Lack of Consensus on Measurement in Software Engineering: Investigation of Related Issues

Pierre Bourque, Sibylle Wolff[*]*, Robert Dupuis*[**]*, Asma Sellami, Alain Abran*

École de Technologie Supérieure - ETS

1100 Notre-Dame Ouest,

Montréal, Canada H3C 1K3

[*] *SAP Labs Canada*

[**] *Université du Québec à Montréal*

pbourque@ele.etsmtl.ca, sibylle.wolff@sap.com, dupuis.robert@uqam.ca, asma.sellami.1@ens.etsmtl.ca, aabran@ele.etsmtl.ca

Abstract:

Even though measurement is considered an essential concept in recognized engineering disciplines, measures in software engineering are still far from being widely used. To figure out why software measurement has not yet gained enough peer recognition, this paper presents a set of issues that still have to be addressed adequately by the software measurement community. These issues were derived from the analysis of comments obtained during two Delphi studies and a Web-based survey conducted to identify and reach a consensus on the fundamental principles of the discipline within the international software engineering community. The paper also discusses the application of metrology concepts as a research direction to address some of the measurement issues identified.

Keywords

Metrology, Software measurement, Fundamental principles of software engineering

1 Introduction

In the IEEE collection of standards, software engineering is defined as:

"(1) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, i.e. the application of engineering to software. (2) The study of approaches as in (1)." [5]

It follows from this definition that the application of engineering to software requires a quantitative approach, and therefore measurement is mandatory from this perspective. Indeed, the use of measurements and quantitative models is known to be essential in engineering. For example, Kirby et al. [7] have stated, talking about the Egyptians, that “to place upright an obelisk of several hundred tons of weight is an engineering feat that requires nicety of calculations and special equipment even in modern times.”

In software engineering, however, there is a lack of general agreement on software measurement. This can be illustrated using the comments collected during a study conducted among world experts and experienced practitioners in software engineering to identify and develop (if feasible) a consensus on the fundamental principles of the discipline [3] :

- Some participants argued that measurement and quantitative models are fundamental to engineering and that without them there is no engineering per se. For them, measurement should always be applied and should be qualified if need be.
- Other participants believed that applying and using measurements and quantitative models in software engineering has too many caveats for it to be universally applicable or even universally desirable. In their opinion, measurements and quantitative models are not always applicable due to excessive cost or to the low level of maturity of software engineering.
- Some participants also believed that one can focus too strongly on measurement and quantitative models to the detriment of better judgment, and that measurement and quantitative models constitute only one form of input to the software engineering decision-making process.

These differences of opinion among such a group of top experts and seasoned practitioners in software engineering is troubling and requires further investigation. Therefore, this paper analyzes these differences of opinion by studying, in a systematic manner, the detailed set of measurement-related comments provided by these experts and experienced practitioners to support their opinions; these comments were collected during a study on the fundamental principles of software engineering [3], [10]. As an outcome of this analysis, questions on software measurement are formulated: these questions represent, individually or in groups, research issues that must addressed for measurement to gain wider recognition and usage in software engineering. This paper also discusses the application of metrology concepts as a research direction to address some of the measurement issues identified.

The set of comments collected during the study on the fundamental principles of software engineering is deemed worthy of study because it covers the entire discipline, comes from an international pool of very competent participants (e.g. members of IEEE Computer Society software engineering committees and experienced practitioners) and relates to the essence of the discipline. These comments can be negative, positive, represent an opinion, denounce two overlapping principles, etc.

Section 2 presents the steps of the study conducted to identify and develop a consensus on a list of fundamental principles of software engineering. Section 3 presents a list of reasons to measure in software engineering. These reasons are used as a conceptual framework to analyze the software measurement-related comments. An illustrative subset of the list of questions on measurement arising from this analysis is presented in section 4. The full set of questions is available in [10]. Section 5 discusses the interpretation and limitations of this analysis of software measurement-related comments and a discussion of the results. A summary is presented in section 6.

2 Fundamental Principles of Software Engineering

Figure 1 presents an overview of the various steps of the project conducted to identify and develop a consensus on a list of fundamental principles of software engineering; steps where comments were obtained are identified in grey.

