A Framework for Evaluation and Validation of Software Complexity Measures

Sanjay Misra[1], Ibrahim Akman[1], Ricardo Colomo-Palacios[2]

[1]*Department of Computer Engineering, Atilim University, Ankara, Turkey*
[2]*Computer Science Department, Universidad Carlos III de Madrid, Spain*

smisra@atilim.edu.tr, akman@atilim.edu.tr, ricardo.colomo@uc3m.es

Abstract. This paper proposes a framework for the evaluation and validation of software complexity measure. This framework is designed to analyze whether or not software metric qualifies as a measure from different perspectives. Unlike existing frameworks, it takes into account the practical usefulness of the measure and includes all the factors which are important for theoretical and empirical validation including measurement theory. The applicability of the framework is tested by using cognitive functional size measure (CFS). The testing process shows that in the same manner the proposed framework can be applied to any software measure. A comparative study with other frameworks has also been performed. The results reflect that the present framework is the better representations of most of the parameters which are required to evaluate and validate a new complexity measure.

Keywords: Software quality, framework, metric, measure, evaluation, validation criteria,

1. Introduction

The requirement of improving quality is the prime objective in developing software. The quality objectives may be listed as performance, reliability, availability and maintainability, and are closely related to software complexity. There is continuous effort to produces new complexity measures [1-4] to achieve quality objectives in software processes, projects and products. A complexity measure must possess such properties as validity, sensibility and usefulness. These properties can be investigated by using evaluation and validation criteria. In general validation and evaluation criteria for complexity measures must be very sound in order to evaluate all aspects of the complexity measure. Existing literature provides a variety of frameworks and proposals [5-7] for evaluating different aspects of software quality. One can also find a number of distinct proposals [8-10] for the evaluation or/and validation of software complexity measures. Unfortunately, too little efforts have been done to develop a complete framework for evaluating software complexity measures. In addition, available proposals are confined to the evaluation of the software metrics quality from only the measurement theory (MT) perspective.

Property based software engineering measurement [11], Goal Question Metric (GQM) paradigm [12-13], IEEE standards 1061 [14], ISO/IEC 9126 Quality Standard [15-18], ISO/IEC 15939 [48], approaches based on MT [11], [19], [20], [21], [22] and [23] and Weyuker's properties [24] have attracted special interest in the last two decades. However, they fulfill the requirements of validation only up to a certain limit and not solve the purpose of complete validation of complexity measures. Kitchenham et al. [9] pointed out that one should consider the important aspects from all important validation criteria for a complete evaluation and validation. Further, software development methods and procedures have changed rapidly over the last two decades and, hence, require new complexity measures. All these factors have led to the need for more efforts and researches to develop new complete and practical evaluation criteria which also enhance and combine features of previous techniques.

The validation of a metric is the most important task in the development of a metric program. However, after a rigorous literature survey it is observed that majority of the metrics available in the literature do not follow proper guidelines for evaluation and validation process. Fenton stated that there is no match in content between the increased level of metrics activity in academia and industry [25]. One of the reasons for this crisis is due to improper validation process. Most of developers of software metrics follow different criteria for the evaluation of their metric. Some give emphasis on empirical validations while others on theoretical validations and also the way of the validation process for theoretical and empirical validation varies from metric to metric. There are no concession on common standard for evaluating and validating the metric program.

Based on the above rationale, in the present work, a formal framework for evaluating and validating software complexity measures is presented. The proposed framework is the integration of several aspects of evaluation and validation process and based on existing validation and evaluation criteria. These criteria are selected and formed in a way that the proposed framework integrates the distinct validation perspectives. More specifically, we analyzed the available validation and evaluation criteria, extracted their important features, suggested some additions/modifications (if required) and then presented them in a formal framework. This framework proposes all the essential conditions for a new complexity measure and includes criteria for practical evaluation, evaluation through perspective of MT and scale measurement. It also proposes concise empirical validation in two stages as preliminary and advanced empirical validation to facilitate the software community to perform short experimentation based on available data (based on situation and circumstances) in the initial stage and real life projects from the industry in the advanced stage. A model for proper empirical validation for software complexity measures has already been presented by one of the authors of this paper [26]. In the present work, this model is considered as one of the components of the proposed framework. Finally, evaluation through self assessment is recommended because, the developer of a new metric is the person best placed to make constructive criticism as he/she knows the drawbacks of his/her proposal.

In the academic community there is no concession on the definition of complexity and complexity measure. The term 'complexity' is used in many of the twenty-five roadmaps for software [4] and is used in almost everywhere in computer and software engineering. IEEE defines [27] software complexity as “the degree to which a system or component has a design or implementation that is difficult to understand and verify”. Briand et al. [11] state that complexity is defined as an intrinsic attribute of an object and not as psychological complexity as perceived by an external observer. The authors have further identified complexity as a measurement concept, which is different than size, length, cohesion, and coupling [11]. Zuse [19] defines the software complexity as it is the difficulty to maintain, change and understand software. Amongst the available definitions, the definition of complexity defined by IEEE [27] is followed in this paper. Actually, IEEE [27] and Zuse [19] both represent the same notion of complexity. However, it has different interpretations in different contexts. The definition of complexity proposed by Briand et al. [11] is not considered because of two reasons: 1. the authors themselves accept that they are not considering the approach of complexity, where studying the impact of software on other systems, 2. Complexity of a software system may not be a function of a single element. All the factors which make the software/software system/software language difficult to understand, maintain and verify are the function of complexity. These functions include size, length, cohesion, coupling and several other elements/factors which are responsible to increase the complexity. This view is also supported by LAKE [28] who has divided the software complexity measures into size, data structure, control flow, information flow and software science, measures. It should be noted here that the definition of complexity considered in this paper has no contradiction with the definition of complexity by Briand et al., but the definition of complexity is considered in broad sense. In this respect, if any measure is a size measure according to Briand et al. [11] the presented framework is also applicable on that measure, since size is considered as a factor of complexity in the proposed framework.

In addition, the terms measure and metric are used interchangeably in this article since they are given anonymously in the literature [19]. Actually, these terms have similar definitions. IEEE defines [27] metric as ‘a quantitative measure of the degree to which a system, component, or process possesses a given attribute’. Pressman [29] explains the measure in software engineering context as ‘a measure provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attributes of a product or processes.

The remainder of this paper is organized as follows: a brief analysis of existing important validation criteria is given in section 2. The introduction of the cognitive functional size (CFS) complexity measure [30] on which the proposed framework is applied is also summarized in section 2. The proposed framework is given in Section 3 in detail. A comparison with the similar frameworks is demonstrated in section 4. The conclusions drawn are given in sections 5.

2. Literature Survey

This section introduces the standards and frameworks for the validation process. In section 2.1, some of important validation criteria are discussed and evaluated. The outcomes of the analysis of these criteria will become the base for the construction of the proposed framework. The other related works and frameworks are introduced in section 2.2. The CFS, which is applied on the proposed framework, is defined section 2.3.

2.1 Existing standards and common approaches

IEEE Standard 1061 [14] proposes some validation criteria to which a software complexity metric should adhere. A metric may be valid with respect to certain validity criteria and invalid with respect to other criteria. The validity criteria are correlation, consistency, tracking, predictability, discriminative power and reliability. These are expressed in terms of quantitative relationship between the attribute being measured and the metric. The existence of quantification is one of the major problems for validation. Further, IEEE standard 1061 tries to solve this problem by suggesting the use of a direct metric which does not depend on a measure of any other attribute and is assumed to be valid by itself. However, Kaner and Bond [31] found this approach a weak and risk-prone substitute for a casual model. In their work they also questioned the usage of direct metrics for the quantification problem of practical evaluation through their user-dependent, subjective multidimensional function characteristics.

