An Analysis of the McCabe Cyclomatic Complexity Number

Alain Abran	Miguel Lopez	Naji Habra
ETS- U. of Québec, Canada	Cetic, Belgium	University of Namur, Belgium
aabran@ele.etsmtl.ca	vp@cetic.be	nha@info.fundp.ac.be

Abstract

On basis of a framework defined to understand, structure, compare and analyze the different measurement approaches presented in software engineering literature, this paper develops a detailed analysis of the well-known McCabe Cyclomatic complexity number. It highlights in particular some misconceptions underlying this measurement approach and also points out the necessity to have well grounded definitions and models for the measurement methods the practitioners are applying in the software industry.

Keywords : McCabe, Cyclomatic number, Software measurement, measurement framework

1 Introduction

Some measurement methods used in software industry are still not well understood. Although these measurement methods are correctly applied by practitioners, there remain ambiguities within their design and corresponding interpretation as well as about their internal strengths as measurement concepts and related measurement instruments based on imprecise empirical representations.

Indeed, the definition or design of software measurement methods is a difficult task: concepts to be measured in software engineering lack enough precision and clarity and there is a lack of detailed consensus about characteristics of the concepts to be measured. So designing new software measures remains a challenge. Even for software measures published over 20 years ago, a number of measurement design-related issues are still open. For instance, the McCabe cyclomatic complexity number is widely used within software industry [HEN96]; however, even after close to 20 years, this "number" is not completely understood and is difficult to interpret.

This paper explores the McCabe number based on its design itself instead of relying on the subsequent numerous interpretations. To do so, an analysis framework based on the software measurement method is used. This analysis framework consists of all the steps of a measurement process from the design or definition of the measure itself until the exploitation of the measurement results derived from the application of this measurement design to a particular context of measurement [HAB04].

This paper explores the different elements of the design of McCabe number based on the documentation provided by McCabe in the site [WAT95,96]. The remainder of the paper is organized in the following manner: Section 2 presents the analysis framework that will be used for exploring the design of a software number. Section 3 presents a brief overview of related work on software complexity measurement and Section 4, the McCabe cyclomatic complexity number, its origin in graph theory and its transposition into software measurement. In Section 5, the design of McCabe is then explored using the analysis framework presented in section 2. A discussion of findings is presented in Section 6, and a summary in Section 7.

2. Analysis Framework

The section presents a summary of the first step of an analysis framework of measurement. The whole framework (described in [HAB04]) starts with the definition of the measurement goal and ending with the exploitation of the measurement results. It is based on the process model of Abran and Jacquet [JAC97] integrating the validation framework of Kitchenham et al [KIT95], ideas from the validation approach of [LOP93] and the models of metrology concepts given by Sellami and Abran [SEL04].
This work aims at better understanding of cyclomatic complexity number proposed in [WAT96] and the framework parts related specifically to definition or design of the measurement method are applied to the analysis of this measurement method.

2.1 Measurement concepts and related terms

Before stating the different activities involved by the measurement method design, it is worth fixing a clear definition of the related measurement concepts.

Entity : An entity is a distinguishable entity on the empirical world to which a measurement process is to be applied.

Attribute : an entity is characterized by a set of attributes corresponding each to one observable characteristic of an entity type.

Entity type : an entity type is a set of entities having similarities (this is called also the *Empirical set)*, measurement being a way to compare entities, the underlying idea is to determine the entities one usually considers to be characterized through the same set of attributes and for which we try to design one measurement process. A set of entities of the same type is also called an empirical set.

Internal Attribute & External Attribute: Internal attributes can be observed statically, i.e., without any execution and thus with no connection with the environment factors, while external attribute are observable only during actual execution within actual environment.

Measurement of an attribute: a measurement of a given attribute of a given entity type is a mapping between the characteristic of an entity type and a corresponding numerical structure. This mapping is defined in a measurement method and leads to a numerical representation of the attribute under consideration

Measurement method[1]**:** Logical sequence of operations, described generically, used in the performance of measurements.

