OR thresholds OR limits OR range OR threshold OR limit)) AND PUBYEAR > 1989 AND (LIMIT-TO(LANGUAGE, "English")) AND (LIMIT-TO(SUBJAREA, "COMP"))	
ACM Library	String 1	("software measure*") AND ("reference value*" OR range* OR threshold* OR limit*)
	String 2	("software measuring") AND ("reference value*" OR range* OR threshold* OR limit*)
	String 3	("software metric*") AND ("reference value*" OR range* OR threshold* OR limit*)

TABLE II- RESULTS AFTER APPLYING SELECTIONS

Data Bases	Initial Selection	Primary Selection	Secondary Selection			Selected
			Irlvt	Rpt	Incompl	
IEEE	3.266	91	80	0	0	11
Elsevier	180	27	24	0	0	3
Compendex	1.254	33	19	11	0	3
Scopus	1.687	54	37	16	0	1
ACM	267	37	33	3	0	1
Total	**6.654**	**242**	**193**	**30**	**0**	**19**

Step 2 – Primary selection

After the initial selection was performed (step 1), the scientific articles were submitted to primary selection, where *titles*, *keywords* and *abstracts* were filtered and analyzed manually.

During filtering, it was found that a large proportion of the articles belonged to other areas of computer science and did not meet the purposes of this SLR. Hence, the number of scientific articles decreased from 6654 (obtained in the initial selection) to 242. This is shown in column 3 of Table II.

In an SLR, it is usual for the initial search to return a large number of irrelevant articles that neither respond to the research questions nor are unrelated to the theme in question [26].

Step 3 – Secondary selection

In secondary selection, the 242 scientific articles selected in the primary selection (Step 2) passed an inspection in which both *introductions* and *conclusions* were read. At this stage, relevance, repetitiveness and completeness were checked.

Out of 242 articles selected earlier, 193 papers were considered irrelevant, because they did not correspond to the objectives of this SLR; 30 articles were considered repeated, because they were found in more than one database; no article was considered incomplete, and all items surveyed were available. Finally, 19 scientific articles passed the selection criteria. In other words, 19 papers were found that had clearly stated an intent to define or analyze thresholds or reference values in their title, abstract or introduction. Table II presents a summary of the results in its rightmost four columns.

Step 4 - Obtaining and evaluation of scientific papers

Those 19 papers were read and discussed one by one, and data were gathered in order to show the state of the art around the research theme. Table III presents the relevant scientific articles that answer the research questions set out in the protocol. Section 3 presents the analysis and discussion of papers identified.

3. Results analysis

This SLR indicates that the number of papers discussing reference values for software metrics has increased in recent years. One of the factors contributed to that increase is likely the market demand for quality products. The SLR shows that 57.8% of scientific papers were published since 2009.

TABLE IV - RELEVANT INFORMATION OF ARTICLES IDENTIFIED IN SLR.

ID	Classification	Articles or tools referenced	Values empirically validated	Technique	Context
A	Type I	[13]	no	experience	specific
B	Type II	[10]	no	distribution analysis	generic
C	Type I	[18]	no	statistical analysis	specific
D	Type I	[9][21]	negative	experience	specific
E	Type II	[31]	negative	statistical analysis	specific
F	Type I	ISM	no	logistic regression	specific
G	Type II	-	yes	distribution analysis	specific
H	Type I	[44]	no	experience	generic
I	Type II	-	yes	statistical analysis	specific
J	Type II	[21]	yes	statistical analysis	specific
K	Type II	-	no	statistical analysis	specific
L	Type II	-	yes	statistical analysis	specific
M	Type II	-	no	ROC courves	specific
N	Type II	-	no	experience	generic
O	Type II	[45]	no	ROC courves	specific
P	Type II		no	experience	specific
Q	Type II	[45]	no	statistical analysis	generic
R	Type I	[31][46] [23][9]	no	learning machine	specific
S	Type II	[13]	no	experience	specific

A total of 66 metrics having thresholds were identified from the 19 papers. Among the metrics identified, there are metrics specific to the OO paradigm as well as traditional metrics adapted to the OO paradigm, such as LOC and cyclomatic complexity. In total, 57.4% of the papers refer to OO metrics specifically and 82.5% of them come from the CK metrics suite [4].

The IEEE Xplore database presented the most relevant articles for this research, with 58% of studies. The databases with the lowest number of relevant studies were Scopus and

 ISBN: 978-1-61208-498-5

the ACM Library, with 5% of scientific articles each. Table IV summarizes the articles analyzed. The first column gives the reference number (see Table III).