Apart from these criticisms, we observed that IEEE standards are not totally inadequate and are valuable for evaluating the complexity measure up to a certain limit. These In fact IEEE standards are based on MT. Further, once a metric is validated through MT concept, it also fulfills the conditions required by IEEE standards. Therefore, evaluation of a proposed metric against MT is suggested in the proposed framework.

ISO/IEC 15939 [32] is an international standard for software measurement process, which guide how to define a suitable set of measures that address specific information needs. In this standard, the software measurement process consists of four activities: Establish & sustain measurement commitment, plan the measurement process, perform the measurement process, evaluate measurement.

Although, all above activities are common practices in measurement, they are important to establish a measurement program. On the other hand, in ISO/IEC 15939 standards its activities and tasks are defined at a very high level and, therefore, additional support is necessary for ease of implementation [33]. By keeping the importance of these activities in mind, all those are included in the proposed framework in such a way that they provide a proper evaluation process for software complexity measure.

The Goal-Question Metric (GQM) approach is used for practical evaluation and is proposed by Basili et al. [13]. GQM is based on the idea that all measures in a measurement program should be meaningful and is used to decide why and what to measure. GQM has three levels: Conceptual level (defines the goal of the measure), operation Level (goals are refined into questions) and quantitative Level (questions are refined into the metrics). The GQM approach is useful for evaluating the practical usefulness of the proposed measure. However, due to existence of too many questions to be answered and too many corresponding metrics to be measured, it is not practical and difficult to administer.

It is recommended to reduce the number of questions and corresponding metrics in order to make practical evaluation easier to apply. A set of basic items are identified for this purpose in the proposed framework, which is not only concise but also covers most of the important factors for the practical evaluation of any complexity metric.

Measurement is simply the process of converting qualities into quantities. Such a conversion process requires a formal description of the systems worked on. The establishment of the scale is also an important issue for software metrics [34]. As a consequence, the developer will be able to know to which scale a metric pertains, and, behind the scales, the empirical properties of software measures are hidden. Therefore, a proposal of a new software metric should be validated through the application of MT basics.

Against this rationale, the proposed framework considers evaluation through the MT perspective as an essential and important criterion for software complexity metrics. The guidelines are extracted from almost all the proposed existing validation criteria in the literature. These guidelines are not only easy to apply but also give sufficient information about the metric, which is important from the MT perspective.
All the above proposals are used for theoretical evaluation and validation of software complexity measures. To evaluate the practical applicability of the proposed software complexity measure, the empirical validation is only the way to prove the worth of the proposal. On the other hand, the situation of the empirical validation for software complexity measures is not very good and it is due the fact that developers of the complexity measures are not using proper criteria for validating their metrics/measures empirically [26]. In several cases the design of experimentations performed for validation were poor [35]. Based on this rational, the proposed framework suggests the application of empirical validation in two parts namely as preliminary empirical validation and advanced empirical validation. In fact, one of the authors this paper has presented a model for proper empirical validation [26]. The same approach is followed in framework.

2.2 The **Other related works**

The literature provides other proposals/frameworks which address how to develop measurement or metric programs. A Framework for Software Quality Measurement [6], DISTANCE: a framework for software measure construction [7], methodology for validating software metrics[8], towards a framework for software measurement validation [9], property based software engineering measurement [11], a methods for obtaining correct metrics [36], production and maintenance of software measurement models [37], a practical view of software measurement and implementation experiences within Motorola [38], and evaluation criteria for Object Oriented Metrics[39], are examples of such proposals. These proposals will be discussed and compared with the proposed framework in section 5.

2.3 Cognitive functional size (CFS) complexity measure

The proposed framework is demonstrated by using CFS [30]. According to Wang et al. [25], the cognitive weight of software is defined as the extent of difficulty or relative time and effort for comprehending given software modeled by a number of basic control structures (BCS). This metric was demonstrated by a case study in three different languages and validated by number of examples [30]. This metric was further evaluated through Weyuker's properties [40] and its extended version through the principles of measurement theory [41].
In the last 8 years, this metric become base for several new proposals/metrics, and several papers are produced based on it. The metrics based on cognitive weights/CFS are summarized and compared in [42]. Further the most of the well known metrics like cylomatic complexity, Helstead metrics are already been researched, evaluated and validated (through all possible criteria) in the literature. The results of any evaluation criteria on these metrics are well known. Cognitive complexity metrics based on cognitive informatics are in nascent stage and CFS is the base of all the cognitive complexity metrics. A rigorous evaluation of CFS may also help to those researchers who are working on cognitive complexity measures based on cognitive informatics. These are the reasons for us to select CFS for applying it on the proposed framework.

3. The proposed framework and its application

The practical success of any proposed metric depends on the establishment of (1) its validation, (2) understandability by its users and (3) a tight link between the metric and the attribute that it is intended to measure. Therefore, a new metric must be evaluated practically and formally for its validation.

In the proposed framework (Figure 1), the first step is to evaluate the practical utility of the proposed measure. In this step, the main concern is "how to recognize and describe the attribute in the empirical observation domain in order to relate its values to the proposed metric." A criterion for practical evaluation is proposed in section 4.1 for this purpose.

In order to make the software discipline more and more mature, tools provided by MT should be used. As a consequence, the second step is to evaluate a proposed measure against the concept of MT. However, MT has problems in establishing the empirical relations between entities [43]. Therefore, authors are in favor of the procedure in which once the developer of a new complexity measure establishes an empirical relation between entities then he/she should validate the metric through the representation condition as given in section 4.2. This stage should also include the establishment of a scale for the proposed measure with admissible transformation and extensive structure.

Empirical validation constitutes the third step of the proposed framework to characterize the practical utilization of the metrics. The empirical validation is categorized in two parts: Preliminary and Advanced. The preliminary empirical validation includes the initial validation of the metrics by applying it to different test cases and examples. In advanced empirical validation the new metric is tested by using real projects from the industry.

After practical, theoretical and empirical validation, and establishing a scale for the measure, the proposed framework contains a set of desirable properties to which the new complexity measure should adhere. These properties prove the usefulness and robustness of the measure. They are simple and general in nature.

It is generally observed that for most of the new metrics/measures, the developer tries to prove his/her metric to be the most suitable measure for any particular attribute (e.g. 30). This, however, is not true in most of the cases since every measure has its own advantages and disadvantages. Therefore, the next and last step in this framework is to discuss the key advantages and disadvantages of the proposed measure. It is also observed that, in most of the investigations, there are opportunities for the improvement in the proposed work; therefore the possibility of future work should also be discussed as a part of the framework. In the last, after complete evaluation and validation, one can reach at the acceptance phase, which will provide the software community a clear idea about whether the proposed metric should be accepted or not.

