Evaluation Criteria for Object-oriented Metrics

Sanjay Misra

Department of Computer Engineering Atilim University, Ankara, Turkey
smisra@atilim.edu.tr

Abstract: In this paper an evaluation model for object-oriented (OO) metrics is proposed. We have evaluated the existing evaluation criteria for OO metrics, and based on the observations, a model is proposed which tries to cover most of the features for the evaluation of OO metrics. The model is validated by applying it to existing OO metrics. In contrast to the other existing criteria, the proposed model is simple in implementation and includes the practical and important aspects of evaluation; hence it suitable to evaluate and validate any OO complexity metric.

Keywords: Evaluation Criteria; Measurement; Verification; OO Metrics; Weyuker's Property; language independency; additive nature of metrics

1 Introduction

Several researchers have proposed a variety of criteria [1-11] for evaluation and validation to which a proposed software metric should adhere. Amongst them, we can mention validation through measurement theory [2, 4, 6], IEEE standards [5] Kaner's framework [3], and Weyuker's properties [11]. However, most of the existing evaluation and validation criteria were proposed when procedural languages were dominant. After the adaptation of OO languages by the software industry, not too much effort has been made to develop a model/ framework for evaluating software complexity measures in the OO domain. There are some proposals for OO languages [12-15]; however, most of them cover only specific features of evaluation. For example, Zuse's properties [15] for OO metrics are mathematical in nature and based on principles of measurement theory. The lack of proper guidelines for evaluation and validation of OO metrics motivate us to develop a new evaluation criterion which includes all the features required for evaluation of the OO metrics. For achieving this goal, first we have analyzed the available validation and evaluation criteria, extracted their important features, suggested additions/modifications (if required), then presented them in a formal way. The validity of the proposed model is evaluated by applying eleven different well-known OO metrics. These metrics are described in the next section.

OO metrics are measurement tools to achieve quality in software process and product. However, in general, software measurement has not yet achieved the needed degree of maturity [9] and it needs standardization [16]. Existing proposals, such as Weyuker's properties [11] and the application of measurement theory in software engineering [2, 4, 6, 17, 18], are a topic of discussion [19-23]. We have also worked in the related area of software measurement and presented several papers. We have presented a paper on the usefulness of Weyuker's properties for procedural languages [24]. In another work, we have analysed how Weyuker's properties are used by the developers of three OO metrics [25]. We have previously performed experimentations to analyse the current situation of standard measurement activities in small and medium software companies [26]. We have also performed a study on the situation of the empirical validation of software complexity measures in practice, and we accordingly proposed a model [27]. The applicability of measurement theory on software complexity measures is also investigated in one of our previous works [22]. In the present paper we analyse the present practices used for evaluation and validation of OO metrics, and we accordingly present a model for evaluating OO metrics. We also propose a framework for evaluating software complexity measures but, the present paper is specifically for OO metrics, since OO languages do not share the same features with procedural languages.

Research Problem and Methodology

The literature survey shows that there exist lapses in the measurement process in software engineering. Following this, several researchers have tried to propose a variety of criteria [1-15, 28-30] for different types of measurements in software engineering. OO programming, which is a relatively new programming paradigm in comparison to procedural languages, has received a lot of acceptance from the industry. Several researchers have also proposed software metrics to evaluate its complexity. However, a lack of standard guidelines makes it difficult to propose useful metrics based on a strong theoretical scientific foundation.

We are motivated to present this paper by the following research questions:

Do the exiting criteria for the evaluation of OO metrics evaluate most of the features required for an OO metric (1)?

Should all the features suggested for metrics also be applicable to OO metrics? (2)

To answer these questions, we keep the agenda of the present work as follows;

1 To evaluate the existing criteria which are used for evaluating OO metrics.

2 To extract the important features from the existing criteria for evaluating OO metrics (several well-known metrics are applied on these criteria to extract the features which are useful for OO metrics).

3 To propose a model (based on the evaluation of the existing criteria) which is based on sound scientific principles and which is also easy to adopt by the community.

4 To validate the model by examining it against several well-known and newly proposed metrics.

The initial version of the present work was presented in ICCSA 2009 [31]. In this paper [31], we evaluated each of Weyuker's properties for OO metrics based on experimentation. In addition, we have evaluated all the considered OO metrics against language independency and additive property. In the present work, we have extended our previous work. We are evaluating the applicability of measurement theory on OO metrics. Based on the evaluation of measurement theory, scale measurement, language independency and Weyuker's properties, we propose a model for the evaluation of OO metrics. This model for the evaluation of the OO metrics includes all the required features which are essential for the evaluation and validation of OO metrics.