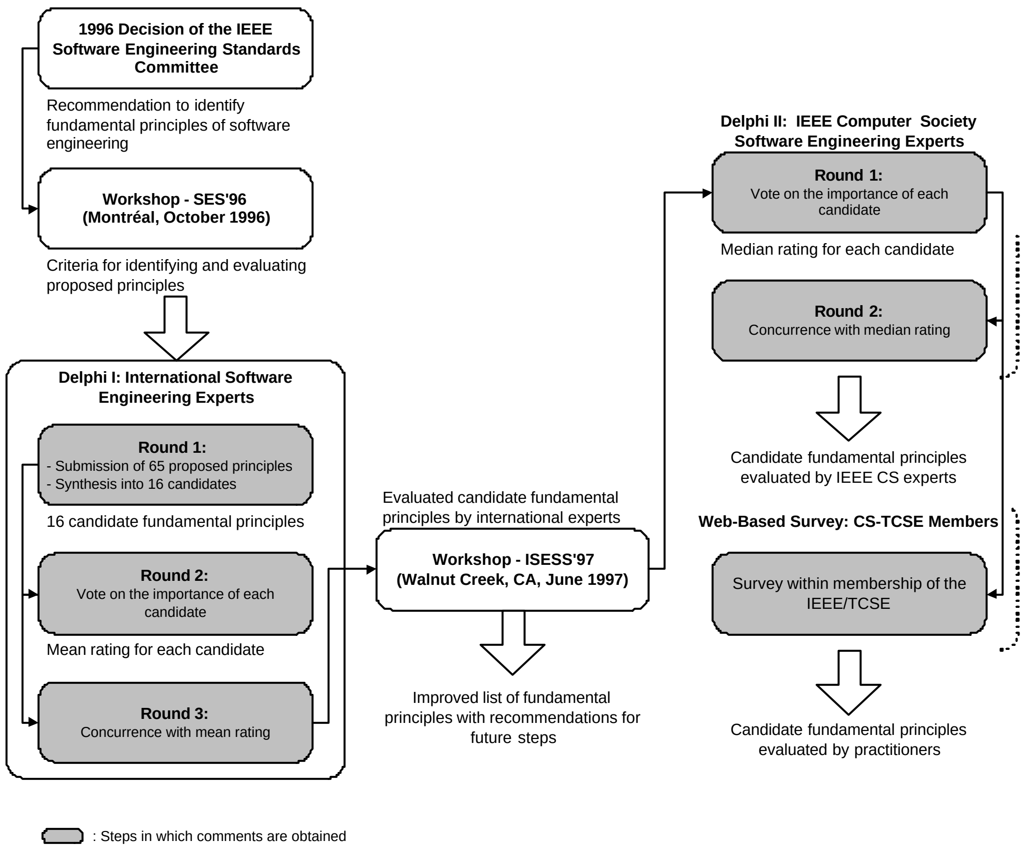
1996 Decision of the IEEE Software Engineering Standards Committee
Recommendation to identify fundamental principles of software engineering
Workshop - SES'96 (Montréal, October 1996)
Criteria for identifying and evaluating proposed principles
Delphi I: International Software Engineering Experts
Round 1: - Submission of 65 proposed principles - Synthesis into 16 candidates
16 candidate fundamental principles
Round 2: Vote on the importance of each candidate
Mean rating for each candidate
Round 3: Concurrence with mean rating
Evaluated candidate fundamental principles by international experts
Workshop - ISESS'97 (Walnut Creek, CA, June 1997)
Improved list of fundamental principles with recommendations for future steps
Delphi II: IEEE Computer Society Software Engineering Experts
Round 1: Vote on the importance of each candidate
Median rating for each candidate
Round 2: Concurrence with median rating
Candidate fundamental principles evaluated by IEEE CS experts
Web-Based Survey: CS-TCSE Members
Survey within membership of the IEEE/TCSE
Candidate fundamental principles evaluated by practitioners
: Steps in which comments are obtained

Figure 1: Project Steps of the Study on Fundamental Principles [3]

This project was prompted by a 1996 decision of the IEEE Software Engineering Standards Committee to begin efforts to identify a list of fundamental principles for software engineering. Such a list was viewed notably as an analysis framework for better organizing, explaining and validating software engineering standards. A first workshop was held at the Forum on Software Engineering Standards Issues of 1996 (SES'96) to establish what a fundamental principle is and which criteria it should conform to, in order to evaluate the proposed principles.

A Delphi study was then conducted in 1997 over the Internet among a group of software engineering experts, to identify a first list of candidate fundamental principles of software engineering. It was also recommended that a subsequent workshop be held to analyze the results of this Delphi study. The list of international experts participating in this first Delphi study can be found in [3] .

