

Can Software Metrics Be Unified?

Yusuf U. Mshelia(✉) and Simon T. Apeh

Department of Computer Engineering, Faculty of Engineering,
University of Benin, PMB 1154, Benin-City, Nigeria
mshelia@ieee.org, apeh@uniben.edu

Abstract. Software metrics produce discrete values as output from input source code as an indication of software quality. Many software have automated software metrics with different tools that are producing varying result. Measurement of software metrics cannot be said to be precise, repeatable and reproducible. This is consequent of varying definitions, design and implementation of the same software metrics, varying result, assessment and analysis of the same metrics from relative and personalized approaches to varying benchmarks, non-uniform definition of implementation contexts, software measurement terminologies and lack of standard reference and calibration with respect to the measure of "level of confidence" in software measurement. Several studies have proposed the unification of software metrics without necessarily looking at the underlying causes of these widely observed inconsistencies across existing metrics and their automated tools. This work identified pitfalls ways to minimize variances in the implementation of software measurements across contexts. From this stage of an ongoing research, we are determining the possibility of objectively unifying software metrics by closing the gap in observed sources of expressed variance and adoption of metrological approaches to software measurement.

Keywords: Metrics variance · Software metrics · Metrology

1 Introduction

Software metrics as a tool used in software quality assessment has gained increasing importance within its different areas of application. Metrics in software measurement has generally drawn its definition within a specific context-area of application. This forms a significant constraint in the inclusive application of software metrics from a globally-defined standard point of view. Software metrics from source code are generally used in fault and software quality predictions.

Fenton and Neil [1] reported the existence of thousands of software metrics that supposedly measures some form of metric, attribute, artefact or quality in software. In addition, more than 5000 software metrics are reported [2] in classifying metrics for assessing for maintainability of object-oriented systems. These numerous software metrics are identified to produce several inconsistences. These inconsistencies from the creation, design, implementation and automation of software metrics have introduced the following concerns:

1. Varying definitions, design and implementation of the same software metrics founded on different theories [3].

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11622, pp. 329–339, 2019.
https://doi.org/10.1007/978-3-030-24305-0_25

2. Varying result, assessment and analysis of the same metrics founded on relative and personalized approaches to benchmarks and interpretation [4].
3. Non-uniform definition of implementation contexts, terminologies and metrics.
4. Measurement of metrics cannot be said to be precise, repeatable and reproducible comprehensively in a different context from which is was originally designed [5].
5. Lack of standard reference and calibration with respect to the measure of "level of confidence" [5].

As an approach to measurement of software artifacts, relative definition of metrics, application and interpretation have been the trend in the definition of software metrics. Unlike the field that studies the science and technology of measurement - metrology, software metrics suffers from insufficient approaches that checks measures of software metrics with the required rigor and standard. Other fields have benefited from metrological approaches to measurement, especially in the measurement of physical objects in physics, chemistry, manufacturing, etc. Software however, is inundated with having to measure intangible non-physical objects without a standard etalon.

As a key to delivering a uniform standard in the definitions, methods, processes and implementation, pitfalls must be identified and duly avoided with best practices and to minimize variances in the implementation of software measurements across contexts.

This work identified sources of variations as observed in literature and experiments and proposed some variance reduction techniques. It further clarifies the systems of measurement in software metrics and metrology. This research paper presents observed challenges in software measurement that has contributed to varying values of the same name metric and ways to achieve uniform standard in software measurement.

The next section presents sources of variance as expressed in literature and observed from experience and literature. Section 3 discusses techniques in reducing variations in metrics results. Software metrics system of measurement is compared to measurement as in metrology in Sect. 4. Finally, from the understanding of the previous sections, we discussed the connections with respect to the contribution of metrics variance in unifying the software metrics in Sect. 5.

2 Sources of Expressed Variance in Software Metrics

2.1 Literature

As generally seen in literature, informed basis for specific work of unifying software metrics at different levels are inconsistency, disparity and non-uniformity in software measurement processes, models, systems, techniques and tools. The need for a consistent terminology and nomenclature of software measures was the focus of Rana et al. [6], Abran et al. [7] and García et al. [8] and an effort towards catalog format for software metrics Bouwers et al. [9]. A metrics collection tool (MECOT) was designed as a class-based metrics collector for accumulating different metrics into a repository [10]. Furthermore, other works have contributed to unifying methods and frameworks in implementing software metrics [11–21]. Furthermore, recommendations in standardizing software metrics within the framework of metrology have been proposed. Of fundamental significance in this area are works by Abran [22] and Flater et al. [5].