The remainder of this paper is organized as follows: The evaluation of the applicability of Weyuker's properties and the principals of measurement theory to OO metrics is given in Section 2. A brief analysis of existing important validation criteria is also given in the same section. The proposed model is given in Section 3. The observations are summarised in Section 4. Lastly, the conclusions drawn from the work are summarised.

2 An Analysis of Existing Validation Criteria

Several authors [12-15, 32] have attempted to provide the features of OO metrics and systems. This section provides a brief discussion of some of the important existing evaluation and validation criteria for OO metrics. The rest of the validation criteria are either related to them or specially confined to a single attribute of a software measure. For example, in [14] the authors have emphasized that object-oriented metrics should be evaluated through some quality attributes and have left out other issues (for example practical usefulness) which are also important for the complete validation process.

Weyuker's [11] properties are well-established evaluation criteria for software complexity measures. A good complexity measure should satisfy Weyuker's properties. Although she proposed the properties at the time when procedural languages were dominant, even at present these properties are also valuable to evaluate OO metrics. A significant number of researchers [33-37] have evaluated OO metrics by the complexity properties proposed by Weyuker [11]. For example, Chidamber and Kemerer [33] have applied them for the theoretical evaluation of their OO metrics and, due to the high popularity and acceptance of Chidamber et al.'s metrics, these properties are assumed to be accepted as an evaluation criterion for OO metrics [25]. Consequently, several researchers have used these properties for the evaluation of their object-oriented metrics. Other examples that follow Chidamber et al.'s criteria for theoretical evaluation include a complexity metric based on the different constituents of the components, such as inheritance of classes, methods and attributes proposed by Sharma Kumar & Grover [36], and two OO software design metrics for exception-handling proposed by Aggarwal, Singh, Kaur & Melhotra [35], who have also used Weyuker's properties for theoretical evaluation. We have observed that, in their evaluation, Weyuker's properties have been misunderstood [25].

Measurement theory provides strong rules for the evaluation of software complexity measures. Evaluation through measurement theory proves the scientific basis of the proposed measure. It also proves that the proposed measure actually measures what it is supposed to measure. However, there is a problem in the applicability of measurement theory to software engineering. It is not easy to find a single method that can be accepted by all the software community. Then the question arises: which one is the valid measurement? Some authors put emphasis on representation condition [4], while others emphasized extensive structure [17]. In the next section we also evaluate the applicability of the principals of measurement theory to OO metrics.

Zuse [15] proposed properties of OO software measure based on measurement theory. The validation criteria for complexity measure related to measurement theory states that a valid measure should assume an extensive structure. Zuse himself proved that software measures for OO languages mostly do not assume an extensive structure. Zuse has also given more properties for this purpose, but most of them are not in common use. One of the reasons for the minimal success of these properties is due to fact that these properties are not easy to understand. We will discuss these properties in more detail in Section 2.2.

Linda [14] identified that efficiency, complexity; understandability, reusability, testability and maintainability are five attributes for measuring software quality. An OO metric should measure one or more of these attributes and must evaluate the object-oriented concepts, classes, methods, cohesion, inheritance, polymorphism, number of messages, and coupling. His proposal is concise and useful. We consider Linda's suggestion in our model.

Radu [13] has proposed a new mechanism named detection strategy for increasing the relevance and usability of metrics in OO design by providing a higher-level means for interpreting measurement results. He suggested that four criteria are useful for good design: low coupling, high cohesion, manageable complexity, and proper data abstraction. Radu's proposal is related to the features of OO software development.

We have considered eleven OO complexity metrics. We have applied all these metrics to existing evaluation and validation criteria. We have also applied these metrics to validate our proposed model. We have considered the weighted methods per class (WMC), depth of inheritance tree (DIT), number of children (NOC), coupling between objects (CBO), response for a class (RFC), lack of cohesion in method (LCOM), [33] weighted class complexity (WCC) [37], complexity measures for object-oriented program based on entropy (CMBOE) [34], component complexity (CC) [36], number of catch blocks per Class (NCBC) and exception handling factor (EMF) [35]. The first seven metrics are taken from Chidamber' and Kemerer's paper [33]. These metrics were proposed in 1994 and presently they are the most accepted metrics for the evaluation of OO code. CC was proposed by Sharma, Kumar & Grover [36] in 2007. WCC was proposed by Misra and Akman [37] in 2008. NCBC and EMF were proposed by Aggarwal, Singh, Kaur and Melhotra [35] in 2006. CMBOE [34] is based on the concept of entropy and was proposed in 1995. One common thing amongst all these metrics is that all of them use Weyuker's properties. This is advantageous for us because we are also evaluating the relevance of Weyuker's properties. Further, all the complexity measures/metrics under consideration are available online; we do not describe the metrics further but instead we recommend our readers to follow [33-37], for the details of these metrics.