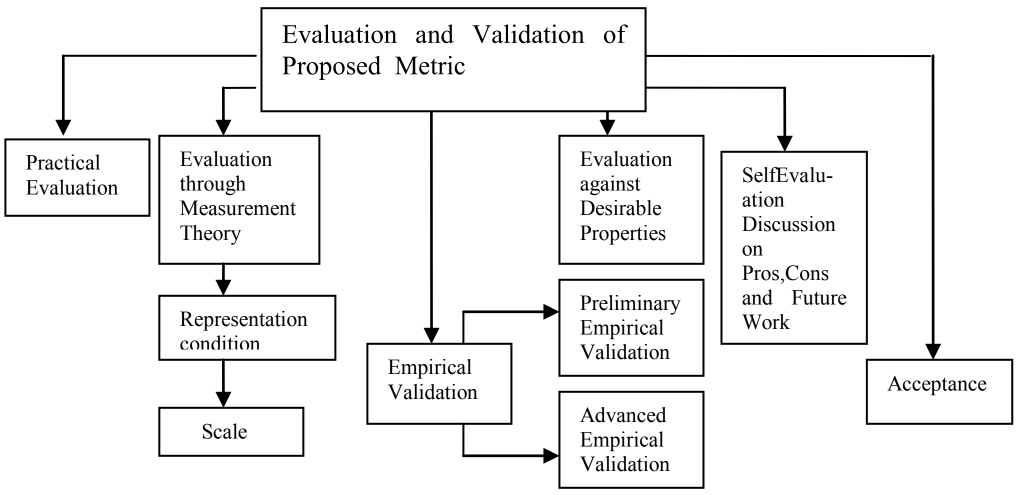
Evaluation and Validation of Proposed Metric
Practical Evaluation
Evaluation through Measurement Theory
Representation condition
Scale
Empirical Validation
Preliminary Empirical Validation
Advanced Empirical Validation
Evaluation against Desirable Properties
SelfEvalu-ation Discussion on Pros,Cons and Future Work
Acceptance

Figure 1. Proposed Framework

In order to make it easier for the reader to follow and for the sake of economy, each stage of the proposed framework is explained with its application on CFS in the following subsections.

3.1 Practical evaluation

A practical evaluation of software metrics is necessary since it helps to observe them in an experimental sense [44] and proves the practical utility of the proposed metrics.

Besides this, many important issues faced by the software engineering community can only be addressed practically [34]. Various authors [9, 10, 14, 31] attempted to address practical validation in the past but usually used different approaches. For example, Schneidewind [8] proposed a methodology based on six validity criteria, while Fenton [22] suggested that a metric is valid if it can be shown that it gives a proper numerical characterization of some attributes.

In this section, a set of criteria which is concise in size and includes most of the important parameters necessary for practical evaluation is proposed. For practical evaluation, studies by Basili et al. [13], Zuse [20] and Kaner [31] are used to extract these features and present them in a formal way.

1. *Objective or goal of measure*: The objective or goal of the measure includes evaluating project status, evaluating staff performance, self-assessments and improvement, informing others about characteristics (such as development status or behavior of product) and informing external authorities about the characteristics of the product.

In the present case the two main objectives of CFS are to contribute to the judgment about product quality and to provide self-assessment and improvement for the developer.

2. *Identify the users and scope of the measure*: The scope of the measure should be clearly defined. As the scope broadens, more confounding variables can come into play, potentially impacting on the metric.

The CFS is not related to the process through which the software is developed and it can be categorized as a technical metric being applicable after coding. Consequently, its scope of use is the software development group.

3. *Identify the entities and attributes to be measured*: It should be clearly defined which attribute of the entity the metric is trying to measure. Is it quality of the product, effectiveness of testing, thoroughness of testing, effectiveness of the tester, skill or diligence of the programmer, reliability of the product etc?

In case of CFS, the entity is the software code and the attributes measured by CFS are the quality of the product and the developer. A more complex product makes it less understandable and consequently less maintainable for future development effort. Also, the developer who can satisfy the user requirements through reduced amount of input/output and lesser quantity of branching and looping primitives (implying small time-complexity) is assumed to be more skillful. Note that the CFS may not be a unique and complete measure for the above attributes.

4. *Definition of metric and its measuring methods/instruments:* Define the function that assigns a value to the attribute. In addition, identify the way, method or instrument by which it is measured.

For the case of CFS, the metric has been defined formally in Section 3. The items to be counted are the number of input, output and cognitive weights of different basic control structures. For automated counting purpose, one can easily develop a token generator and use string matching algorithms.

5. *Relationship between attribute and metric*: The relationship between attribute and metric should be determined in order to quantify the attribute affect of the metric.

In fact, it is subjective to construct a relationship between metric and product quality. However, quality of products depends on several factors and sensitive software complexity metrics are one of the tools to control the quality. This concludes that there is direct relation with CFS and the quality of product. If the CFS value increases, it is clear that the product quality will decrease. This is also true for all other sensible measures/metrics.

3.2 Evaluation through the perspective of measurement theory

In the previous section, the details of the first stage of the proposed framework, which evaluates the proposed measure by identifying its entities, attributes and objective, is given. In this section the guidelines for evaluation of a new metric/measure against MT perspective are provided since the relation between MT and evaluation criteria for software complexity measure is well established. These guidelines follow the general principles from the MT perspective, [19], [21], [23], [45] which are mostly accepted by software community and therefore important and necessary for the evaluation of any software complexity measure.

3.2.1 Evaluation of measure against representation condition

The components of the qualified system are (1) Entities whose attributes are targeted for quantification; (2) Empirical binary relations showing the intuitive knowledge about the attributes and (3) Binary operations describing the production of new entities from the existing ones. These components are reflected in the following definitions [23].

Definition 1: (Empirical Relational System-ERS)

For a given attribute, an Empirical Relational System is an ordered tuple $ERS=<E, R_1,...,R_n, o_1,..., o_m>$ where E : the set of entities, R_1, ..., R_n denote n empirical relations such that each R_i has an *arity* n_i, and $\mathrm{R_i} \subseteq \mathrm{E}^{\mathrm{ni}}$. o_1, ..., o_m denote m empirical binary operations on the entities that produces new entities from the existing ones, so o_j: $\mathrm{E}\times\mathrm{E}\rightarrow\mathrm{E}$ and the operations are represented with an infix notation, for example, $e_\mathrm{k}= e_\mathrm{i}$ o_j e_l.

According to this definition, the components of the quantification system are the values representing the decided quantities; the binary relations showing the dependencies among them and the binary operations describing the production of new values from the existing ones.

For CFS, the entities are the program bodies. The only empirical relation is assumed to be *more_or_equal_complex* and the only empirical binary operation is the *concatenation* of program bodies. This can be explained by a solid example. Assume that a program body P is given and a new program body Q is obtained by simply duplicating P. One may easily establish the relation *more_or_equal_complex* between P and Q.

Definition 2: (Numerical Relational System-NRS)

A Numerical Relational System is an ordered tuple $NRS=<V, S_1,...,S_n, p_1,..., p_m>$ where V : the set of values, $S_1, ..., S_n$ denote n relations such that the arity of S_i is equal to the arity of R_i, and $S_i \subseteq V^{ni}$ *and* $p_1, ..., p_m$ denote m numerical binary operations on the values that produce new values from the existing ones, so p_j: $V \times V \rightarrow V$ and the operations are represented with an infix notation, for example, $v_k = v_i \, p_j \, v_l$.

For CFS, V is the set of positive integers, the binary relation is assumed to be $\geq$ and the numerical binary operation is the addition (i.e. +) of two positive integers.

Definition 3: Measure m is a mapping of entities to the values and it considers neither the empirical nor the numerical knowledge about systems, i.e. m: $E \rightarrow V$.

The measure for CFS complexity is defined by Equation (1). Note that a measure by itself does not provide any mapping between empirical and numerical knowledge.

Definition 4: A measure must satisfy the following two conditions known as *Representation Condition*:

(1) $\forall i \in 1..n \forall <e_1,\dots e_n> \in E_{ni} (<e_1,\dots e_n> \in R_i \Leftrightarrow <m(e_1),\dots,m(e_n)> \in S_i)$

(2) $\forall j \in 1..m \forall <e_1,e_2> \in E \times E (m(e_1 o_j e_2 = m(e_1) p_j \, m(e_2))$

The first part of the Representation Condition says that for a given empirically observed relation between entities, there must exist a numerical relation between corresponding measured values and vice versa. In other words, any empirical observation should be measurable and any measurement result should be empirically observable. The second part says a measured value of an entity which is obtained by the application of an empirical binary operation on two entities should be equal to the value obtained by corresponding numerical binary operation executed over individually measured values of entities. In other words, the complexity of the whole should be definable in terms of the complexities of its parts.