In Round 1 of this first Delphi study, the international experts were asked to submit proposals based on selected criteria. Each was asked to draw up a list of the five proposed principles they felt were most pertinent. They were also invited to add any amount of comments or explanations so that the Delphi study coordinators could better understand and explain their selection to the participants in Round 2. Thirteen international experts responded, which means that 65 proposed principles were obtained. Then, two Delphi study coordinators consolidated these 65 suggestions into a smaller number of principles, which met the criteria of the SES’96 workshop. This produced a list of 16 candidate fundamental principles. At this stage, the objective was to include the largest number of suggestions possible, while at the same time trying to reduce overlap among the candidates.

In Round 2, the experts were asked to rate each of the 16 candidate fundamental principles from Round 1 on a scale of 1 to 10. Participants were also asked to comment on their ratings, which most of them did. This also provided them with the opportunity to contest the way in which their initial five suggestions had been consolidated into the 16 candidate fundamental principles.

The goal of Round 3 of a Delphi study is to reinforce and confirm the ratings that emerge. Therefore, the experts were sent the mean scores of each candidate fundamental principle. They were then asked whether or not they concurred with the rating and to add more comments if need be.

A second workshop was held at the International Symposium on Software Engineering Standards of 1997 (ISESS’97) to eliminate or reformulate some of the principles and the criteria. This second workshop produced a list of improved criteria, as well as a more refined list of fundamental principles. These improvements were to be incorporated in the second Delphi study and in a subsequent Web-based survey.

A second Delphi study was therefore conducted in 1998 among 31 IEEE Computer Society “software engineering officials” in order to improve the principles. These officials were members of IEEE Computer Society editorial boards or of sanctioned committees related to software engineering. The list of participants in this second Delphi study can be found in [3]. From these two workshops and two Delphi studies, a list of fifteen candidate fundamental principles of software engineering has been compiled (See Table 1). Participants were also given the opportunity to provide comments during the two rounds of this second Delphi study.

Finally, a Web-based survey was conducted in 1999 among the membership of the Technical Council on Software Engineering (TCSE) of the IEEE Computer Society, with the cooperation of the IEEE Computer Society, to help verify the relevance of these candidate principles for practitioners and to help determine which of these fifteen candidate principles are indeed fundamental. A substantial number of comments were obtained through this Web-based survey as well. Demographics of the 574 participants who took part in this survey can also be found in [3].

- Apply and use quantitative measurements in decision-making
- Build with and for reuse
- Control complexity with multiple perspectives and multiple levels of abstraction
- Define software artifacts rigorously
- Establish a software process that provides flexibility
- Implement a disciplined approach and improve it continuously
- Invest in the understanding of the problem
- Manage quality throughout the life cycle as formally as possible
- Minimize software component interaction
- Produce software in a stepwise fashion
- Set quality objectives for each deliverable product
- Since change is inherent to software, plan for it and manage it
- Since tradeoffs are inherent to software engineering, make them explicit and document them
- To improve design, study previous solutions to similar problems
- Uncertainty is unavoidable in software engineering. Identify and manage it

Table 1: List of candidate fundamental principles (in alphabetical order) [3]

3 Comment Analysis Framework and Steps

This section presents the framework adopted to analyze the measurement-related comments and to structure the questions presented in the next section. The steps followed to analyze the comments are also presented in this section, as well as an example of the analysis of two software measurement-related comments.

Oman and Pfleeger [8] identify six key reasons for measuring in software engineering; these reasons were selected as the analysis framework of the software measurement-related comments:

- **Measuring for understanding:** Certain measurements allow a better understanding of the activities of software development and maintenance. It is possible, therefore, to understand the current situation by establishing baselines, thus enabling the formulation of objectives for future behaviour.

- **Measuring for experimentation:** Experimentation is necessary in software engineering, notably to improve software development methods, to better understand the effects of various technologies and to identify the areas requiring the most improvement. Measurement plays an important role in experimentation by enabling the creation, and testing, of hypotheses.

- **Measuring for project control:** Measurement enables project control by facilitating the evaluation of project status, and by predicting future events in the project.
- **Measuring for process improvement:** Measurement helps to improve the quality of software engineering processes, by better evaluating them and by understanding more fully the impact of adopted changes to these processes.
- **Measuring for product improvement:** Measurement provides insight into how software engineering processes, products, resources, methods and technologies are interconnected. For example, measurements can help answer questions regarding the effic iency of techniques and tools, the productivity of development activities and the quality of products, thereby enabling product improvement.
- **Measuring for prediction:** For new activities, it is necessary to predict required effort, as well as development costs and other factors. Measurement provides a baseline to predict these activities. Waiting until the end of project to measure cost and time attributes is clearly unacceptable.