TABLE III. RELEVANT ARTICLES THAT ANSWER THE RESEARCH QUESTIONS SET OUT IN THE PROTOCOL.

ID	Year	Title	Ref.	Base
A	2009	An outlier detection algorithm based on object-oriented metrics thresholds	[29]	IEEE
B	2010	Deriving metric thresholds from benchmark data	[18]	IEEE
C	2011	Benchmark-Based Aggregation of Metrics to Ratings	[30]	IEEE
D	2000	Thresholds for object-oriented measures	[31]	IEEE
E	2002	The optimal class size for object-oriented software	[32]	IEEE
F	2009	Clustering and Metrics Thresholds Based Software Fault Prediction of Unlabeled Program Modules	[33]	IEEE
G	2003	A metrics suite for measuring reusability of software components	[34]	IEEE
H	2007	Observing Distributions in Size Metrics: Experience from Analyzing Large Software Systems	[35]	IEEE
I	1997	Software metrics model for quality control	[36]	IEEE
J	2010	A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems	[37]	IEEE
K	2014	Extracting relative thresholds for source code metrics	[38]	IEEE
L	2011	Identifying thresholds for object-oriented software metrics	[20]	Elsevier
M	2011	Class noise detection based on software metrics and ROC curves	[39]	Elsevier
N	2011	Improving the applicability of object-oriented class cohesion metrics	[40]	Elsevier
O	2010	Finding software metrics threshold values using ROC curves	[22]	Compendex
P	1992	Software metrics for object-oriented systems	[19]	Compendex
Q	2005	An empirical exploration of the distributions of the Chidamber and Kemerer object-oriented metrics suite	[41]	Compendex
R	2011	Calculation and optimization of thresholds for sets of software metrics	[42]	Scopus
S	2010	Estimation of Software Reusability: An Engineering Approach	[43]	ACM

The second column categorizes the papers into: i) Type I - studies that use existing reference values to achieve a goal, such as outlier detection and predicting failures, and ii) Type-II studies that aim to establish or optimize reference values. Among the identified articles, 31.6% use existing thresholds and 68.4% aim to identify or optimize thresholds as shown in Table IV.

The thresholds classified as Type I and presented by selected papers were gathered from tool documentation or from other studies that they had referenced. The thresholds obtained from tools such as: McCabe IQ, or ISM are hard to reproduce because the tools are not readily available and some thresholds were determined “by authors experience”.

The labels "no", "yes" and "negative" shown in Table IV mean respectively that "there was no validation", "there was validation", or "there was validation, but the result states that the reference values are bad values". Articles D and E [31][32] had negative validation, representing 10.5% of the articles. A total of 21.1% of articles validated the thresholds and reference values, and 68.4% did not validate them. These results are undesirable, because only 21.1% validated values and only article L [20] out of 21.1% were classified as general context. The other articles were considered neither validated nor general. Software engineering should have well validated thresholds in order to support software engineers during the development process.

Other articles had used the thresholds to validate only the method used to discover thresholds, but they did not validate their own thresholds as presented. This was the case for articles R and S [42][43].

Regarding the techniques used to obtain the thresholds, as Lanza and Marinescu indicated in [16], there are two main approaches: professional experience and statistical analysis. Of the articles analyzed, 31.6% obtained thresholds through experience, i.e., the authors determined arbitrarily and subjectively the thresholds, and 68.4% obtained them through statistical analysis. Methods like machine learning and error models were classified as statistical analysis approaches.

The context was classified as generic and specific. The 'generic' label indicates the reference values fulfill all of the following criteria: a) Three or more systems; b) more than 50% of systems are developed by people different from the authors, c) more than one domain, and d) more than one programming language. Otherwise the label 'specific' is used. A total of 79% of selected papers were classified as specific, and 21% were classified as general.

During the analysis process, several methods were found to calculate thresholds. These include experience, statistical analysis, error models, clustering, distribution analysis, and machine learning.

Most of the papers would not be amenable to replication due to incomplete details such as missing versions of systems, names of systems, details about applied metric interpretation to measure software and so on. Those details should be included in articles. In fact, it is necessary to establish a protocol to guide authors to supply that information, allowing replication and validation of research of this kind.

Several articles did not define precise instructions for how metrics were counted in papers. For instance, what is the difference between ‘comments’ and ‘lines of comments’? How were lines of comment blocks counted? And how were lines counted that had both code and comment?

Another difficulty faced was determining whether a value refers to a minimum or maximum, for instance in Schneidewind’s article (1997) [36].