For CFS, the representation condition requires that (1) if, for any two program bodies, e_1 and e_2 are in *more_or_equal_complex* relation (i.e.$<e_1, e_2> \in$ *more_or_equal_complex*) then the measured CFS complexity value of entity e_1 should be greater than the measured complexity value of entity e_2 (i.e. $m(e_1) > m(e_2)$) and vice versa. Considering the program bodies P and Q; if Q is the double of P then the number of BCSs, inputs and outputs for Q double. Consequently, for part (1) of the condition, it is possible to say that the empirically observed *more_or_equal_complex* relation between two program bodies leads to a numerical binary relation > among those entities or vice versa. However, part (1) is only satisfied if there are such clear empirically observable relations between program bodies for example P and Q.

For part two of the representation condition, the CFS complexity value of a program body which is obtained by concatenation (i.e. the empirical binary operation) of e_1 and e_2 is equal to the sum (i.e. the numerical binary operation) of their calculated complexity values. Therefore, CFS satisfies the second part of the representation condition. Overall, then, CFS satisfies the representation condition.

3.2.2 Evaluation of Measure based on Scale

In the previous sub-section, evaluation through the measurement theory stage of the proposed framework was explained. CFS is used as an example for this purpose. It was identified that the proposed measure is satisfied by the representation condition. Now, there is a need to find out the scale of the proposed measure. There are two ways for getting the scale of the measure: through admissible Transformation and extensive Structures. Admissible Transformation is the simplest way to find the scale of a measure. On the other hand, Zuse [19] has stressed the advantage of using extensive structure because it is one of the most important measurement structures which characterizes empirical conditions of reality, hypothesis of reality and empirical conditions behind the software measure. Therefore, for the sake of convenience for the developer of a new complexity measure, it is recommended that a complexity measure should be evaluated by admissible transformation or by extensive structure below [21, 23].

Admissible Transformation

Definition 6: A scale is a triple <ERS, NRS, m>, where ERS is an Empirical Relational System, NRS is a Numerical Relational System, and m is a measure that satisfies the representation condition.

Definition 7: Given a scale *<ERS, NRS, m>*, the transformation of a scale f is *admissible* if m`= f o m (i.e. m` is the composition of *f* and *m*) and *<ERS, NRS, m'>* is a scale. Based on admissible transformation, four different types of scales can be considered as follows [21]:

Nominal scale: each entity is labeled for categories and there is no ordering relation among them. An example of nominal scale is the labeling of given programs according to the name of their authors.
Ordinal scale: entities are categorized in the form of *total ordering*. The associated values make entities comparable. As an example, program bodies can be assigned degrees from 1 to 5 with comparative meanings (e.g. 1 for least reliable to 5 for the most reliable).
Interval scale: the difference between the assigned numerical values can be quantified in their amount. A new scale m` from *m* can only be obtained through transformations of the form m`= a**m* + b where a>0. An example can be the scale Celcius that can be converted into Fahrenheit.

Ratio scale: the ratio between the numerical values associated to the entities is used for quantification. The form of transformation is: m`= a**m* where a>0. The main difference between interval and ratio scale is the existence of true zero-point in ratio scale. An example of ratio scale is the LOC measure of the size of a program body.

For case of CFS, it can be very easily proved that <*ERS*, *NRS*, *m*> for CFS is a ratio scale. Reconsidering the two program bodies P and Q, Q/P =2 and then a=2. This implies that m`= 2**m*. Therefore, it can be informally stated that the proposed measure CFS is defined on *ratio scale*.

One way of establishing whether a given scale is a ratio scale or not is to investigate whether the scale's Empirical Relation System is an Extensive Structure or not [19],[45]. However, once one can get the scale of a measure with admissible transformation, which is easier to understand and require only preliminary knowledge of measurement theory, it is not recommended to follow the Extensive structure, which is more complex and uses more technical measurement terms in comparison to admissible transformations.

3.3 Empirical validation

Empirical studies are used to investigate the software development and practice for understanding, evaluating, and developing in proper contexts. It allows the analyst to test out the theories with the support of emipirical observations. It includes formal experiments, case studies and surveys observed in industry, the laboratory or classroom [46]. However, these empirical validation approaches are applicable for software measure only up to a certain extent. It is because empirical validation is generally applied with simple cases on proposed metrics in the literature [30]. Furthermore, there are researchers, who suggested performing empirical validation with students in class room environment [47, 48] e.g. controlled experiments (grad students), observational studies (professionals, grad students), case studies (class projects). Although it is arguable that students are the future software developers but experiments with students may reduces the practical value of experiments [46]. Validation process based on such data may be acceptable only for taking initial knowledge regarding some quality factors like understandability. This is because the proposed metric will be later used by the software professionals in the real environment and experiments. Additionally, case studies should be done properly using cases as much close to real environment as possible.

Ideally, a new metric should be applied to real projects in industry by the developers from industry and then its validity should be evaluated against other similar metrics. However, in many cases, the type of empirical study depends upon situation and circumstances and, in the initial phases of any new proposal, it is not always possible to apply a new metric directly to the real projects from industry. It is because of, if the developer of the metric is academician and at a particular time, not getting the proper real (industrial) environment, then he try to validate his proposal through other means(data and projects on Web).

By considering these practical problems related to empirical validation, the suggested guidelines in the proposed framework are categorized in two stages [26] as preliminary and advanced empirical validations. For detail of these stages, readers are referred to the read the paper [26].

a) Preliminary Empirical Validation: This phase is divided into two. The first phase includes small experiments, case studies and comparative study. The second stage includes the application of the metric on real cases from industry.
   1. This stage is based on short experimentations, case studies and contains a comparative study. This stage is especially recommended when data/project from the industry is not readily available. In this case, the applicability of the new metric should be tested against a number of different examples/case studies. These examples/case studies may be small in size and collected from the literature. A comparison with similar metrics is also proposed at this stage since it provides valuable information on the usefulness of the new metric.

The preliminary empirical validation for CFS is demonstrated by eight programs from Misra et al. [40] on the complexity measures (Table 1). For comparative study, well known complexity measures, such as statement count, cyclomatic complexity [49] and Halstead programming effort [50] are selected**.** Wang's et al. [30] has also used 20 programs from a book [51] for comparing with physical size.

Table 1. Comparative complexity values

Complexity measure	Statement Count	Cyclomatic Complexity	Effort Measure	CFS
Prog. 1	12	2	1859	8
Prog. 2	17	2	5191	9
Prog. 3	18	2	6237	9
Prog. 4	37	5	15556	46
Prog. 5	23	4	5079	30
Prog. 6	15	2	2869	14
Prog. 7	11	3	1221	21
Prog. 8	11	4	1039	30

Analyses of these programs give valuable information about the metric under investigation. If CFS is compared with other measures, similar trends are observed. Unlike other metrics, CFS complexity values are due to the internal architecture of the program, cognitive complexity, and structural complexity. It gives complexity values, which are small in number and easy to calculate in comparison to other metrics. A detailed analysis and a comparative study of CFS can also be found in [52].