Fenton and Pfleeger [4] identify a seventh reason for measuring: “measuring for evaluation.” This is somewhat similar to measurement for prediction, since attributes of entities are measured during the software development process, but by focusing on the past and the present rather than on the future.

- **Measuring for evaluation:** It is important to be able to understand what is occurring now, as well as what has happened in the past.

The methodology designed for the analysis of the software measurement-related comments is presented in Figure 2. The three inputs to the analysis are the set of comments from the two Delphi studies and the comments from the Web-based survey, as well as the measurement categories defined previously. The analysis itself includes three steps: the selection of comments relevant to measurement and the classification of these comments using the measurement categories, followed by a synthesis of these comments. Finally, the outcome of this analysis is a list of issues identified as underlying the lack of generalized consensus on measurement in software engineering. These issues are formulated as a set of questions relevant to measurement in software engineering.

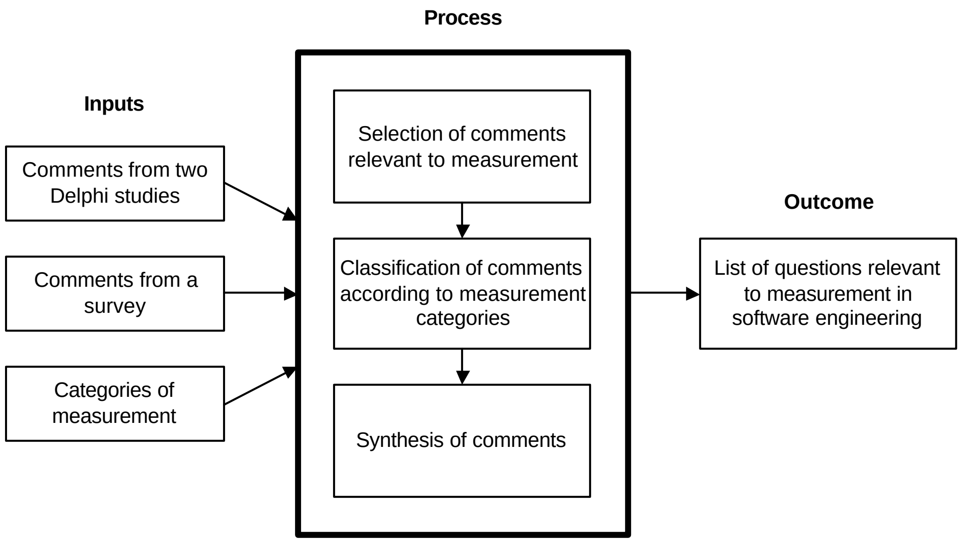
Process
Inputs
Comments from two Delphi studies
Comments from a survey
Categories of measurement
Selection of comments relevant to measurement
Classification of comments according to measurement categories
Synthesis of comments
Outcome
List of questions relevant to measurement in software engineering

Figure 2: Methodology for Analysis of Comments [10]

The first step, the selection of comments relevant to measurement delivered 85 comments from the full set of comments collected during the two Delphi studies and from the Web-based survey. These comments were then classified according to the various reasons for measuring in software engineering. It should be noted that the comments classified according to the measurement categories are related to "why" we measure in software engineering. However, some comments could not be classified based on these categories because they relate to measurement from a general viewpoint or to the "what and how" of measuring. Comments can also pertain to the utility itself of measurement in software engineering. An additional category referred to as "comments about measurement in general" was therefore added, and, of course, some comments are included in more than one category.

Subsequently, a synthesis of the comments related to measurement was completed. Comments were grouped together by issue within each category. Some of the categories have many issues, while others have only a few. The subtle differences between the comments were also identified. These subtleties were analyzed to verify whether they would generate new questions, or whether a single question would cover all subtleties. The last step consisted in producing the list of questions.

As an example, Table 2 presents the analysis of two comments that were categorized within the "measuring for process improvement" category. Three questions were produced from these comments.

ID.	Participant Comments	Derived Questions
'i'	This [measurement] is one of the only mechanisms we have by which we might hope to improve our processes to something resembling an engineering discipline.	• Can a discipline where measurement does not play an important role claim to be a legitimate engineering discipline? • Do software engineering educational programs emphasize sufficiently the importance of measurement in software engineering? • Does measurement constitute the only mechanism enabling process improvement?
'j'	Uncertainty is created by the lack of engineering and scientific discipline in the definition of the problem, or services to be provided by the system or product(s). To become an engineering discipline, the future software engineer must be taught the same engineering and scientific fundamentals concerning systems and products. This means clear quantification of the performance required by the process and results provided by the associated system or product(s).	