As an approach to unifying software metrics, an approach in establishing suite of metrics are also proposed as in the cognitive complexity metrics of Object Oriented Programs [23]. In identifying bugs in files, a work by Huo and Li [24] proposed a deep model called LS-CNN (Long Short-term memory based on Convolutional Neural Network) to enhance the unified features for bug localization by exploiting the sequential nature of source code, where a particular network structure is designed to combine CNN (Convolutional Neural Network) and LSTM (Long Short-Term Memory) that captures sequential semantics as well as structural information from source code. Another that outlined perceptions and challenges in defect prediction is by Wan et al. [25]. Code smells with respect to bad programming practice are reported as a contributor metric quality variance [26, 27]. The reporting of bugs was described to lead to high irregular patterns as the repository size is increasing with enormous rate, resulting in uncertainty and irregularities [28, 29]. Irrelevant features exist in source-codes that has motivated the work on the framework that validates the source code metrics and identify a suitable set of source code metrics with the aim to reduce irrelevant features and improve the performance of the fault prediction model.

The results of software metrics as expressed in the study of software metrics has posed challenges that largely varies in output, application and context. We define variance in this context as the difference in the way the implementations of metrics have contributed to the independent definitions, results applications and interpretations. Consequently, the following are causes to some of these differences:

2.2 Correlation Analysis of Software Metrics Suites

The analysis of Project L^3 [30], established that the relationship in the output values computed by the tools vary between a positively strong and negative correlation to negatively strong and weak correlation to perfectly positive and negative correlations. In addition, there exists no correlation at all to some degree for all the project classes for which results were attempted to be correlated.

A reason identified for such a wide gap in the degree of correlation across the metrics tools used in the experiment is that not all the tools analyzed the same number of metrics. Example from strongly negative results and no relationship are sourced in the fact at the tools have a non-uniform number of variables the tools independently measure.

2.3 Definition of Counted Reference Point

Counting is the most prevalent technique used in the software quality measurement. In software metrics, nearly every quantity that is not actually a physical quantity is determined through counting [5]. Table 1 below enumerates the “base measures and their measurement units” from parts 2–4 of ISO 9126, a standard for the evaluation of software quality with instructive units. It therefore summarized that almost 3/4 of the measures are counts of something other than bytes, time, or money.

Table 1. Derived Categorization of ISO 9126 quality metrics by types of units of measurement from [5]

Std part	Total	Time	Money	Bytes	Other count	Weighted score
2.(External)	38	10		2	26	
3.(Internal)	34	2		1	31	
4.(In use)	10	4	1		4	1
Total	82	16	1	3	61	1

The measurement of counted metrics may have an ambiguous definition in the implementation of metric tools which ripples to relative or self-defined analysis and interpretation. Line of Code (LOC) metric for example is an ambiguous metrics in that what makes it precise is the reference and pointer to what it is counting. Because LOC can take the forms of Logical LOC, Physical LOC, Blank LOC, Comment LOC. The difference in all these is the reference of what is counts. The variations have different applications as commented LOC will not be necessary in understanding software size, however, it will go a long way in deciding the comprehension or understandability of the source code.

In particular instances of measurements by count within the source code, there is no reference as to if blank lines are part of the count of a physical line of code or not, since most code editors generate a serial count irrespective of what is coded.

2.4 Naming Metrics and Their Definitions

Despite all the efforts and new developments in research and international standardization during the last decade, software measurement is currently in the phase in which terminology, principles, and methods are still being defined, consolidated, and agreed. In particular, there is no consensus yet on the concepts and terminology used in this field. For instance, software measurement researchers and practitioners have not reached an agreement on the precise meaning of some terms commonly used, such as ‘measurement’, ‘measure’, ‘metric’, ‘measurable attribute’, etc. Even worse, inconsistencies between the different research measurement proposals often occur [8]. Rana et al. [6] also reported that one of the limitations to develop a generic software quality prediction approach is inconsistency found in naming software product measures. In some cases, the same measure has been given different labels whereas in other cases same label has been used for different measures and classified these two anomalies as Type I and Type II inconsistencies respectively.