2. In the second stage of the preliminary validation, one must have to apply the metric on a real project and then evaluate it against other metrics. In fact, usefulness of a new metric is validated by applying it to

data collected from a real project. One can find a project on the web; on which he/she can apply the metric to verify its applicability; however, he/she must have to take some contemporary projects from the industry.

b). Advanced Empirical Validation (Acceptance): A new software metric cannot be accepted as long as its usefulness is not proved from the software industry. For its acceptance from industry, the proposed metric must be applied by software developers, in different projects and in different environments. It is because of, only after performing a family of experiments one can build up the cumulative knowledge to extract useful measurement conclusion to be applied in practice [50]. After the series of experiments, the results should be analyzed and compared. If the new metric is proved to be better than the existing metric/metric program in an organization only then it will be accepted. Otherwise it may be left for further improvement. After the improvement of the metric the same validation process should be revisited from the beginning.

It is worth to point out that there is a practical difficulty in this stage because most of the software industries are likely to be unwilling to apply a new technology/metric since it is difficult to convince them that the metric is more beneficial in comparison to the existing ones. This is one of the reasons why most of the new metrics are not empirically validated. Nevertheless, advanced empirical validation is a must requirement for not only validating a new metric but also necessary for its acceptance **by** the industry.

Advanced empirical validation of CFS was not demonstrated by the developer of the metric. It may be perhaps due to the aforementioned practical difficulty. In addition, it does not fall into the scope of this article and is left as the task of future work.

3.4 Evaluation through a desirable set of properties

Practical evaluation covers practical utility. Evaluation through MT explores what is measured and why it is measured by using quantitative models. A new measure should possess a set of other simpler and essential properties against which a software complexity measure should be evaluated. The usefulness and applicability of these properties should be demonstrated by using different measures. For this purpose, statement count, Halstead programming effort, cyclomatic complexity and cognitive functional size (CFS) were selected. The applications of the proposed properties are summarized in Table 2. It is worth to mention that all these properties are not new to the academic and research community, but they have not been accumulated and presented in a formal way.

Property 1. **The measure should be simple.**

By “simple,” it is meant that the measure should involve only simple calculations for complexity and should not involve complex mathematical functions. There should be a trade off between the efficiency of a measure and efficiency of the computation.

Statement count or lines of code is the simplest measure of complexity; however there are several different criteria for counting the line of code. In Effort measure, Halstead took a statistical approach of the program complexity. This complexity measure uses straight and simple formulae to calculate length, volume and efforts. In cyclomatic number, McCabe developed a graph theoretic complexity measure for managing and controlling program complexity. The metric is independent of physical size and depends only on the decision structure of a program and hence is calculated from its flow graph representation. The calculation of CFS is simple and easy, since one can easily count all the variables used in the formulation of the CFS. It is also meaningful because it calculates the complexity of software by considering the internal architecture of the code. Against this backdrop, this property is satisfied by all measures.

Property 2. **The measure should be language independent.**

Language independency is suggested [54] in the wake of structured programming and should be adhered to any good complexity measure. A program can be developed using basic program constructs like assignments, selections and loops existing in all structured programming languages. If one calculates the complexity of these constructs separately and then forms a basis of complexity measure depending on these individual complexities and also includes the complexity due to implementation details, it is assumed that the measure is language independent.

Cyclomatic number and CFS are based on control structures and are the same in all programming languages. However, statement count and effort measure do not satisfy this property.

Property 3. **The measure should be developed on proper scale.**

In their study, Piattini et al. [34] stated that establishment of the scale is an important issue for software metrics and, therefore, this property is needed [19]. This means there is always a need for a scale upon which a comparison of two measures of the same type can be made. With such a comparison, one can observe which of the two measures is more desirable. For example, there is a realistic lower boundary, such as zero for number of errors.

For most of the reviewed complexity measures, no clear cut norms and scale are discussed in their original papers. For the case of CFS, the assignment of the upper and lower bounds of the complexity values should be investigated in the future. The scale of the CFS is observed on the ratio scale.

Property 4. **Metrics in metrics/measures suite should be consistent.**

Often, one metric alone is insufficient to measure the features of the design paradigm or to accomplish the objectives of the software project. This suggests that a collection or suite of measures is needed to provide the range and diversity necessary to achieve the software project's objectives. A suite of measures adds an additional consideration. If a smaller value is better for one type of measure in the suite, then smaller should be better for all other types of measures in the suite.

For cyclomatic complexity and statement count, suites of measures are neither proposed nor required. For Halstead programming effort measure, size, length, volume and effort are proposed in a suite. For CFS, only the cognitive functional size of code is calculated, which means there is no need for a suite of measures.

Property 5. **A measurement should have some foundation that can be explained or visualized**.

If a measure cannot be explained with some fundamental unit or relative scale, then the value becomes arbitrary, in which case the reliability of the measure becomes suspect.

The statement count or line of code is the prediction of size, effort measure relates to efforts required (it provides estimates for number of errors and the manpower required for software development), cylomatic complexity is related to control flow complexity and the CFS relates the complexity of the program with the cognitive aspect. Therefore, it can be concluded that all these measures have a strong foundation for their proposal.

Property 6. **The measure should give the complexity as a positive number**.

A measure giving no information or negative information should not be considered as a measure. However, the complexity value can be zero for a program if it has only sequential assignment statements.

All the complexity measures under consideration give the complexity values in positive numbers.

Property 7. **The measure should differentiate between the complexities of the basic program constructs.**

This ranking is essential even intuitively. The ranking can be done either intuitively in the increasing order for assignment, selection, while-do and do-until etc. or measures like work function and entropy and information content may be used to find their individual complexities.

The statement count and effort measure cannot differentiate between the complexities of basic program constructs. Cyclomatic complexity metric depends only on the decision structure of a program and hence is calculated from its flow graph representation but it cannot differentiate between different kinds of control flow structures. In the CFS calculation cognitive weights of basic control structures are different according to their logical structure.

Property 8. **The measure should differentiate between a sequence of the same construct and a nesting of them or an equivalent construct.**

A measure should be sensitive enough to differentiate between a sequence of the same construct and a nesting of them or an equivalent construct. For example, a nesting of three IF statements is more complex than three sequential IF statements, which is more complex than an equivalent CASE statement, if such an equivalent CASE statement is possible.

The above property is not satisfied by metrics, statement count, effort measures and cyclomatic number. This is because they cannot distinguish between different kinds of control flow structures since the statement count is a measure of size, effort measure is based on the counting the number of operators and operands and cyclomatic complexity depends on the decision structure of a program, In CFS calculation, cognitive weights of the sequence, nesting, branching and case statements are different according to their structure and are more than the sequence structure.

Property 9. **The measure should consider the modular complexity in the following ways:**

(a) The complexity of a program should always be affected by addition, deletion and replacement of a module.

(b) The complexity measure should also reflect the interaction among the modules.

This property is satisfied by the statement count and effort measure but not always by the cyclomatic complexity metric. With the use of addition, subtraction or deletion the cognitive weights and the number of input and outputs change. Therefore, CFS is satisfied by this property.

Table: 2 Desired properties and complexity measures*

	Metrics			
Proper.	Effort Measure	Statement count	Cyclomatic number	CFS
1	yes	Yes	Yes	Yes
2	no	No	Yes	Yes
3	no	No	No	Yes
4	yes	No	No	No
5	yes	Yes	Yes	Yes
6	yes	Yes	Yes	Yes
7	no	No	No	Yes
8	no	No	No	Yes
9	yes	Yes	No	Yes

*) “yes” and “no” represent whether the corresponding property is satisfied or not respectively.

As a result it was found that, to the contrary of other measures, CFS measure satisfies eight properties out of nine. It does not satisfy property four, because CFS calculates the cognitive functional size of code only. This concludes that the proposed properties provide useful information for investigating the robustness of a complexity measure.