Measurement procedure[2]**:** Set of concrete operations, described specifically, used in the performance of particular measurements according to a given method.

Numerical structure: a numerical structure is defined as a set of distinguishable elements together with a collection of defined numerical relations (including ordering) on those elements.

Measurement result: the result of the measurement of a given attribute of a given entity is the numerical representation assigned to the attribute by the mapping prescribed by the measurement method. To measure a given entity attribute is thus to apply that mapping on the entity in order to get the corresponding numerical element.

Measurement process : the term measurement process is used for the whole process of measurement starting with the definition of the goal for the design of a measure, to its application in a context of measurement up to the exploitation of the measurement results in relationship with other information whether qualitative or quantitative.

2.2 Design of the measurement method

The major activities required for developing the design of a measurement method are described below:

1- *Defining the measurement objective:* that is selecting the objective of the measurement in terms of the characteristic to be measured and for which entity type.

2- *Characterizing the Empirical Structure:* that is defining the attribute in a clear and precise way so as to have a sufficient unambiguous characterization of the attribute. This requires to derive a model that represents the practitioner's knowledge of the attribute under consideration. A key difficulty is that while practitioners have an intuitive view of the attribute to be measured, such intuitive views are basically 'individual' views based on particular experience and knowledge, the totality of which is not necessarily shared, nor well described. These imprecise individual views do not usually lead to common models and numerical mapping thereby leading to rarely compatible and comparable numerical representation. In designing a measurement method, there is a necessity of establishing and documenting in detail clear 'conventions' both for the characterization of the empirical structure and for the mapping of an attribute to a numerical representation. Modeling the empirical world would be viewed as a two level modeling : a meta-modeling level and a modeling level.

The meta level corresponds to the choice of the modeling language to be used for describing the empirical world and the way the attribute is to be modeled in the language chosen (it includes also conceptual meta-modeling).

For example, to measure some design qualities of an object-oriented program (OO program), the meta model could be the one proposed by the OMG group in the Meta-Object Facility [MOF03], which proposes the concepts present in the object -oriented programming.

---

[1] From "International Vocabulary of Basic and General Terms in Metrology", International Organization for Standardization, Switzerland, 2nd edition, 1993, ISBN 92-67-01075-1

[2] From "International Vocabulary of Basic and General Terms in Metrology", International Organization for Standardization, Switzerland, 2nd edition, 1993, ISBN 92-67-01075-1

The second level is the characterization of the attribute to be measured which corresponds to the modeling of the attribute itself according to the rules defined in the previous part. It includes the axioms, the equations, the schemas, etc describing the attribute under consideration.

3- *Defining a numerical model* This activity consists in the definition of a numerical world to which the attribute characterization and meta-model is to be mapped. It is twofold. On the one hand, it includes the definition of the numerical structure to which the empirical set is to be mapped and this is related to the concept of scale type [KIT95], [FEN96]. On the other hand, it includes the definition of correspondence rules to be used to map empirical elements into numerical ones.

The remainder of the process involves the application of the measurement method, the analysis of the measurement results and their exploitation. These activities will not be developed further here, this paper focus on the measurement design activities.

3. Complexity Measurement - Related work

Prior to investigating the explicit and implicit definitions behind the McCabe 'cyclomatic complexity number', it is interesting to survey how complexity has been defined. Therefore, this section explores briefly some meanings of complexity in a general sense and in a software sense.

In the general sense as documented in the Webster, complexity is *the quality or state of being complex.* Furthermore, complex means *composed of two or more parts* [WEB04].

In Software Engineering, a large number of authors have investigated this generic concept of complexity and provided a variety of definitions corresponding *to multiple views of complexity* [WIT97]. This section deals with a few of these views of complexity.

In [IEEE], complexity is defined as *the degree to which a system or component has a design or implementation that is difficult to understand and verify*. So, on the one hand, the definition argues that the complexity is a property of the design implementation, *i.e.* source code or design. On the other hand, this definition of complexity contains also a relationship to on the effort needed to understand and verify the design implementation. This means that two different entities are involved in the definition, process (effort) and product (design or source code) which, of course, can lead to a number of interpretations.