In [37], the authors used three versions of Eclipse to determine values. However, using different versions of the same software will not result in the same level of generality as if completely different systems had been used. The same argument can be made when multiple systems in the same domain are analyzed.

In next section, some metrics are analyzed considering the values found in the articles. The reference values presented by article E [32] are results from a negative validation meaning that values are invalid to use.

III. COMPARING REFERENCE VALUES PRESENTED BY PAPERS

After reading all selected papers, the gathered reference values are presented in tables below with columns labelled

metric, reference, value, and nature of measure. Respectively, each table contains the name of the metric evaluated, the reference to the paper that presented the reference value, the reference value presented or proposed to the metric mentioned, and the nature of measure that represents the meaning of the value presented like maximum, minimum, desirable, good, bad, typical, etc.

In this section, the reader will note the existence of different reference values for the same metric.

1. *Weighted Methods per Class (WMC)*

Two interpretations for the WMC metric were found. The first interpretation, called here WMC1, is calculated by summing the complexity of each method in a class and assuming the complexity of each method is 1. That means the WMC is a simply counting the number of methods (each method has complexity 1). The second interpretation, called here WMC2, is calculated by summing the McCabe Cyclomatic Complexity of the methods.

Table V presents just one reference value that was found for WMC1 and several different values for WMC2. For the same interpretation, the WMC2 threshold could be 20 or 100 as cited and calculated, respectively, in [22]. This situation makes the work of software engineers difficult, since they will not know what value should be used as a threshold in their projects.

TABLE V - VALUES OF WMC METRIC

Metric	Ref.	Value	Nature of Measure
WMC1 - Counting methods	A	14	Max
WMC2 - Sum complexities	D	100	Max
	K	100	Max
	K	20	Max
	N	24	Max
	N	100	Max
	R	100	Max
	S	20 and 100	Desirable and Max
WMC - not defined	K	32	80% quantiles (relative threshold)

2. *Depth of Inheritance Tree (DIT)*

The papers presented moderate differences among suggested DIT thresholds. This metric measures the maximum depth of the inheritance hierarchy in a system. In Table VI, it is observed that values from 6 to 10 are most commonly found to be the maximum suggested value or upper threshold. This is still a large range, so further research is needed to determine how much worse a system would be if it had a DIT of 10 vs. 6.

TABLE VI - VALUES OF DIT METRIC.

Ref.	Value	Nature of Measure
A	7	Max
D	6	Max
L	2	Typical
Q	10	Max
Q	6 (in Java or C++)	Max
S	3 and 6	Desirable and Max

3. *Cyclomatic Complexity (CC)*

The Cyclomatic Complexity metric is the number of linearly independent paths in program flow and has significantly different reference values in the papers studied, as shown in Table XII.

TABLE XII - VALUES OF CICLOMATIC COMPLEXITY METRICS.

Metrics	Ref.	Value	Nature of Measure
Cyclomatic Complexity Per Method	B	<=6]6;8]]8;14] >14	low risk moderate risk high risk very-high risk
	F	10	It was impossible check it, because original reference on web is not available
	M	P1 - 3 P2 - 5 P3 - 5 P4 - 3 P5 - 4	Best value for each dataset P1, P2,…, P5
	P	10	Max
	R	C - 24 C++ - 10 C# - 10	Max
Cyclomatic Complexity per Module	P	100	Max Value, considering 10 methods and each one supporting max complexity equal 10.
Design Complexity Per Module - Number of paths including calls to other modules	M	P1 - 3 P2 - 3 P3 - 3 P4 - 3 P5 - 3	Best value for each dataset P1, P2,…, P5

4. *Number of Children (NOC)*

NOC represents the number of children that any given class has. In Table VIII, it is observed once again that there are different values for the maximum value or upper threshold ranging from 3 to 10.

TABLE VIII - VALUES OF NOC METRICS.

Ref.	Value	Nature of Measure
A	3	Max
Q	10	Max
Q	]4, 6[Java and <6 C++	Desirable and Max

5. *Lack Of Cohesion Methods (LCOM)*

LCOM measures lack of cohesion and has several interpretations and different names as shown in Table IX. As different cohesion views appeared over time, new metrics were developed. Article [40] explains the subtle difference among the various LCOM metrics.

TABLE IX - VALUES OF LCOM 1, 2, 3, 4, 5 AND LOCM METRICS.