Table 2: Analysis of two comments within the "Measuring for Process Improvement" category

4 Questions about Measurement in Software Engineering

An illustrative subset of the list of questions is presented in this section. Within each category or reason for measuring, a summary of the analysis of the relevant comments followed by the derived questions are presented. Readers are referred to [10] for the full text of the comments, analysis and derived questions.

Measuring for understanding

The single comment identified within the category of "measuring for understanding" states that the need to measure is a sign that no one understands the problem, artifact or technology. This may indicate that either there is a belief that measuring does indeed help in understanding software engineering problems, artifacts and technologies or, perhaps the reverse, that it does not help at all. Questions derived from the analysis of this comment are the following:

- What is our level of understanding of the problems, artifacts and technologies of software engineering?
- To what extent can measurement help us better understand the problems, artifacts and technologies of software engineering?
- Which aspects of these software entities (e.g. problems, artifacts and technologies) is it important to measure?
- How much weight should be given to each of these aspects?

Measuring for evaluation

Some comments related to measuring for evaluation deal with the use of evaluation to minimize the uncertainty of projects in software engineering. Some of the questions derived from these comments are the following:

- Does software engineering have a higher level of uncertainty in comparison with other disciplines of engineering, or have these other engineering disciplines learned over time to better control their uncertainties?
- Could the use of measurement in software-related activities (such as feasibility studies, architecture definition, risk identification and mitigation, planning, and cost and schedule estimation) help decrease the level of uncertainty in software engineering projects?

Measuring for experimentation

No comment on measuring for experimentation was identified in the set of collected comments. This absence is in itself intriguing and might be related to experimentation not being present in software engineering to a sufficient degree.

Measuring for project control

One participant asserted that every project should plan and quantitatively specify its reliability strategies based on its reliability objective(s). Another participant asserted that every software project should set a reliability objective or objectives for its deliverable product and that it should be estimated at various points in the project and compared with the objective(s) set at the outset of the project. Questions derived from the comments in this category are the following:

- Is it really useful to use quantitative measures in project planning and control?
- Doesn't control necessarily imply measurement?
- Is quantitative measurement necessary for the control of all types of software engineering projects?

Measuring for process improvement

A participant suggested that the use of quantitative data should not only guide the management of the projects themselves, but should also enable senior management to gauge the progress of process improvement in their organization. Other participants asserted that the act of measuring the progress and the performance of software engineering processes is essential for implementing improvements needed for meeting industry demands. Derived questions are the following:

- Are measurements used for process improvement in software engineering valid?
- Are there any measurements that enable the control of progress in process improvement due to the usage of other measurements?

Measuring for product improvement

One participant asserted that software quality evaluation must be scientific and quantitative. However, results must also be useful and the costs should not exceed the advantages. The following question is derived:

- Must measuring for product improvement be done at all costs, or it is essential to calculate return on investment obtained from the use of these measurements?

Measuring for prediction

Some comments suggested that, in certain projects, it is important to be able to estimate in advance their cost and schedule because it is simply unacceptable to wait until the end of the project to know these values. It is also often necessary to know if a given software product can achieve the required quality objective, such as reliability. It is then important to obtain measurements to be able to predict what will occur if the project is carried out. Derived questions are the following:

- In the case of predictions, which are the most reliable: predicted values of measurements formulated at the beginning of the project or actual measurements taken during the project?
- Can the use of measurements taken from previous projects enable better predictions on future projects?

Measurement in general

Measurement plays an important role in engineering. This is why, according to some participants, if measurement is not used systematically in software engineering, the discipline cannot be recognized as an engineering discipline. However, another participant pointed out that, despite the importance of measurement, the practice of software engineering will always remain a combination of art and science. Derived questions are the following:

- Does measurement truly play a crucial role in software engineering?
- If measurement does not play a crucial role in software engineering, are the prospects for the discipline being recognized as a branch of engineering by the various official organizations automatically eliminated?

5 Interpretation, Limitations and Discussion

This analysis of the opinions of experts and experienced practitioners on the use of measurement and quantitative models in software engineering illustrates the lack of generalized consensus on the subject. This lack of consensus among experts and experienced practitioners is of course very puzzling, especially since:

- the IEEE definition of software engineering according to [5] explicitly requires the application of a quantifiable approach;
- the expression “software engineering” is widely used in research and in practice, and the extensive use of measurements and quantitative models is a given in recognized engineering disciplines.