In [EVA87], complexity is *the degree of complication of a system or system component, determined by such factors as the number and intricacy of interfaces, the number and intricacy of conditional branches, the degree of nesting, and the types of data structures*. This definition refers to the structure of the system (entity type = *product)* and defines some of its characteristics and some elements of its meta-model. This kind of view focuses on a single concept to be measured, and, therefore facilitates its understanding.

In [WIT97], the characterization and meta-model of complexity has a broader scope and includes a typology for the complexity concept, including computational, psychological and representational complexity. Computational complexity is based on the notion proposed by [HEND96, WIT97], and is defined in terms of hardware resources (processors cycles, memory, disk space) required to execute the software. Psychological complexity also proposed by [HEND96, WIT97] refers to the complexity problem solved by the software, the structural complexity which includes characteristics of the software (size, cohesion, coupling,...), and programmer complexity that includes his knowledge and experience of the problem and solution domain.

Although the previous list is not exhaustive, it allows to capture the lack of consensus on both the definition of software complexity and about a measurement method design of this software attribute. This lack of consensus on the concept of software complexity (and related entity attribute and characterization) makes it therefore quite challenging to researchers attempting to design measures of software complexity. For each view of complexity a related measurement design can be relevant and, it is then important to clearly define the attribute to be measured and the entity to which it is related to in order to correctly design related measures.

4. McCabe Cyclomatic Complexity Number

4.1 Definition

McCabe did not provide a clear definition (characterization and meta-model) of the complexity attribute: this does not facilitate the understanding of this measure and it is challenging to analyze McCabe interpretation of software complexity. Similarly while defining a numerical assignment rule, McCabe did not provide the properties of the numerical representation derived from the application of his documented algorithm. Both topics are discussed next.

4.2 Cyclomatic number in Graph Theory

McCabe's work is based on his analysis of some measurement concepts in the graph theory and on his transposition of these concepts to the domain of software measurement. Understanding of related measurement concepts, and in particular of the cyclomatic number in graph theory, is therefore important to understand the input to the design of the number proposed by McCabe.

Of particular relevance is the following set of important definitions:

Simple Graph: A simple graph is a (usually finite) set of vertices V (or nodes) and set of unordered pairs of distinct elements of V called edges. A simple graph, G, could be defined as a pair of sets (V, E), where V is a finite non empty set of vertices and E is a set of pairs of vertices, called edges. It is often represented by G=(V,E) [BER01].

Chain: In a simple graph G=(X,A), a chain *c* is a finite sequence of vertices $x_0,x_1,...,x_m$ such that for all i with $0 = i = m$, x_i,x_{i+1} is an element of A. A chain is represented by $c=[x_0,x_1,...,x_m]$. The length of the path *c* is the integer *m*, and it is represented by l=m(c) [BER01].

Formatted: Not Highlight

Connected Graph: A graph G is connected if for all x, for all y [vertices], it exists a chain connecting x and y [BER01]. A graph that is not connected can be divided into connected components.

Cycle: A cycle graph is a path that begins and ends with the same vertex.

Simple Cycle: A simple cycle is a cycle which has a length at least of 3 and in which the beginning vertex only appears once more, as the ending vertex, and the other vertices appear only once.

Directed Graph: A directed graph (also called digraph or quiver) consists of a set V of vertices, and a set E of edges, and maps $E \to V : e \to (s(e), t(e))$, where s(e) is the source and t(e) is the target of the directed edge e.

Formatted: Not Highlight

Formatted: Not Highlight

Strongly Connected Graph: A directed graph that has a path from each vertex to every other vertex is called strongly connected.

Cyclomatic Number: The cyclomatic number v(G) of a strongly connected directed graph is equal to the maximum number of linearly independent cycles. Equation 1 gives the cyclomatic number

$$v(G) = e - n + p \quad (1)$$

where there are e edges, n vertices and p separate components.