Metric	Ref.	Value	Nature of Measure
LCOM1	N	42 and 21	Mean and 75th percentile
LCOM2	L	0, [10;20] and >20	Intervals mean: Good, Regular and Bad
	N	27 and 8	Mean and 75th percentile
LCOM3	N	1.67 and 2	Mean and 75th percentile
LCOM4	N	1.62 and 2	Mean and 75th percentile
LCOM5	N	0.76 and 1	Mean and 75th percentile
LOCM (McCabe Tools)	A	75	Max
LCOM (not defined)	K	36	80% quantiles (relative threshold)

6. *Operator and Operand Countings*

Halstead`s metrics count Unique Operators, Unique Operands, Total Operators and Total Operands as shown in Table X. There are different reference values for each dataset from NASA in [39] called P1,…, P5 in this paper. Halstead used these direct measures to calculate indirect measures, for instance, volume of software can be used to indicate

complexity. The higher the volume of software, the higher its complexity.

TABLE X - VALUES OF COMPLEXITY OPERATOR AND OPERAND METRICS.

Ref	Value to UNIQUE		Value to TOTAL	
	Operator Count	Operand Count	Operator Count	Operand Count
F	25	0	125	70
I	10	33	26	21
M	P1 – 7 P2 – 12 P3 – 15 P4 - 18 P5 - 15	P1 – 7 P2 – 17 P3 – 19 P4 - 21 P5 - 20	P1 - 13 P2 - 42 P3 - 54 P4 - 53 P5 - 50	P1 - 8 P2 - 27 P3 - 36 P4 - 57 P5 - 34

7 Response For a Class (RFC), Coupling Between Object Classes (CBO), Fan-in, Afferent Coupling (AC) and Number of Function Calls (NFC)

Metrics shown in Table XI to Table XIII are related to method calling. Fan-in is known as afferent coupling and Fan-out is known as efferent coupling.

There are different values for the same metric in this case too. For instance, Table XI shows in its first line the maximum value is 2 and in line 6 the maximum value is 13.

TABLE XI - VALUES OF CBO AND METRICS.

Ref.	Value	Nature of Measure
A	2	Max
D	5	Max
J	5	Max
J	9	Max
O	5	Max
O	13	Max
R	5	Max

In Table XII, there are discrepancies among values. For instance, the RFC maximum value starts with 0 and ends with 222.

TABLE XII - VALUES OF RESPONSE FOR CLASS (RFC).

Ref.	Value	Nature of Measure
A	100	Max
D	100	Max
J	100	Max
J	40	Max
K	49	80% quantiles - relative threshold
O	100	Max
O	44	Max
R	100	Max
S	[50;100] and 222	Desirable and Max

TABLE XIII - VALUES OF FAN-IN, AFFERENT COUPLING (AC) AND NUMBER OF FUNCTION CALLS (NFC) METRICS.

Metric	Ref.	Value	Nature of Measure
FAN-IN	B	10, 22 and 56	70%, 80%, 90% percentiles
AC	L	1, [2;20] and >20	Intervals mean: Good, Regular and Bad
NFC- Number of Function Calls	R	5	Max

8. Number of Attributes, Methods and Parameters

Metrics shown in Table XV measure characteristic related to classes and methods like number of attributes and methods per class and the number of parameters in a method signature. Some variations are considered, such as whether modifiers are public or private . The maximum value of 0 for the public attributes measure was presented in paper H, due to suggested good practices for OO modeling. Values originating from good practices could be called theoretical recommendations. However, paper H considers 0 as good, but accepts up to 10 as the regular situation, when considering the distribution analysis of dozens of open source systems. Those values could be called practical recommendations.

TABLE XV - VALUES OF NUMBER OF ATTRIBUTES, NUMBER OF METHODS AND NUMBER OF PARAMETERS.

Metric	Ref.	Value	Nature of Measure
Number of Attributes	E	39	Invalid Threshold. Do not use.
	K	0.1	75th percentile (relative threshold)
Number of Public Attributes	L	0, [1;10], >10	Intervals mean: Good, Regular and Bad
	H	0	Max
	K	0.1	75th percentile (relative threshold)
Number of Methods (NM)	B	29, 42 and 73	70%, 80% 90% quantiles
	E	1	Invalid Threshold. Do not use
	R	20	Max
	K	16	80% quantiles (relative threshold)
Number of Public Methods	H	[5;10]	Min and Max Interval
	L	[0;10], [11;40], >40	Intervals mean: Good, Regular and Bad
Number of Parameters (NP)	B	10, 22, 56	70%, 80% 90% percentiles

These reference values have not been widely accepted for the following main reasons: a) The thresholds that have been found cannot distinguish 'good' from 'bad' values, they just present statistical results; b) the thresholds originate from studies of only one (or a few) application domains, geographical regions, or groups of companies, reducing the generalizability of results; c) There are important discrepancies among thresholds proposed in different scientific papers; d) The papers do not explain why a new threshold proposed is better (or worse) than older ones. They just show numbers and assert their new numbers as new suggested thresholds.