3.5 Self evaluation of proposed measure: discussion of pros cons and future work

The drawbacks of a proposed measure are best described by its developer in earlier stages since he/she has better understanding of the background of the proposed measure at that stage. Therefore, in general, the developer of a new complexity measure is expected to validate his/her measure through some validation criteria. After validation, he/she should analyze to see whether the proposed measure is more suitable in comparison to similar measures. It is possible that the proposed measure may be a good indicator of some specific attributes. However, it is also possible that these measures lack in other fields. This argument is the key reason for including self evaluation in the proposed framework. Therefore, after validating the complexity measure, one should not only mention the special features of his/her proposal but also mention its drawbacks.

Furthermore, it is also observed that in most of the investigations there are opportunities for the improvement in the proposed measure, which should be suggested by the developer himself/herself. Therefore, discussion for the possibility of future work is also included in the proposed framework.

When CFS is examined from the self evaluation perspective, some drawbacks are observed. The pros, cons and future work for CFS is suggested as follow. The features of this metric are:

1. It can be used for the complexity of the program and thereof the understandability of the code.
2. It can be used to evaluate the efficiency of the design. A low complexity value gives better design information.
3. It is a language independent complexity metric since it uses cognitive weights and a distinct number of input and output variables**.**
4. The metric is on the ratio scale, a fundamental requirement for a measure from the MT perspective.

Therefore, the proposed metric can be implemented for the calculation of the complexity of the code. However, there are also some drawbacks of the proposed measure, as given below.

1. The present method gives the complexity value in number form, which is generally high for large programs. High complexity values are not desirable.
2. It is difficult to assign the upper and lower boundaries for the complexity values.
3. It is not possible to identify the underlying source of complexity with the proposed measure since it depends on several factors, such as number of input, output and basic control structures.

In the light of experience, the future work for CFS is proposed to include the following:

1. Assignment of the upper and lower boundaries of the complexity values should be investigated.
2. Further analysis is needed for the assessment of complexity.
3. Apart from the preliminary empirical evaluation, more test cases and typical examples (data from the industry) should be applied for the empirical evaluation.

4. Improvement of the proposed metric should be studied for the consideration of the remaining features.
5. Algorithm development to calculate the complexity automatically should be considered.

3.6. Acceptance: After the complete validation process, it is required the acceptance of the metric from the industry. It has been observed in the whole validation process for the example complexity measure, CFS, that is evaluated in almost all perspectives of the proposed framework and satisfy most of the criteria except one, i.e. advanced empirical validation, which is one of the most important component of the proposed framework. The main reason for selecting CFS for applicability of the proposed framework is that, to the best of our knowledge, CFS has not been evaluated through real projects in the industry. In this respect, its practical applicability is still not proved and, therefore, one can conclude that the CFS has not completed the whole validation process.

It is worth mentioning here that, in this paper, the purpose is not to evaluate particular software metric but to demonstrate, applicability of the proposed framework. In this point of view, evaluation of the drawback or the validity of the source of origin of the metrics is not under consideration.

4. A comparison with existing frameworks and models

This section provides the introduction of some of the existing methodologies, frameworks and models which are proposed for evaluation or/and validation of software complexity measurers. A comparison of all these works with the proposed framework is also discussed in the following paragraphs. It is worth mentioning here that we are not including those frameworks/methodologies for comparison which are proposed to cover some specific activities or/and attributes and not directly related to software measure e.g. a framework proposed by Mouchawrab et al. [5] was specifically degined for object-oriented software testability. .

Schneidewind [8] proposed a framework for validating software metrics. This framowork consists of quality factor, quality metrics, validated metrics, quality functions, validitation criteria and metric validation process. In particuler, in **his** framework, validation criteria, which is the major point for comparison with the proposed framework, is based on association, consistency, discriminiative power, tracking, predictability and repeatability, and is close to the IEEE 1061-1998 standards.

When the proposed framework is compared with the methodology proposed for validating software metrics [8], it is observed that the set of properties (such as association, consistency, and discriminiative power etc.) are similar to the properties suggested by IEEE standards [14]. Additionally, Schneidewind [8] suggested applying nonparametric statistical method for metric validation. Both of these (i.e. properties and nonparametric statistical method) are not very practical for software community, because in metric validation there should be a balance between theory and application. This means the theoretical criteria and principles should be developed in such a way that they can be easily adopted by software developers. Software metrics are normally applied by the software developers during different phases and difficult statistical analysis and mathematical formulation makes this process impractical.

Daskalantonakis [38] proposed a practical view of software measurement and implementation experiences. He has given some practical points to observe the utility of the proposed metric. He recommended that the metric should be simple, objective, cost effective and informative. These points are useful and practical for evaluating complexity measures, and therefore included in the proposed framework. On the other hand, in his [38] proposal, the theoretical validation part is totally missing and, without it, one cannot develop the scientific base of a proposed metric.

Cantone and Donzelli [37] proposed Measure Model Life Cycle (MMLC) model for production and maintenance of software measurement models. This model is based on four main faces: Measurement Model (MM) identification, MM creation, MM acceptance, and MM accreditation. This model is partially based on GQM metric and claimed to be an integrated component of management activities pointed to generate, refine and achieve the organizational goal. MMLC is proposed to be a good model for general measurement program in an organisation. However, in its original shape, MMLC is not suitable for software complexity measure. The authors [37] have proposed to set goal oriented solution hypothesis in MM identification phase, which is not very practical for evaluation of software metrics. The proposed framework has already adopted their key points, namely identification, creation, and acceptance. Through the first phases in the proposed framework, one can identify the proper metric for specific purpose by evaluating its objective, identification, scope and proper relation between attribute and metric. Since the proposed framework is an evaluation framework, it also evaluates the proposed metric for its practical usefulness. Furthermore, the creation phase of MMLC is not applicable for the proposed framework. In proposed framework, one gets acceptance of a metric through theoretical and empirical validations, and gets accreditation after its proper implementation in the industry. In other words, on the contrary to the MMLC model, which is a theoretical lengthy model, the proposed model defines simple and straightforward for metric evaluation and validation.

Calero et al. [36] proposed methods for obtaining correct metrics, which is later combined with MMLC model [37] and used in evaluation of metrics for data warehouses [53]. For the theoretical validation metrics, they used different approaches. Firstly, they applied Briand’s framework for evaluating the same set of metrics in a different study [51]. Later, in [50], they used DISTANCE [7] framework, since DISTANCE [7] will guarantee that the metric will be on ratio scale. However, ratio scale is a desirable property but cannot be one of the must properties, since several object oriented metrics are not in ratio scale and Zuse [19] proved that most of the object oriented metrics are not satisfied by ratio scale. As a result, in comparison to Calero et al. [44] and Serrano et al. [50], the first phase (i.e. Practical evaluation) in the proposed framework is simple. It includes all the parameters required for evaluation of a metric in a practical way and is clearly defined in comparison to work of Serrano et al. [50]. For theoretical validation process; the framework follows the representation conditions, a well known concept of measurement, which covers most of the properties desired for software complexity measures from measurement theory point of view.

Poels, and Dedene [7], proposed a framework: DISTANCE, which provides the necessary and sufficient properties for software measures. This is a totally theoretical approach based on measurement theory. It also provides the scale type. This means, the ratio scale is to be obtained when a measure satisfies corresponding properties. On contrary to the proposed framework, it neither considers the practical usefulness of the metric nor proposed a proper way of empirical validation.