A fundamental concept in measurement is the definition of units of measurement. It must be noted that the identification of measurement units in the McCabe cyclomatic number definition has not been explicitly documented. This topic is studied next.

First, on the left hand side, the units can be derived from the definition itself, that is 'a cycle'. The characterization of this concept can be derived from the definition itself, that is 'linearly independent cycles in a strongly connected graph'. A numerical assignment rule is also given in the definition, that is the 'maximum number' of linearly independent paths. It can be noted that meta-model is not stated directly from the left hand side, but indirectly from the related definitions of 'strongly connected directed graph'.

Second, on the right hand side, there are three distinct types of units, that is: edges, vertices and 'separate components'. A numerical assignment rule is given next in the format of addition-subtraction of these numbers; however, such an addition-subtraction of different types of units is explicitly invalid

in a numerical sense when taking into accounts that different types of units are involved (for instance, it does not make sense to add apples to oranges). It could be valid only provided that these units be transposed into a higher level of abstraction with another type of more generic attribute and with a corresponding unit (for example: three fruits of the type of apples can be added to two fruits of the type oranges to give a total number of five fruits – in this example, fruits is a generalization of both apple and oranges, and when you add fruits, you loose information at the lower level of abstraction, that is 'five fruits' does not provide information about how many apples nor about how many oranges).

In summary, the graph theory has not documented clearly how the units are manipulated in their measurement procedure and, therefore, lacks clarity, which lacks of clarity can make it challenging to transpose these concepts into other f ields.

4.3 McCabe Cyclomatic Number for a software

A key contribution of McCabe has been his attempt at transposing into software the cyclomatic number from graph theory. A summary of this transposition is presented next.

According to McCabe [WAT96], the program is modeled as a *control flow graph*. A control flow graph is an abstract structure used in compilers. It is an abstract representation of a procedure or program, maintained internally by a compiler. Each vertex in the graph represents a basic block. Directed edges are used to represent jumps in the control flow. There are two specially designated blocks: the entry block, through which control enters into the flow graph, and the exit block, through which all control flow leaves.
Program control flow graphs are not strongly connected, but they become strongly connected when a virtual edge is added connecting the exit node to the entry node [WAT96].

Since the control flow graph has been transformed into a strongly connected graph, the graph cyclomatic number can be applied to this representation of programs. So, the cyclomatic number, when applied to software in McCabe transposition, becomes :

$$v(G) = e - n + p + 1 \text{ Virtual edge} \qquad (2)$$

Addition of 1 to Equation 1 is in order to integrate the supplementary virtual edge.

Furthermore, in McCabe transposition only individual modules are taken into account, instead of the whole software. And, in that particular situation defined by McCabe, the number of connected components *p is* always equals 1. McCabe then defines Equation (2) as

$$v(G) = e - n + 2 \qquad (3)$$

Again, when analyzed taking into consideration the units of each related elements being added-subtracted, equation (3) is quite puzzling. Indeed, in addition to the difficulties already noted in section 4.1, the number 2 is derived from an invalid addition of different units, that is, one 'connected component' and one 'virtual edge'.

A proper rewriting of (2) taking the units into consideration would lead to (4) instead of (3), that is:

$$v(G)^{independent\ cycle} = e^{edge} - n^{nodes} + p^{connectedcomponents} + 1^{virtual\ edge} \qquad (2)$$

becomes

$$v(G)^{independent\ cycle} = (e+1)^{edge+\ virtual\ edge} - n^{nodes} + p^{connectedcomponents} \qquad (4)$$

Of course, adequate interpretation of units in equation (4) remains an issue.

4.4. McCabe Cyclomatic Complexity Number

Finally, based on Equation 3, McCabe suggested a measure of a program complexity, *i.e.* cyclomatic complexity, which he interpreted as *the amount of decision logic in a single software module.* [WAT96] In other words, it is inferred that a software module is represented by a control flow graph and its *cyclomatic complexity is defined for each module to be e - n + 2, where e and n are the number of edges and nodes in the control flow graph, respectively*[WAT96].