Article [20] seems to have reference values that are more reliable, considering the number of systems and domains, but it considers only Java systems.

Some reference values that were proposed omit explanations of how they were calculated or the reason for those values. Sometimes, it was stated that the values were established based on author's experience [13][19], suggesting for us to close our eyes and just trust. Therefore, there is a long distance to be walked in this journey to improve metrics and their use.

During this analysis, no evidence was found regarding whether different kinds of software (e.g., CPU bound vs. I/O bound) should have the same thresholds or reference values. A similar question arises regarding whether software employing particular frameworks, or generated by code-generation tools should be expected to have reference values consistent with software that does not employ such technologies. For example, such software might contain attributes and methods that are empty or not used.

In [33] the authors showed different values for five projects (Table X). There were significant differences among the values for certain metrics in different articles. So, the question arises

of whether it is even possible calculate a single threshold or reference value in many cases.

Furthermore, in articles [31] and [32] the authors show negative results for those validations. The first one [31] demonstrates that there was no *threshold effect* (sudden effect change at some threshold) in some metrics. The second one [32] demonstrates there is no empirical evidence for the "Goldilocks Conjecture" that there is a 'sweet spot' for a given metric. Even if these are valid conclusions, surely there must be negative effects at some extreme values. In other words, even if Goldilocks found that all the beds were comfortable, she likely still would not have wanted to sleep on a board or on quicksand.

This context, without reference values that are generic, brings to mind learning processes like machine learning, neural networks, fuzzy systems, and so on. Those methods, without generic values for training, have to be trained in each context and must have their application limited to that context only.

Considering the current scenario without generic thresholds and reference values, more effort needs to be applied to find reasonable values.

We found no articles presenting values of metrics that simultaneously take into consideration dimensions such as domain, architecture, language, size of system, size of developer teams, modeling approach, code generation technology, build system, or delivery system. We believe that reference values may be quite different depending on where systems are situated in the space defined by the above dimensions. Additionally, there were no papers comparing metrics in several independent systems or different versions of the same system. By independent systems, we are referring to systems produced by others than those that are collecting, calculating or validating thresholds or reference values. In some cases, it was not clear whether the systems studied were independent of the evaluators.

Many reference values found should be used only with caution, because, for instance, either they do not have validation, or their measurement cannot be repeated, or they might be specific to a certain type of system, and hence not generic.

As mentioned earlier, it is also important to consider how a metric is interpreted or implemented and what impact this has on reference values. For instance, when the metric LOC is applied, it is necessary to define how blank lines, comments, and statements in more than one line will be counted.

There are also likely to be inherent differences in reference values for different programming languages. For example, some object-oriented languages might intrinsically need different numbers of classes, or different depths of inheritance due to such features as inner classes and multiple inheritance. Other feature differences could lead to different numbers of attributes, methods, and so on.

Thus, we recommend that a Metrics Research Protocol should be developed. This would promote consistent research and enable the exchange of 'big data' in this field among many researchers. It would be similar to what has happened in biology (e.g., genomics), particle physics, and so on.

There are some metrics for which there likely should be no natural limit and for which a more complex system would always have higher values. For instance, this would apply to LOC, number of classes, number of methods, and number of attributes in a system. For such metrics, the reference values should suggest averages or medians per unit, where the unit might be the class.

The situation described in this paper indicates that the software engineering community must conduct considerable additional research if it wants to be considered a true branch of engineering.

IV. Conclusion and Future Work

The main objective of this study was to conduct a survey of software metrics that have reference values or thresholds associated with them. For this, a systematic literature review was performed to identify, interpret, and evaluate the relevant scientific articles available.

During the conduct of the SLR, 6654 scientific articles were identified after searching of IEEE Xplore, EiCompendex, Elsevier Science Direct, Scopus and the ACM Library. The primary selection obtained 242 papers, based on scanning titles and keywords. With further refinement, 193 papers were classified irrelevant, 30 were classified repeated, and none were classified incomplete. Finally, the SLR resulted in 19 articles read and analyzed completely.