As metrics are created, the naming structure does not provide for exclusivity to titles or names of the metrics. This causes the recreation of the same named metrics of a different theory or implementation targeting to measure different artefact. Of the 568 software metrics surveyed by de AG Saraiva et al. [2] for analyzing metrics, many were found with inconsistencies in the names of metrics because there were metrics with the same names but different meaning. There were also metrics with different names, however with similar meanings.

Furthermore, from the definition of fundamental forms within the framework of the counting source statement in [31], multiple other forms[1,2] of LOC have been identified to bear components of the already identified forms in addition to some added components or even entirely not built on executable lines of code but on the general content of source code.

2.5 Exclusions from Optimization Techniques

Optimization of source code is aimed at to eliminating dead code, avoiding recursion, avoiding floating-point operations to the possible extent, avoiding large library routines and removal of loop-invariant statements. However, the use of correct optimization level of the compiler provides different optimization levels from 0 to 3 as the highest which can be set based on the need.

As a result, the optimization of software metrics tools can choose to exclude some portions of the source code in count situations especially if a lot of code reuse is involved. The optimization that is abstracted from the programmer and handled by the compiler can produce different results based on the compiler need perception in assigning what levels of optimization is needed.

2.6 Implementing Algorithm and Optimization Techniques

Being that every computational system is a representation of an underlying theory of the algorithm. These theories may have some different intermediate representations of algorithms for achieving the same result. Therefore, the data structure of such algorithmic statements may take different transformations that within some contexts could take just a statement to accomplish whereas within another context some transformation and result will take multiple statements.

Therefore, if a created metrics is not utterly explicit in the definition of its theory and to its intermediate steps, the chance of implementing the same metrics may have or adopt different algorithms especially when the programmer has multiple approaches as an alternative. As a result, the different approaches will consequently yield different result especially in counts of logical statements.

Again, optimization either by automatic optimizers or programmers can change the structure of either the metrics analyzer or an optimized algorithm used in source code to be assessed.

2.7 Metrics Aggregation Techniques

Source-code metrics are usually defined on a micro-level of method, class, package, etc., and should therefore be aggregated in order to provide insights in the evolution at the macro-level (system) [32]. Popular aggregation techniques like the standard summary statistical measures as mean, median, or sum. Their main advantage is universality

[1] http://www.locmetrics.com/index.html.

[2] https://en.wikipedia.org/wiki/Source_lines_of_code.

(metrics-independence): whatever metrics are considered; the measures should be calculated in the same way. However, as the distribution of many interesting software metrics is skewed, the interpretation of such measures becomes unreliable [33].

In the same line, the results of metrics vary with respect to the aggregation used in the analysis of metrics results. From nature of the relation between various aggregation techniques between Gini, Theil, Atkinson, and Hoover indices (I), correlation between I_{Theil} and I_{Kolm}, as well as between $I_{Atkinson}$ and I_{Kolm} increases as the system size increases, while correlation between mean and I_{Kolm} fluctuates without clear relation to system size.

3 Variance Reduction Techniques

3.1 Homogeneity of Standard Names and Definitions

Resulting from lack of a common terminology and inconsistencies between the different standards, the existing situation may seriously risk the effectiveness and potential benefits of software standardization efforts. A consistent software measurement terminology may provide an important communication vehicle to companies when interoperating with others [8].

Currently, the International Organization for Standards (ISO) is updating the definition of software engineering terminologies as a foundation to upgrade software measurement terminologies. Furthermore, there is need for increased referential materials to terminologies, approaches and frameworks for measurement.

The International Vocabulary of Basic and General Terms in Metrology (called VIM) [34] provides general standard terminology in measurement science or metrology. However, because of the uniqueness of some specific fields of measurements, other some areas have customized the VIM definitions within the context of the specific area. For the purpose of a concise definition of measurement concepts that is peculiar to software metrics, there is also a corresponding need for a precise work as in the Eurachem Guide [35], a collection of terminology in analytical measurement.