Moreover, McCabe refers to the cyclomatic number of control flow graph as cyclomatic 'complexity' number: McCabe, while using the term 'complexity', does not provide his definition of complexity, nor of the attribute itself, nor of his direct characterization. His approach is basically his mapping of the concepts he selected from graph theory into his view of software as a control flow graph.

5. Analysis of McCabe Number

5.1 Scale types

In [Zus97], a proof of the scale type associated with the cyclomatic number is suggested. According to the author, the number of McCabe can be validated internally as an ordinal, ratio, and absolute scale. Zuse suggests a specified operation of flowgraphs concatenation in order to prove the ratio scale. Nevertheless, using the cyclomatic number as a ratio scale requires working with the additive operation specified in [Zus97]. So, computing averages and standard deviations on this number must be carefully done. Indeed, we must ensure that the numerical addition corresponds to the empirical operation proposed by the author. In other words, does the numerical addition means the same as the proposed empirical operation ? This work remains very interesting and important, since it provides mathematical foundations for the scale type.

5.2 Interpretation of McCabe number in industry

When only the left hand side of equations (2) or (3) is taken into account, the cyclomatic number is defined in terms of units of 'number of cycles' and represents clearly a quantification that can be qualified as a 'count': this operation produces clearly numerical number of the *integer* types, which numbers are clearly on a *ratio* scale: for instance, 10 independent cycles is thus clearly five times the number of 2 independent cycles.

However, with the term 'complexity' being used to label the measure, users of the number derived from the above equation do not seem to interpret this number on a ratio scale. They seem rather to *interpret* this number on something like an unspecified interval scale which would be something of the type of an *exponential* scale. This is particularly clear when the number is used as an indicator for testing effort. For instance, having two programs P_a and P_b with cyclomatic numbers of 10 and 5, respectively, we are inclined to interpret that the testing time P_a to be much more than twice the testing time of P_b. In fact, the question is to be sure that the cyclomatic number, that is the 'number of independent paths' correspond indeed to the number of the paths to be tested (or at least to be in linear relationship with it). Such a relationship is worth to be determined clearly.

More generally, it seems that there is a transposition of the ratio scale of the cyclomatic number made by users of the McCabe number. A possible explanation is maybe related to their intuitive individual perception of the complexity of the graphical representation of the control flow graphs presented in conjunction with the McCabe number. So, while a graph of a cyclomatic number of 5 seems easy to be represented visually, a graph of a cyclomatic number of just the double 10, seems much more harder to be represented visually and a graph of a cyclomatic number of 20 is a real disaster.

It appears therefore that, using both the control flow graphs and McCabe number, the practitioners build their own interpretation scale. Of course this is done implicitly, without clear rules and non measurement-related convention.

This above discussion is related to the distinction between the psychological complexity and the computational one [WIT97].

5.3 Discussion on Cyclomatic Complexity Entity & Meta Model

The entity measured by the cyclomatic complexity number is a control flow graph. Indeed, *control flow graphs describe the logic structure of software modules. A module corresponds to a single function or subroutine in typical languages* [WAT96]. According to McCabe, the measured entity is the source code of a given module, which corresponds to a functions or a subroutine. So, the meta-model is made up of all elements present in a graph, *i.e.* vertices and edges. *The nodes represent computational statements or expressions, and the edges represent transfer of control between nodes* [WAT96].

But, do graphs correctly represent the source code entity in order to measure its cyclomatic number? In other words, is the assumption concerning the one-to-one relation of a given module source code and its corresponding graph verified?

One module source code is related to one and only one graph. But, the contrary is not necessary true. That is, one graph is related to one or many source codes. In [ZUS97], this set of source codes is considered as an *equivalence class* of source codes. So, it is not obvious that the final source code corresponds to the measured graph. And, in that sense, it could be interesting to test this assumption, which argues a one-to-one relation between a measured graph and a final source code.