The original questions that motivated this SRL were: a) What software metrics have reference values or ranges assigned to them? b) What values or ranges were identified in the literature? Both of these questions were answered, and details were discussed throughout this paper. However, our analysis showed that the values are not yet generic enough or sufficiently validated to be useful.

The major contributions of this work are: i) the identification of a set of measures that have reference values ii) the summary of measures, values, systems evaluated, domains and languages involved, and technical validation, of these iii) critical evaluation of 19 articles.

The main conclusions of this paper are: i) There are conflicts among the most reference values; ii) There are several non-reproducible research papers in the field; iii) There are reference values based on weak or absent validation; iv) The selected reference values are for the most part not generalizable; v) There is little comparison between reference values and discussion of how one value is better than another; vi) The thresholds found cannot be used to distinguish 'good' from 'bad' values, they mainly represent statistical results; and vii) The scientific community should establish a protocol to determine what authors should consider minimum information and procedures that a paper must have when they study and purpose thresholds and reference values for software metrics.

Thus, the values presented in papers should not be trusted. The lack of reference values for software metrics persists. Additional investigation involving other articles not covered in this SLR and new statistical analysis involving multiple software systems must be conducted. Considering the discrepancies among values presented in this paper, we assert that the issue involving metrics and their thresholds and reference values is completely open and deserves more effort.

The possible threats to validity of this study include the limitations of search engines of the digital libraries, and lack of retrieval due to insufficient detail in the title, abstract or keywords of the papers. Other valid articles without keywords

 ISBN: 978-1-61208-498-5

in the titles or provided keywords might not have been found. The authors will conduct a new SLR involving the name of each metric and also conduct a statistical analysis of more than 100 open source software projects.

Software metrics have played a key role in organizations. Even though there has been a growth of research related to software metrics and thresholds in the last few years, this issue still needs further research and publications that provide support to software engineers.

Various actions should be taken as future work:

i) Perform a backward and foward SLR considering the set of articles discussed in this paper as the starting point. This might uncover important information that may have been abandoned over time, as well as complementary data about thresholds and methods.

ii) Perform a comparative analysis in order to identify discrepancies among thresholds selected in the literature, considering software both within various domains and across domains.

iii) Study and conduct research to establish thresholds for metrics of interest, creating quality protocols useful as for reference.

iv) Evaluate evolution of measures between different versions of the same software, of the same domain and different domains in order to get average values and uncover discrepancies.

v) Compare different metrics tools in order to look for discrepancies in the same metrics applied in the same projects, to understand why they produce different values, and to enable creation of warnings and advice about their use.

vi) Develop and propose a protocol to facilitate research into reference values for various metrics and software types, and for specific software instances to assure sharing of data and replicability.

vii) Model thresholds as an n-dimensional problem considering different domains, sizes, languages, paradigms, kinds of system and so on.

viii) Develop a comparative analysis about correlation among similar metrics in order to identify distinct behaviors even though those metrics assess the same software attribute (characteristic).

The software engineering research groups from UFLA and UOttawa continue their work in advancing all these proposals. In particular, we are extending the Umple technology [47, 48] to compute metrics for state machines and networks of associations embedded in code, and will develop systematic reference values for these metrics.

Finally, the results obtained in this SLR served to understand the state of the art and serve to guide subsequent studies related software metrics and thresholds.

ACKNOWLEDGMENT

The authors thank the CNPq / Brazil for financial support and the Research Groups on Software Engineering, Federal University of Lavras (PQES / UFLA) and the University of Ottawa is research environment.

REFERENCES

[1] F. B. E. Abreu and R. Carapuça, “Object-Oriented Software Engineering: Measuring”, in *Proc. of 4th Int. Conf. on Software Quality*. Milwaukee: American Society for Quality, pp. 1-8, 1994.

[2] A. J. Albrecht, “Measuring Application Development Productivity”, in *Proc. of first IBM Application Development Symposium*, New York: IBM, pp. 83-92, 1979.

[3] J. Bieman and B. Kang, “Cohesion and Reuse in an Object-Oriented System”, in *Proc. of Symposium on Software Reusability (SSR'95)*, New York: ACM, pp. 259-262, 1995.

[4] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design", *IEEE Transactions on Software Engineering*, v. 20, n. 6, pp. 476-493, 1994.

[5] J. A. Dallal and L. C. Briand, “A Precise Method-Method Interaction-Based Cohesion Metric for Object-Oriented Classes”, *ACM Transactions on Software Engineering and Methodology*, v. 21, n. 2, pp. 1-34, 2012.