To summarize the measurement-related comments:

- some of the participants asserted that measurement must be an integral and mandatory part of software engineering and without it there is no engineering per se;
- others were of the opinion that, while measurement should have an important role in software engineering, software measurement itself has too many limitations in its current state for it to be universally applicable or even universally desirable.

Though the authors of this paper believe that the set of collected comments is worthy of analysis and indicative of important unresolved issues regarding measurement in software engineering, limitations of this study must be highlighted. First, the two Delphi studies and the Web-based survey were not meant to clarify the role or the importance of measurement in software engineering, but focused instead on the fundamental principles of the field. Second, even though defined steps were followed to select and analyze the comments relevant to measurement, this remains a qualitative approach.

Further studies focusing directly on the role of measurement in software engineering are obviously required to verify, refine, delete and add questions to the list produced by this analysis.

In other disciplines of engineering, measurement methods, measuring instruments and techniques for measurement have long been established and must satisfy a set of metrology concepts as documented in the VIM [6]. In addition, quality criteria about measurement results must be known, such as accuracy, repeatability and reproducibility.

The issues highlighted in the set of questions documented in the previous section support the opinion that there is a lack of maturity of measurement in software engineering. This is not surprising since the majority of software measures proposed in the literature are not based on verifiable approaches, and therefore cannot be qualified as measurement methods [9]. Software engineering measures should be consistent with measurement concepts in other disciplines of engineering. Moreover, metrology concepts from the physical sciences are applied successfully in other scientific and engineering fields [1], [2].

The use of measurement and the interpretation of measured data in software engineering should demand considerable attentio n. The main objective in any measurement is to establish the values of attributes of some entities. These values should be meaningful and should be based on the foundations of adequate measurement methods and related measuring instruments based on traceable measurement references.

Indeed, analyzing the problem of measurement in software engineering represents a measurement design problem. What is needed is a more precise definition of software measurement, perhaps more than in any other engineering field. The approach taken so far for designing software measurements is not mature regarding metrology concepts and therefore is insufficient to support adequate experimentation in software engineering. At some point, measurement standards will be required to sup port software engineering as a *bona fide* engineering discipline.

6 Summary

This paper presents the results of a detailed and rigorous analysis of the software measurement-related comments collected during a study conducted to identify and reach a consensus on the fundamental principles of the discipline within the international software engineering community. To better grasp the underlying elements of the lack of consensus on measurement in software engineering and to help move it forward, this paper presented a subset of a list of questions arising from the analysis of the collected comments. The derived questions were structured around seven reasons for measuring in software engineering and an additional category of "measurement in general". These questions should be viewed individually or in groups as research issues to be addressed for measurement in software engineering to be more widely accepted and practiced. The application of metrology concepts to software measurement was presented as a research direction to solve some of the identified issues.

References

1. Abran, A., Software Metrics Need to Mature into Software Metrology. in *NIST Workshop on Advancing Measurements and Testing for Information Technology (IT),* (Gaithersburg (Maryland), 1998).
2. Abran, A., Sellami, A. and Suryn, W., Metrology, Measurement and Metrics in Software Engineering. in *Ninth International Software Metrics Symposium (METRICS'03),* (2003).
3. Bourque, P., Dupuis, R., Abran, A., Moore, J.W., Tripp, L.L. and Wolff, S. Fundamental Principles of Software Engineering - A Journey. *Journal of Systems and Software*, *62* (1).
4. Fenton, N.E. and Pfleeger, S.L. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., 1997.
5. IEEE. Standard Glossary of Software Engineering Terminology, IEEE, 1990.
6. ISO. International Vocubulary of Basic and General Terms in Metrology, International Organization for Standardization - ISO, Geneva, 1993.
7. Kirby, R.-S., Withington, S., A.-B., D. and Kilgour, F.-G. *Engineering in History.* Dover Publications Inc., New York, 1990.
8. Oman, P. and Pfleeger, S.L. *Applying Software Metrics*. IEEE Computer Society Press, 1997.
9. Sellami, A. and Abran, A., The contribution of metrology concepts to understanding and clarifying a proposed framework for software measurement validation. in *13th International Workshop on Software Measurement,* (Montreal, 2003).
10. Wolff, S. La Place de la Mesure au Sein des Principes Fondamentaux du Génie Logiciel, Université du Québec à Montréal, 1999.