Another point discussed in [WAT96] is the "virtual edge" added to the control flow graph in order to obtain a strongly connected graph. The rationale of that *work around* is to allow the use of the cyclomatic number of a strongly connected graph. But, adding a "virtual edge" does not modify the nature of the considered entity, *i.e.* source program. The justification of this "virtual edge" is as follows: *This virtual edge is not just a numerical convenience. Intuitively, it represents the control flow through the rest of the programming which the module is used* [WAT96]. So, it can be seen as the connection between the module and the rest of the program.

5.4 Discussion on Related Attribute

In McCabe, only the concept of *cyclomatic number* which refers to graphs is explained. However, by adding the label 'complexity' to the 'cyclomatic number' expression, McCabe leads the readers to believe that the attribute he considered is the complexity of a source code program, but does not document explicitly this claim by association.

Is this claim by association relevant, and valid? As mentioned in the previous section, using graphs to model source code programs can be erroneous. And, therefore, the cyclomatic complexity number applied to a software module can be hardly relevant. In other words, it is necessary to explain the assumption related to the applicability of graph theory concepts, *i.e.* cyclomatic complexity, in the measurement of software.

Formatted: Not Highlight

It could be interesting to ensure that this assumption does not imply some risks when the measurement results are used in the context of a testing effort planning or a error rate estimation.

In regard with Section 3, many definitions of complexity can be found in the software measurement literature. In that context, complexity concept remains a concept about which no consensus has yet emerged for its measurement in software engineering. The vagueness of the concept indicates the misunderstanding of complexity. So, referring to the label of 'cyclomatic complexity number' of the graphs of a program bypass the requirement of an explicit definition about source code program complexity, which remains to date difficult to formulate and agree upon in the community.

6. Discussion

In the Abran & Jacquet framework, the distinct entities of the generic measurement process have been clearly identified:

- Step 1 = the design of a measure (eg.. the design of a measure of temperature in Celsius degrees)
- Step 2 = the application of a measurement design to a particular context (e.g. the act of measuring temperature with a measuring instrument in a particular context)
- Step 3 = the results derived from the application of a measurement procedure, eg. the temperature itself measured in Celsius at a particular day, in a particular location (and other recorded context information, such as wind conditions, etc.)
- Step 4 = the use of these measurement results for evaluation or decision making, usually through the use of interpretation scales for specific context or the use of models of relationships across measures of multiple distinct entities and attributes (eg.: at 20 Celsius, somebody will go play golf, others not because it might be too windy for them; somebody will go to the beach, others not because the water temperature is not warm enough, etc.).

In the software measurement literature, many researchers have often been not precise enough when discussing the use of the measurement results in various models of evaluation by not using the appropriate specialized terminology: for instance they still use the measurement terminology when they discuss the use of their *measurement results* rather to use the terminology of theoretical hypothesis and of experiments to explore whether or not there is a relationship across multiple variables, what are the strengths of such relationships, and how much these relationships vary (or not) under various conditions.

In McCabe's work, this lack of recourse to the terminology appropriate to the statement of an hypothetical theory, and required experimental work has led to ambiguous statements and insufficiently supported claims. For instance, McCabe by adding the term 'complexity' to the 'cyclomatic number' has explicitly made an association between this number and a concept of complexity, but has neither described explicitly such association nor associated quantitative numbers to this association nor described under which conditions it would hold for particular quantities.

This lack of clarity has led, for example, practitioners, and researchers alike to make associations that have low proven quantitative properties:

- *Complexity can be used directly to allocate testing effort by leveraging the connection between complexity and error to concentrate testing effort on the most error-prone software* [WAT96]. This assertion by McCabe has led to a generalization such as: "The more the cyclomatic complexity number is high, the more the error rate is high" derived from the McCabe assertion that a relation exists between the cyclomatic number relabeled as 'complexity' and the testing effort. Of course, the 'more the error rate is High' and 'the most error prone' is clearly not on a ratio scale, but at best on an ordinal scale.
- *Cyclomatic complexity measure correlates with errors in software modules* [WAT96]. Again this statement has led users of McCabe number to associate low errors with low cyclomatic number. However, a coefficient of correlation (r) between two given variables X & Y does not measure any causality relation between those variables. A coefficient close to 1 does not mean that a variable implies the other, it simply expresses that the two variables vary in the same direction.
- *Maintainers can keep maintenance changes from degrading the maintainability of software by limiting the cyclomatic complexity number during a modification* [WAT96]. The same comments apply.