2.1 Weyuker's Properties and OO Metrics

We have evaluated all the above complexity measures against each of Weyuker's properties. Table 1 has been constructed based on the applicability of Weyuker's properties to different complexity measures. The data for this table is collected from the original papers, where they used Weyuker's properties for the theoretical evaluation of their metrics.

Table 1
Different OO measures and Weyuker's properties

P. N	WMC	DIT	NOC	CBO	RFC	LCOM	WCC	CMBOE	CC	NCBC	EMF
1	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
3	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
5	Y	Y/Y/N	Y	Y	Y	N	Y	N	Y	N	N
6	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y
7	Y	Y	Y	Y	Y	Y	N	N	Y	Not used	Not used
8	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
9	N	N	N	N	N	N	N	Y	Y	Not used	Not used

P. N: property number, Y: Yes (satisfied), N: No (not satisfied)

Misra and Akman [25] have evaluated the applicability of Weyuker's properties to component complexity (CC) [36], number of catch blocks per class (NCBC) and exception handling factor (EMF) [35], and they have demonstrated how these properties have been misunderstood when applying for their measures. In their work [25], the authors have applied Weyuker's properties to these metrics in a concise way. We extend their work and now evaluate the practical applicability of these properties with eleven OO metrics.

Based on the observations of Table 1, we have drawn the following conclusion for the applicability of each Weyuker property to evaluate any OO metric. Weyuker's properties were initially proposed for procedural languages. We present these properties for the OO domain by considering class [25] as a basic unit instead of program bodies.

Property 1: (∃P) (∃Q) ($|P| \neq |Q|$).Where P and Q are the two different classes

This property states that a measure should not rank all classes as equally complex.

From Table 1, it is discovered that all complexity measures satisfy this property. Since not all classes can have the same value for a metric, all measures satisfy this property.

If any measure ranks all classes as equally complex, then it is not easy to say it is an effective measure. Most of the metrics satisfy this property.

The above discussion shows that this property is not very valuable for OO metrics.

Property 2: Let c be a non-negative number, then there are only a finite number of classes and programs of complexity c

This property states that there are only a finite number of classes of the same complexity.

Since the universe of discourse deals with a finite set of applications, each of which has a finite number of classes, this property will be satisfied by any complexity measure at the class level. Here, c is assumed to be the largest possible number, and should be represented as an upper-bound on the size of the program bodies.

As shown in Table 1, Weyuker's second property is satisfied by all complexity measures. Since this property is not capable of differentiating between complexity measures for OO languages, and since any sensible OO measure at class level should satisfy this property, it is also a property that is not useful for the evaluation of OO metrics.

Property 3: There are distinct classes P and Q such that $|P| = |Q|$

This property states that there are multiple classes of the same complexity.

In other words, this property states that even if there exist different classes, the complexity of these classes may be the same. All the complexity measures mentioned satisfy this property as given in Table 1. This is due to the fact that in complexity measures, we can find several classes having the same metric value.

Property 2 and property 3 evaluate the same conclusion for any complexity measure. If there exist multiple classes of equal complexities, it also proves that the number of multiple classes is finite as is represented by property 2. For OO metrics, we can consider this property, since it also reflects property two.

Property 4: $(\exists P)\ (\exists Q)(P \equiv Q\ \&\ |P| \neq |Q|)$

This property states that implementation is important. If there exist classes P and Q such that they produce the same output for the same input. Even if the two classes have same functionality, they are different in the details of implementation. Or, if two programs consisting of many objects have the same functionality, they are different in the details of implementation. This property can be used for the purpose of evaluating OO metrics.

This property is also satisfied by complexity measures given in Table 1. This is due to the fact that even if two classes of design perform the same function, the details of the design matters in determining the metric for the class. The choice of the number of methods is a design decision and independent of the functionality of the class. In view of the discussion, this property is satisfied by all complexity measures.

Property 5: $(\forall P)\ (\forall Q)\ (|P| \leq |P; Q|\ \&\ |Q| \leq |P; Q|)$

This property states that, if the combined class is constructed by class P and class Q, the value of the class complexity for the combined class is larger than the value of the class complexity for class P or class Q.

If we evaluate the complexity measures given in Table 1 against this property, we find that this property is not satisfied by LCOM, DIT, CMBOE, NCBC, and EMF.