3.2 Counts and Reference Points

As shown in Table 1, counting is the most prevalent technique used in the software quality measurement, there is need in precise and explicit pointer to what to measure and what not. In the definition of counts, there must be no ambiguity or else, it will be subject too relative and subjective opinions and applications. In software metrics, nearly every quantity that is not actually a physical quantity is determined through counting.

3.3 Metrics Optimization Buffering

Realizing that metrics algorithm implemented independent of external modification will to a degree be consistent across different platforms and within the same context. Hence, if an optimization buffer or avoidance is adopted, results can be uniform.

3.4 Symmetry with Metrology

Metrology, defined by International Bureau of Weights and Measures (IBWM) is the science of measurement, embracing both experimental and theoretical determinations at any level of uncertainty in any field of science and technology. Measurement as in software metrics are quite different from that of metrology. Since metrology has an established structure for instituting measurement standards, software metrics can benefit from the it like other areas of measurement in physics and chemistry.

4 Software Metrics System Vs. Metrology

Generally, measurements are used to make comparisons, to make decisions, or simply to ascertain facts about an object of measurement.

Measurement of software metrics are measures of quality of software – a non-physical, non-tangible property or attribute of the software. Causal relationships that are widely observed to exist between real properties of the objects of measurement and measured quantity values are expressed in the form of physical laws which are strongly supported by empirical evidence. If a set of analogous laws about nontrivial properties of software existed, software engineering could have "widely accepted measurements traceable to solid principles, rigorously defined quantities, and base and derived units".

Equally, the discovery of measurement laws might just depend on first making the right measurements of the right properties. It is idealized that the primary difference between software measurement and physical measurement is the maturity of the

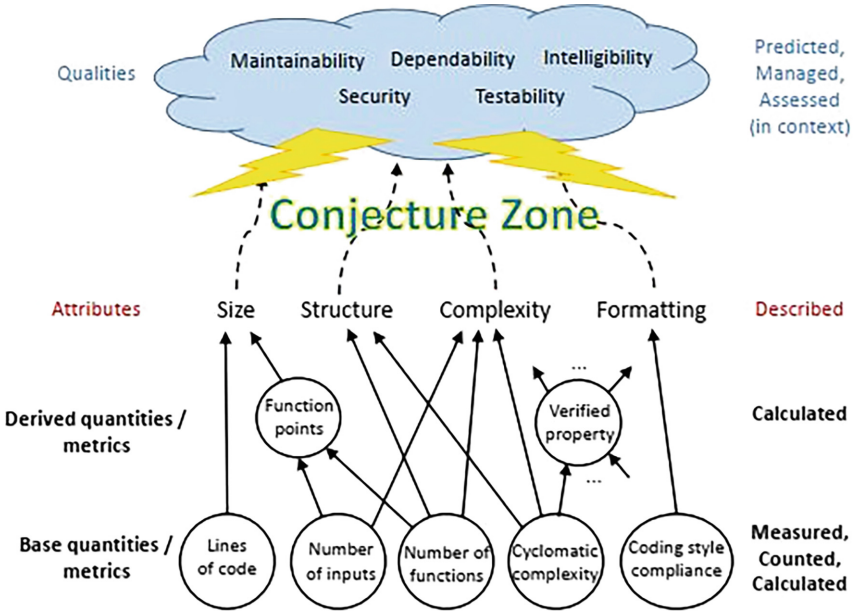

Fig. 1. Relating quality to quantity from Flater et al. [5]

measurement discipline. To the extent that software measurement is analogous to physical measurement, applying the established methods ought to shortcut the centuries-long process of refinement that evolved the capability to perform reliable physical measurements [5].

In further explanation of metrology framework for software measurement, vocabulary definitions in processes of that characterize *repeatability vs. reproducibility, measurand vs. realized quantity, instantaneous vs. cumulative quantity* were defined.

Figure 1 above shows the relationship between quality as an object of measurement as in software to quantity as in metrology. In order to realize this feat, specific areas in software metrics and metrology must be aligned. Areas of the differences that should be addressed to align to metrology in order to achieve the beauty of measurement standard, calibration, units and scaling are *traceability* and *measure of uncertainty.* This is largely lacking in the software metrics systems of quality assessment.

Other difference between the software metrics and metrology are outline in Table 2 below.