7 Conclusion

McCabe cyclomatic number is both widely quoted in the software measurement literature and widely implemented in automated software measurement tools. However, there is still considerable ambiguity in both the design of this measure and use of the measurement results in evaluation models.

This paper has investigated in particular the measurement foundations of this 'number' and highlighted a key problem of measurement units which indicates that related concepts have not been adequately neither explored nor explained. Without such knowledge and insights, it is difficult to improve such a design.

On the other hand, the artificial labeling of the 'cyclomatic number' as a 'complexity' concept has led to considerable ambiguity on the use of this number as a 'measurement' number rather than as a qualitative empirical model which vary according to the empirical contexts.

Further research is required to:

- Provide an adequate description of the treatment of measurement units or, alternatively, a confirmation that such a transformation is invalid
- explain the transformation of scale types from the ratio scale in the 'cyclomatic number' to the 'ordinal' scale on a look-alike log scale in the evaluation models of associations with 'testing' for 'error-prone' modules, for instance.

Finally, it is important to note that the analytical framework use to study the McCabe number can be used to study any other candidate software measure.

References

[BER01] Cl. Berge, *Theory of Graphs*, Dover Publications, 2001.

[EVA87] M. W. Evans and J. Marciniak. *Software Quality Assurance and Management*. John Wiley and Sons, Inc., 1987.

[HAB04] N. Habra, A. Abran, M. Lopez & V Paulus, Toward a framework for Measurement Lifecycle University of Namur, Technical Report, TR37/04, 2004 *(to be published)*.

[HEN96] B. Henderson-Sellers, *Object-Oriented Metrics. Measures of Complexity*, Prentice Hall, 1996.

[IEEE90] Institute of Electrical and E. Engineers. An Empirical Evaluation (and Specification) of the all-du-paths. *IEEE Standard Computer Dictionary*: *A Compilation of IEEE Standard Computer Glossaries*, 1990.

[FEN96] N. Fenton N. and S.L. Pfleeger, *Software Metrics – A rigorous and Practical Approach*, 2nd. International Thomson Computer Press, London (1996).

[JAC97] J.-P. Jacquet & A. Abran, "From Software Metrics to Software Measurement Methods : A Process Model," in *Proceedings of the Third International Symposium and Forum on Software Engineering Standards*, ISESS '97, Walnut Creek (CA), 1997.

[KIT95] B. Kitchenham, S.L. Pfleeger & N. Fenton, Towards a Framework for Software Measurement Validation, *IEEE Transactions on Software Engineering*, Vol.21 n°12, December (1995), pp. 929-944.

[LOP03] M. Lopez, V. Paulus & N. Habra, Integrated Validation Process of Software Measure. In *Proceedings of the International Workshop on Software Measurement* (IWSM 2003) (2003).

[MOF03] Meta-Object Facility (MOF™), version 1.4 at http://www.omg.org/

[SEL04] A. Sellami and A. Abran, The Contribution of Metrology Concepts to Understanding and Clarifying a Proposed Framework for Software Measurement Validation *(to be published)*.

[WAT95] A. H. Watson McCabe Complexity *in Software Development Systems Management Development*, Auerbach 1995.

[WAT96] A. H. Watson and T. J. McCabe. Structured testing: A testing methodology using the cyclomatic complexity metric. *NIST Special Publication*, 1996.

[WEB04] Merrian-Webster Online http://www.m-w.com/

[WIT97] S. A. Whitmire. *Object-Oriented Design Measurement*. Wiley Computer Publishing, 1 edition, 1997.

[ZUS97] H. Zuse,. *A Framework for Software Measurement*. de Guyter, 1st edition, 1997.