[6] G. Gui and P. D. Scott, "New Coupling and Cohesion Metrics for Evaluation of Software Component Reusability", *In Proc. of 9th Int. Conf. for Young Computer Scientists* (ICYCS’ 2008), Hunan: IEEE, pp.1181-1186, 2008.

[7] M. Halstead, *Elements of software science*. New York: Elsevier, 1977.

[8] S. Henry and D. Kafura, "Software Structure Metrics Based on Information Flow", *IEEE Transactions on Software Engineering*, v. -7, n. 5, pp. 510-518, 1981.

[9] M. Lorenz and J. Kidd, *Object-oriented software metrics*. Englewood Cliffs, NJ: PTR Prentice Hall, 1994.

[10] T. McCabe, "A Complexity Measure", *IEEE Transactions on Software Engineering*, v. -2, n. 4, pp. 308-320, 1976.

[11] H. Washizaki, H. Yamamoto, and Y. Fukazawa, “A Metrics Suite for Measuring Reusability of Software Components”, in *Proc. of 9th Int. Symposium on Software Metrics*, Sydney: IEEE Computer Society, pp. 211-223, 2003.

[12] CCCC - Cccc.sourceforge.net, "Software Metrics Investigation", 2015. [Online]. Available: http://cccc.sourceforge.net/. [accessed: 13-July-2016].

[13] McCabe IQ. “MaCabe software”. Available: http://www.mccabe.com/pdf/McCabe%20IQ%20Metrics.pdf. [accessed: 13-July-2016].

[14] A. Terceiro et al, “Analizo: An Extensible Multi-Language Source Code Analysis and Visualization Toolkit”, in *Proc. of the Brazilian Conf. on Software 2010*, Salvador: SBC, pp. 1-6, 2010.

[15] N. Fenton and M. Neil, "Software metrics: successes, failures and new directions", *Journal of Systems and Software*, v. 47, n. 2-3, pp. 149-157, 1999.

[16] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems.* New York: Springer, 2006.

[17] E. Tempero, “On Measuring Java Software”, in *Proc. of 31th Australasian Computer Science Conference*, Wollongong: CRPIT, pp. 7-7, 2008.

[18] T. L. Alves, C. Ypma, and J. Visser, “Deriving Metric Thresholds from Benchmark Data”, in *Proc. of the 10th IEEE Int. Conf. on Software Maintenance*, Timisoara: IEEE Computer Society, pp. 1-10, 2010.

[19] J. C. Coppick and T. J. Cheatham, “Software Metrics for Object-Oriented Systems”, in *Proc. of 20th ACM Computer Science Conference*, New York: ACM, pp. 317-322, 1992.

[20] K. Ferreira, M. Bigonha, R. Bigonha, L. Mendes, and H. Almeida, "Identifying thresholds for object-oriented software metrics", *Journal of Systems and Software*, v. 85, n. 2, pp. 244-257, 2011.

[21] L. H. Rosenberg, R. Stapko, and A. Gallo, “Risk-Based Object Oriented Testing”, in *Proc. of 24th Annual Software Engineering Workshop*, Greenbelt: NASA, 1 CD-ROM, 1999.

[22] R. Shatnawi, W. Li, J. Swain, and T. Newman, "Finding software metrics threshold values using ROC curves", *Journal of Software Maintenance and Evolution: Research and Practice*, v. 22, n. 1, pp. 1-16, 2010.

 ISBN: 978-1-61208-498-5

[23] V. A. French, “Establishing Software Metric Thresholds”, in *Proc. of 9th Int. Workshop on Software Measurement*, Mont-Tremblant, pp. 1-10, 1999.

[24] J. Biolchini et al, Systematic review in software engineering. Rio de Janeiro: UFRJ, Technical Reports RT - ES, 679/05 , 31 p., 2005.

[25] B. Kitchenham, Procedures for performing systematic reviews. Keele: Keele University, Technical Report TR/SE-0401; NICTA Technical Report, 0400011T.1, 33 p., 2004.

[26] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. Keele: EBSE Technical Report EBSE-2007-01, 57 p., 2007.

[27] T. Dyba, T. Dingsoyr, and G. K. Hanssen, “Applying Systematic Reviews to Diverse Study Types: An Experience Report”, in *Proc. of 1st Int. Symposium on Empirical Software Engineering and Measurement*, Washington: IEEE Computer Society, pp. 225-234, 2007.

[28] Jabref, "JabRef reference manager", 2015. [Online]. Available: http://www.jabref.org. [accessed: 13-July-2016].