Table 2. Comparing software measurement to metrology

S/N	Software metrics systems	Metrology
1	Measures quality	Measures quantity
2	Objects of measurement are non-physical	Objects of measurement are physical
3	Undefined measure of uncertainty	Defined measure of uncertainty
4	No standard unit of measurement, calibration and scaling	Available standard unit of measurement, calibration and scaling
5	Metrics application are context	Measurement applications are global
6	Traceability not defined	Traceability is defined

5 Conclusion

First of all, the current system of measurement has been identified to bear numerous variations in definition, process implementation, and results that software metrics tools. The many contexts that has resulted to the numerous frameworks, many nomenclatures and applications on source code has diversified the process that could have otherwise brought uniformity to software measurement and quality predictions.

Whereas many other methods that for implementing software metrics are coming up, they can as well produce different results if some of these outlined sources are not catered for in their different automations and implementations.

To arrive at a homogeneous implementation of software measurement, the variations observed in the software metrics systems should be addressed. Of the outlined sources of variations, they can only minimize the occurrence of variations. Metrology can form a standard basis for defining quality and outlining measurement standards in software.

The researchers reported variations in software metrics systems as expressed in the literature, and other causes and proposed recommendations on how to reduce these variations. In addition to the reported sources of variations, if ambiguous definition of counted reference, naming variations and metrics algorithm optimization of analyzers re addressed, variations will largely be reduced. However, even when applied standard unit, calibration and scaling will not come from these. Consequently, software metrology is identified as an established field whose structure when applied to software metrics, will unifying formalized standards in software measurement.

What is next? Points of symmetry between software measurement and metrology within the constraint of traceability is the next phase of this work.

References

1. Fenton, N.E., Neil, M.: Software metrics: roadmap. In: Proceedings of the Conference on the Future of Software Engineering, pp. 357–370 (2000)
2. de AG Saraiva, J., De França, M.S., Soares, S.C., Fernando Filho, J., de Souza, R.M.: Classifying metrics for assessing object-oriented software maintainability: a family of metrics' catalogs. J. Syst. Softw. **103**, 85–101 (2015)
3. Mshelia, Y.U., Apeh, S.T., Edoghogho, O.: A comparative assessment of software metrics tools. In: 2017 International Conference on Computing Networking and Informatics (ICCNI), pp. 1–9 (2017)
4. Olaye, E.: Benchmarking software quality using traditional software metrics on web-based systems, Ph.D., Computer Engineering, University of Benin, Benin-city (2017)
5. Flater, D.W., et al.: A rational foundation for software metrology (2016)
6. Rana, Z.A., Awais, M.M., Shamail, S.: Nomenclature unification of software product measures. IET Softw. **5**, 83–102 (2011)
7. Abran, A., Sellami, A., Suryn, W.: Metrology, measurement and metrics in software engineering. In: Proceedings of the 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No. 03EX717), pp. 2–11 (2003)
8. García, F., et al.: Towards a consistent terminology for software measurement. Inf. Softw. Technol. **48**, 631–644 (2006)
9. Bouwers, E., van Deursen, A., Visser, J.: Towards a catalog format for software metrics. In: Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics, pp. 44–47 (2014)
10. Mshelia, Y.U.: MECOT: a software quality metrics collection tool. J. Syst. Integr. **10**, 21–35 (2019)
11. Alawneh, L., Debbabi, M., Hassaine, F., Jarraya, Y., Soeanu, A.: A unified approach for verification and validation of systems and software engineering models. In: 13th Annual IEEE International Symposium and Workshop on Engineering of Computer-Based Systems (ECBS 2006), pp. 10, 418 (2006)
12. Lochmann, K., Goeb, A.: A unifying model for software quality. In: Proceedings of the 8th International Workshop on Software Quality, pp. 3–10 (2011)
13. Mercier, S., Abran, A., Lavoie, M., Champagne, R.: Unified software method: towards a method of measurement of the necessary changes to software in maintenance (2006)
14. Bartolomei, T.T., Garcia, A., Sant'Anna, C., Figueiredo, E.: Towards a unified coupling framework for measuring aspect-oriented programs. In: Proceedings of the 3rd International Workshop on Software Quality Assurance, pp. 46–53 (2006)