Fenton and Kitchenham [9] proposed a framework for software measurement validation. This work is basically used for software metric validation to find answers for “how to validate a measure”; “how to assess the validation work of others”; and “when it is appropriate to apply a measure in a given situation”. They proposed several properties for theoretical and empirical validation. Basically the properties for theoretical validation are accumulative properties from other researcher’s work and based on principals of measurement theory. Although they provided the empirical validation, it gives only the correlation between the measured values of attributes and the values predicted by models. A comparison of the proposed framework with the framework of Fenton and Kitchenham [9] shows that the proposed framework includes all those theoretical properties proposed by them. However, Fenton and Kitchenham’s framework lacks in proposing the real empirical studies for the practical usefulness of the metric.

Briand, Morsaca and Basili [11] proposed property based software engineering measurement, which was a mathematical framework and was based on precise mathematical concepts. They proposed different set of properties for different measurement concepts: size, length, complexity, cohesion, and coupling. These properties are further based on principles of measurement theory and do not discuss on the practical usefulness of the proposed measure.

By comparing with the mathematical framework by Briand et al.[11], It is observed that the proposed framework is more practical since, as discussed in introduction section, It is applicable to any software measure. Briand et al.’s framework [11] also lacks in the proposing how a measure is accepted by practitioners since, by satisfying only theoretical properties, one cannot say that the proposed measure is good one.

Stockman et al. [6] proposed a framework for software quality measurement. This framework consists of a multidimensional concept of quality attributes applied to both product and process. This work is dedicated to the quality issues involved in full software development process. For quality modelling they proposed five steps: process optimization, quality specification, end product quality control, intermediate product quality control and prediction. It is observed that this framework is very generic for quality aspects of the software product and process in comparison to the framework proposed in this article. Software complexity measures are one of the tools to control the quality of product and this framework do not discuss about how to control the quality of a software measure.

Misra [39] has developed an evaluation model for object-oriented (OO) metrics. First they evaluated the existing evaluation criteria for OO metrics, and then presented a four step model. The author claimed that their model cover the most of the features for evaluation of OO metrics. If the proposed framework is compared with the model [39], one can observe that the proposed framework is developed by considering almost all the characteristics of complexity measures and applicable on all types of complexity measures including object oriented measures. On the other hand, the model proposed by Misra [57] was an abstract model and developed specifically for OO metrics.

In his book, Zuse [19] provides an extensive collection of most of the techniques which are applied to software measurement. Zuse [19] has also evaluated most of the available literature and suggested guidelines for software measurement. All the models included in his book and proposed models by Zuse are based on principals of measurement theory. Zuse [19] framework for software measurement also lacks in the practical applicability of software measures. Since the base of his framework was totally mathematical, it does not consider the practical aspects and other features of the measure. On the other hand, the proposed framework considers all these issues, which are important for complete evaluation and validation process.

Table 3. A comparative study of the proposed framework with others

Different frameworks / Criteria		Serrano et all.[53]	Briands et al.[11]	Fenton and Kitchenham [9]	Schneidewind [8]	DISTANCE [7]	Zuse [19]	The proposed framework
Practical Evaluation		Adhoc	no	no	No	no	yes	yes
Theoretical Validation:	Representation Condition	No	yes	yes	No	not exactly	yes	yes
	Scale measurement	Yes	yes	yes	No	yes	yes	yes
Empirical Validation	Small examples/case study	Yes	no	no	No	no	no	yes
	projects from the web	No	no	no	No	no	no	yes
	Real projects from the Industry	Yes	no	no	No	no	no	yes
	Replicated Experiments	Yes	no	no	No	no	no	yes
	Acceptance	Yes	no	no	No	no	no	yes
Self Evaluation discussion		No	yes	no	No	no	no	yes

All the frameworks mentioned in the previous paragraphs have been compared with the proposed framework. A compatibility of other works with the proposed framework is provided in Table 3. In this table only those frameworks are included which are specifically developed for software measures. The Table 3 and the above comparison proved that the proposed framework includes most of the features which are required for evaluation and validation of a software measure however; these all features are independently measured by different frameworks. None of them produced comprehensive measurement criteria for software measures except us.

4.1 Limitation of the work

In their study, Hall and Fenton [55], applied two metric programs in two different companies, and they got one successful and one unsuccessful result. In the detailed analysis of their results, the authors observed that this discrepancy was due to lack of upper management support, resource availability, communication, and adequate feedback from the stakeholders in the program. In an another survey which was performed in more than hundred organisations, Gopal et al. [56] observed that technical and organizational factors plays a major role in success of metric programs. Therefore, it is important to note that a new metric, which is developed based on scientific principles, evaluated both theoretically and empirically in a proper way, may fail due to the organizational behavior [57] and upper management support. This means, the proper evaluation and validation of a new metric by using a framework does not guarantee the successes of that metric. This concludes that the proposed framework only provide the guidelines for proper evaluation and validation of a software metric and provides the necessary but not sufficient conditions for them.

5. Conclusions and future work

In this study, a framework is proposed for evaluating and validating software complexity measures. It is the first attempt to integrate all the important features from different criteria and to present them under a single umbrella. The guidelines in this framework cover most of the issues from all perspectives and will help to improve the quality of the proposed metric. However, although they are essential they are not complete. Further improvement and research is required in this area. The framework has been tested and demonstrated by using the cognitive functional size measure since it has not been evaluated from all perspective, which is required for complete evaluation and validation of a metric. The demonstration of the framework on CFS proves that it can be applied to any software measure. It is easy to implement, straight forward and don't require depth knowledge of the measurement theory. With these qualities, it may be hoped that the proposed framework may be a valuable contribution to the community.

References:

1. Awais, M.M., Shamail, S., Rana, Z.A. : 'Nomenclature unification of software product measures', *IET Software*, 2011, **5,** (1), pp. 83-102.
2. Baski, D., Misra,S.: 'Metrics suite for maintainability of eXtensible Markup language web services', *IET Software*, 2011, **5,** (3), pp. 320-341.
3. Gupta, V., Chhabra, J. K.: 'Package coupling measurement in object-oriented software' *J. of Comp. Sci. & Technology,* 2009, **24**,(2), pp. 273–283
4. Robertas, D., Vytautas, Š.: 'Metrics for evaluation of metaprogram complexity', *J. Com. & Inf. Sci*, 2010, **7,** (4), pp.
5. Mouchawrab, S., Briand L.C., Labiche, Y.: 'A measurement framework for Object-Oriented software testability', *Inf. Soft. Tech*.,2005, **47,** pp. 979-997