[29] O. Alan and C. Catal, “An Outlier Detection Algorithm Based on Object-Oriented Metrics Thresholds”, in *Proc. of the 24th Int. Symposium on Computer and Information Sciences*, Guzelyurt: IEEE Computer Society, pp. 567-570, 2009.

[30] T. L. Alves, J. P. Correia, and J. Visser, “Benchmark-Based Aggregation of Metrics to Ratings”, in *Proc. of the 21th Int. Workshop on Software Measurement; Int. Conf. on Software Process and Product Measurement*, Nara: IEEE Computer Society, pp. 20-29, 2011.

[31] S. Benlarbi, K. El Emam, N. Goel, and S. Rai, “Thresholds for Object-Oriented Measures”, in *Proc. of 11th Int. Symposium on Software Reliability Engineering*, San Jose: IEEE Computer Society, pp. 24-38, 2000.

[32] K. El Emam et al., "The optimal class size for object-oriented software", *IEEE Transactions on Software Engineering*, v. 28, n. 5, pp. 494-509, 2002.

[33] C. Catal, U. Sevim, and B. Diri, “Clustering and Metrics Thresholds Based Software Fault Prediction of Unlabeled Program Modules”, in *Proc. of 6th Int. Conf. on Information Technology New Generations*, Las Vegas: IEEE Computer Society, pp. 199-204, 2009.

[34] H. Washizaki, H. Yamamoto, and Y. Fukazawa, “A Metrics Suite for Measuring Reusability of Software Components”, in *Proc. of 9th Int. Symposium on Software Metrics*, Sydney: IEEE Computer Society, pp. 211-223, 2003.

[35] R. Ramler, K. Wolfmaier, and T. Natschlager, “Observing Distributions in Size Metrics: Experience From Analyzing Large Software Systems”, in *Proc. of 31th Annual Int. Computer Software and Applications Conference*, Beijing: IEEE Computer Society, pp. 299-304, 2007.

[36] Schneidewind, N. F. “Software Metrics Model for Quality Control”, in *Proc. of 4th Int. Software Metrics Symposium*, Albuquerque: IEEE Computer Society, pp. 127-136, 1997.

[37] R. Shatnawi, "A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems", *IEEE Transactions on Software Engineering*, v. 36, n. 2, pp. 216-225, 2010.

[38] P. Oliveira, M. T. Valente, and F. L. Paim, "Extracting Relative Thresholds for Source Code Metrics", *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*, pp.254-263, 2014.

[39] C. Catal, O. Alan, and K. Balkan, "Class noise detection based on software metrics and ROC curves", *Information Sciences*, vol. 181, no. 21, pp. 4867-4877, 2011.

[40] J. Al Dallal, "Improving the applicability of object-oriented class cohesion metrics", *Information and Software Technology*, vol. 53, no. 9, pp. 914-928, 2011.

[41] G. Succi, W. Pedrycz, S. Djokic, P. Zuliani, and B. Russo, "An Empirical Exploration of the Distributions of the Chidamber and Kemerer Object-Oriented Metrics Suite", *Empirical Software Engineering*, v. 10, n. 1, pp. 81-104, 2005.

[42] S. Herbold, J. Grabowski, and S. Waack, "Calculation and optimization of thresholds for sets of software metrics", *Empirical Software Engineering*, v. 16, n. 6, pp. 812-841, 2011.

[43] T. R. G. Nair and R. Selvarani, “Estimation of Software Reusability: An Engineering Approach”. ACM SIGSOFT Software Engineering Notes, New York, v. 35, n. 1, p. 1-6, 2010.

[44] K. Wolfmaier and R. Ramler, “Common Findings and Lessons Learned from Software Architecture and Design Analysis”, in *Proc. of 11th IEEE Int. Software Metrics Symposium*, Como: IEEE Computer Society, pp. 1-8, 2005.

[45] L. H. Rosenberg, “Applying and Interpreting Object Oriented Metrics”, in *Proc. of 10th Software Technology Conference*, Utah, pp. 1-18, 1998.

[46] T. Copeland, *PMD applied*. Alexandria, Va.: Centennial Books, 2005.

[47] T. C. Lethbridge, A. Forward, and O. Badreddin, “Umplification: Refactoring to Incrementally Add Abstraction to a Program”, in *Conference on Reverse Engineering*, Boston, pp. 220-224, 2010.

[48] Cruise Group, "Umple: Merging Modeling with Programming", 2016. [Online]. Available: http://www.umple.org. [accessed: 13-July-2016].

[49] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

 ISBN: 978-1-61208-498-5