15. Cavano, J.P., McCall, J.A.: A framework for the measurement of software quality. In: ACM SIGMETRICS Performance Evaluation Review, pp. 133–139 (1978)
16. Abran, A., Bourque, P., Dupuis, R.: The SWEBOK initiative and software measurement intentions. In: Proceedings of the 12th International Workshop on Software Measurement (IWSM 2002), pp. 168–178 (2002)
17. Jacquet, J.-P., Abran, A.: From software metrics to software measurement methods: a process model. In: Proceedings of IEEE International Symposium on Software Engineering Standards, pp. 128–135 (1997)
18. Kelemen, Z.D., Trienekens, J., Kusters, R., Balla, K.: A process based unification of process-oriented software quality approaches. In: 2009 Fourth IEEE International Conference on Global Software Engineering, pp. 285–288 (2009)
19. Akingbehin, K., Maxim, B.: A three-layer model for software engineering metrics. In: Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2006), pp. 17–20 (2006)
20. Symons, C., Abran, A., Ebert, C., Vogelezang, F.: Measurement of software size: advances made by the COSMIC community. In: 2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), pp. 75–86 (2016)
21. Dumke, R.R., Braungarten, R., Kunz, M., Schmietendorf, A., Wille, C.: Strategies and appropriateness of software measurement frameworks. In: Proceedings of the International Conference on Software Process and Product Measurement (MENSURA 2006), pp. 150–170 (2006)
22. Abran, A.: Software metrics need to mature into software metrology (recommendations). In: NIST Workshop on Advancing Measurements and Testing for Information Technology (IT), Maryland, USA (1998)
23. Misra, S., Adewumi, A., Fernandez-Sanz, L., Damasevicius, R.: A suite of object oriented cognitive complexity metrics. IEEE Access **6**, 8782–8796 (2018)
24. Huo, X., Li, M.: Enhancing the unified features to locate buggy files by exploiting the sequential nature of source code. In: IJCAI, pp. 1909–1915 (2017)
25. Wan, Z., Xia, X., Hassan, A.E., Lo, D., Yin, J., Yang, X.: Perceptions, expectations, and challenges in defect prediction. IEEE Trans. Softw. Eng. (2018)
26. Gupta, A., Suri, B., Kumar, V., Misra, S., Blažauskas, T., Damaševičius, R.: Software code smell prediction model using Shannon, Rényi and Tsallis Entropies. Entropy **20**, 372 (2018)
27. Gupta, A., Suri, B., Misra, S.: A systematic literature review: code bad smells in Java source code. In: Gervasi, O., et al. (eds.) ICCSA 2017. LNCS, vol. 10408, pp. 665–682. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62404-4_49
28. Kumari, M., Misra, A., Misra, S., Fernandez Sanz, L., Damasevicius, R., Singh, V.B.: Quantitative quality evaluation of software products by considering summary and comments entropy of a reported bug. Entropy **21**, 91 (2019)
29. Singh, V.B., Misra, S., Sharma, M.: Bug severity assessment in cross project context and identifying training candidates. J. Inf. Knowl. Manag. (JIKM) **16**, 1–30 (2017)
30. Mshelia, Y.U., Apeh, S.T., Olaye, E.: Parametric correlation and variance in constituent software metrics tools of project L^3. J. Data Anal. **11**, 125–137 (2016)
31. Park, R.E.: Software size measurement: a framework for counting source statements, Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst. (1992)
32. Vasilescu, B., Serebrenik, A., van den Brand, M.: You can't control the unfamiliar: a study on the relations between aggregation techniques for software metrics. In: 2011 27th IEEE International Conference on Software Maintenance (ICSM), pp. 313–322 (2011)

33. Vasa, R., Lumpe, M., Branch, P., Nierstrasz, O.: Comparative analysis of evolving software systems using the Gini coefficient. In: 2009 IEEE International Conference on Software Maintenance, pp. 179–188 (2009)
34. Vim, I.: International vocabulary of basic and general terms in metrology (VIM), International Organization, vol. 2004, pp. 9–14 (2004)
35. Barwick, V., Prichard, E.: Eurachem guide: terminology in analytical measurement–Introduction to VIM 3. Disponible à l'adresse www.eurachem.org (2011)