6. Stockhome, S.G., Todd, A.R., Robinson, G.A.: 'A framework for software Quality measurement', *IEEE J. Selected Areas in Communications*, 1990, **8**, (2), pp.224-233.
7. Poels, G., Dedene G.: 'Distance-based software measurement: necessary and sufficient properties for software measures', *Inf. Soft. Tech.*, 2000, **42,** (1), pp. 35-46
8. Schneidewind, N.: 'Methodology for validating software metrics', *IEEE Trans. Soft. Eng.*, 1992, **18**, (5), pp. 410-442
9. Kitchenham B., Fenton, N.: 'Towards a framework for software measurement Validation', *IEEE Trans. Soft. Eng.*, 1995, **21,** (12), pp. 929-943.
10. Mendonca, M., Basili, V.: 'Validation of an approach for improving existing measurement frameworks', *IEEE Trans. Soft. Eng.*, 2000, **26,** (6), pp. 484-499
11. Briand, L.C., Moraska, S., Basili, V.R.: 'Property based software engineering measurement', *IEEE Trans. on Soft. Eng.*, 1996, **22,** (1), pp. 68-86
12. Briand, L., Morasca, S., Basili, V. R.: 'An operation process for goal-driven definition of measures', *IEEE Trans. on Soft. Eng., 2002,* **28,** (2), pp. 1106-1125.
13. Basili, V. R., Caldiera, G. Rombach, H. D.: 'The Goal question metric paradigm', Encyclopedia of Software Engineering (Marciniak, J.J., editor), John Wiley & Sons, 1994, Vol.1P. 578-583.
14. IEEE Computer Society: Standard for software Quality Metrics Methodology. Revision IEEE Standard (1998)1061-1998.
15. ISO/IEC, "ISO/IEC 9126-1, Software Engineering- Product quality-part 1: Quality model, 2001.
16. ISO/IEC, "ISO/IEC 9126-2, Software Engineering- Product quality-part 2: External Metrics, 2002.
17. ISO/IEC, "ISO/IEC 9126-3, Software Engineering- Product quality-part 3: Internal Metrics, 2002.
18. ISO/IEC, "ISO/IEC 9126-4, Software Engineering- Product quality-part 4: Quality in Use metrics, 2002.
19. Zuse, H. A.: 'Framework of Software measurement', Walter de Gruyter, Berlin, 1998.
20. Zuse,H.: Properties of software measures, *Software Quality Journal*, 1992, **1**, pp. 225- 260
21. Fenton N. E. , Pfleeger, S. L.: Software Metrics: A Rigorous and Practical Approach, 2nd Edition Revised ed. Boston: PWS Publishing, 1997.
22. Fenton, N.E.: 'When a software measure is not a measure', *Software Engineering Journal*, 1992, pp. 357-362.
23. Morasca, S.: Software Measurement, Handbook of Software Engineering and Knowledge Engineering, 2001, World Scientific Pub. Co. pp. 239-276
24. Weyuker, E.J.: 'Evaluating software complexity measure', *IEEE Trans. on Soft. Eng.,* 1988, **14,** (9), pp.1357-1365
25. Fenton N. E.: 'Software metrics: sucess, failure and new directions', *J.of System and Software*, 1999, **47,** (2-3), pp.149-157.
26. Misra, S.: 'An approach for empirical validation process for software complexity measures', Acta Poletechnica, Hungarica, 2011, **8,** (2), pp. 141-160
27. IEEE Computer Society (1990). IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 610.12 – 1990.

28. Lake A.: ‘Use of factor Analysis to develop OOP software complexity metric’, Proc. Annual Oregon Workshop on software metrics, 1994. pp. 1-15.
29. Pressman, R.S.: Software Engineering: A Practitioner’s approach, Fifth edition, 2001, McGraw Hill .
30. Wang. Y., Shao J.: ‘A new measure of software complexity based on cognitive weights’, Canad. J. of Elec. Comp. Eng., 2003, **28,** (2), pp. 69-74
31. Kaner, C.: Software engineering metrics: what do they measure and how do we know? , Proc. 10th International Software Metrics Symposium, Metrics 2004, pp.1-10.
32. ISO, IS0 15939:2002, Information technology – Software engineering – Software measurement process, International Organization for Standardization, Geneva, 2002.
33. Bégnoche, L., Abran, A., Buglione, L. : ‘A Measurement Approach Integrating ISO 15939, CMMI and ISBSG’, Proc. 4th Soft. Measurement European Forum (SMEF), Rome, 2007. pp.1-19
34. Piattini M., Calero C., Genero M.: ‘Table oriented metrics for relational databases’, *Software Quality Journal*, 2001, **9,** pp. 79–97.
35. Zelkowitz M.V., Wallace, D. R.: ‘Experimental models for validating technology’, *IEEE Computer*, 1998, pp. 23-40.
36. Calero, C., Piattini, M., Genero, M.: ‘Method for obtaining correct metrics’, Proc. 3rd International Conf. on Enterprise and Information Systems (ICEIS’2001), 2001, pp. 779–784.
37. Cantone, G., Donzelli, P.: ‘Production and maintenance of software measurement models’, *Jour. of Soft. Eng. Knowledge Eng.,* 2000, **5,** pp. 605–626
38. Daskalantonakis, M. K.: ‘A practical view of software measurement and implementation experiences within Motorola’ *IEEE Trans. on Soft. Eng.,1992,* **18,** (11), pp.1998-1010
39. Misra, S.: ‘Evaluation criteria for object oriented metrics’, *Acta Poletechnica, Hungarica,* 2011, **8,** (4), (In press).
40. Misra,S., Misra. A. K.: Evaluating cognitive complexity measure with Weyuker properties, Proc. of IEEE ICCI (ICCI2004), 2004, pp.103-108.
41. Misra, S.: ‘Measuring cognitive functional size measure’, *Int. J. of Sof. Sci. and Comp. Intelligence’*, IGI Global Publication, 2009, **1,** (2), pp. 91-100.
42. Misra, S. ‘Cognitive complexity measures: an analysis’ Modern Software Engineering Concepts and Practices: Advanced Approaches, pp.263-279, IGI Global, 2011.
43. Misra, S., Kilic, H.: ‘Measurement theory and validation criteria for software Complexity measure’, *ACM SIGSOFT Soft. Eng. Notes*, 2006, **31, (**6), pp.1-3.
44. Brilliant, S.S., Kinght, J.C.: ‘Empirical research in software engineering’, *ACM SIGSOFT Soft. Eng. Notes,* **24,** (3), 1999, pp. 45–52.
45. Briand L., Emam E. K., Morasca S.: ‘On the application of measurement theory in software engineering’, *J. of Empirical Soft. Eng.*, 1996, **1**, (1), pp. 61-88.
46. Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M.,Jones, P.W., Hoaglin, D.C., El-Emam, K., Rosenberg, J.: ‘Preliminary guidelines for empirical research in software engineering’, *IEEE Trans. on Soft. Eng.,* 2002, **28,** (8), pp. 721-734.

47. Basili , V.: The role of controlled experiments in software engineering research, Empirical Software Engineering Issues, *Lecture Notes in Computer Science*, 2007, **4336**, pp. 33 – 37
48. Zazworka, N., Basili, V. , Zelkowitz , M. V.: 'An environment for conducting families of software engineering experiments', *Advances in Computers*, 2008, **74,** pp. 175-200
49. McCabe. T.H.: "A complexity measure" *IEEE Trans. Soft. Eng., 1976,* 2, (6), pp.308-320
50. Halstead, M.H. : 'Elements of software science', Elsevier North-Holland, New York. 1997
51. Wiener, R.,Pinson, L.J.: Fundamentals of OOP and data structures in java, Cambridge: Cambridge University Press, 2000
52. Misra, S.: 'Evaluation and comparison of cognitive complexity measure', *ACM SIGSOFT Soft. Eng. Notes.*, 2006, **32,** pp. 1-5.
53. Serrano, M., Trujillo, J., Calero, C., Piattini, M.: 'Metrics for data warehouse conceptual models understandability'. *Inf. Soft. Technol., 2007,* **49,** (8), pp. 851-870.
54. Misra, **S. ,** Misra, A. K.: **'**A proposed additional property to the Weyuker's existing properties', Int. J. of Information Technology and Management', 2006, **5**, (1), pp. 66-76
55. Hall, T., Fenton, N. : 'Implementing effective software metrics programs," *IEEE Software*, 1997, pp. 55-65
56. Gopal, A., Mukhopadhyay, T., Krishnan M.S.: 'The impact of institutional forces of software metrics programs', IEEE *Trans. on Soft. Eng.,* 2005, ***31,*** *(*8), pp. 679-694
57. Gopal, A., Mukhopadhyay, T., Krishnan, M.S., Goldenson, D.R.: 'Measurement programs in software development: determinants of success', IEEE *Trans. on Soft. Eng*, 2002, **28,** (9), pp. 863-875.