The failure of this property by LCOM is because the number of non-empty intersections will exceed the number of empty intersections [33]. DIT tree also does not satisfy this property for any special case, when for any two classes P and Q are combined, and when one is the child of the other [33]. CMBOE also does not satisfy this property [34]. For some cases of NCBC and EMF, they also do not satisfy this property. The method the developer of NCBC and EMF suggested, given that this property is not satisfied by any of the measures, was not appropriate [25]; it is possible that this property is not satisfied by these measures.

As a conclusion, this property is worth considering for the purpose of evaluating OO metrics.

Property 6a: $(\exists P)\ (\exists Q)(\exists R)(|P| = |Q|)\ \&\ |P;R| \neq |Q; R|)$

6b: $(\exists P)\ (\exists Q)\ (\exists R)\ (|P| = |Q|).\ \&\ |R; P| \neq |R; Q|)$

This property shows the non-equivalence of interaction. Also this is a contextual property; it states that, if a new class is appended to two classes which have the same class complexity, the class complexities of the two newly combined classes are different, or the interaction between P and R can be different than the interaction between Q and R, resulting in different complexity values for P+ R and Q + R. This is also an important property, so we may use it as one of the evaluating criteria for OO complexity measures.

From Table 1, it is found that all complexity measure satisfy this property except WCC. All the measures satisfy this property because the interaction between the two classes, P and R, can be different from that between Q and R, resulting in different complexity values. For WCC, the cognitive weights of the methods and the number of attributes are fixed for any program. Joining programs *R* with *P* and *Q* adds the same amount of complexity; hence, property 6 is not satisfied by this measure.

Property 7: There are program bodies P and Q such that Q is formed by permuting the order of the statements of P, and ($|P| \neq |Q|$)

This property states that permutation is significant. It means that the permutation of the elements within the item being measured can change the metric values. The purpose is to ensure that the metric values change due to the permutation of classes. According to Chidamber and Kemerer [33], in the case of OO programming in any class or object, changing the order in which the methods or attributes are declared does not affect the order in which they are executed. Moreover, they argue that this property is meaningful in traditional programming languages but not for OO programming. In this respect, all Chidamber et al.'s metrics are satisfied by this property. For NCBC and EMF, the authors have also applied Chidamber et al.'s statement and do not apply this property for their measure. However, we do not agree with their statements. In the original Chidamber et al. paper [33], the authors argued that this property is not useful in calculating the complexity of a class because the order of statements within the class is not important in calculating the complexity. However, in our opinion, if the class complexity is calculated by adding the method complexities (since the order of statements are important in this case) it is a challenge to neglect this property. However, one should be careful when using this property. This is also the reason why WCC does not satisfy this property. Although, Chidamber and Kemerer [33] suggest that it is not a useful property for OO metrics, from our point of view it is a useful measure for evaluating OO metrics.

Property 8: If P is renaming of Q, then $|P| = |Q|$

This property requires that when the name of the class or object changes it will not affect the complexity of the class. Even if the name of the member function or member data in the class changes, the class complexity should remain unchanged. All complexity measures in Table 1 satisfy this property. Since the complexity is not affected by renaming, it is not a meaningful property for any OO metric.

Property 9: (∃P) (∃Q) (│ P │+│ Q│).< (│ P; Q│)

This property states that interaction increases complexity. This property states that the class complexity of a new class combined from two classes is greater than the sum of the complexities of its components. In other words, when two classes are combined, the interaction between the classes can increase the complexity metric value. Due to this reason, most of the complexity measures do not satisfy this property. Again, the developers of NCBC and EMF [35] have not made use of this property by saying that this property is not applicable for OO measure, as stated by Chidamber et al. [33]. Again, we do not support their statement; no such type of indication in the original paper of Chidamber et al. [33] has been found, and it has been argued that this property may not be an essential feature of OO software design. It appears only as a suggestion and they themselves have used this property for evaluating all their measures. Further, the usefulness of this property for OO metrics has been discussed and proved by several researchers, and it remains a topic of research. [19-21]. The author of the present paper has also proposed a modification in this property [24]. This proposal was basically for procedural languages, but we check its applicability to OO measures in this paper.

Conclusion of the Evaluation of Weyuker's properties: Weyuker's first, second, third, fourth, sixth and eighth properties are satisfied by all complexity measures. It is because of the fact that most of the properties are general in nature and hence can be satisfied by any complexity measure. Furthermore, the observation shows that only six properties [25] are useful for OO metrics. However, in these properties, there is no orientation towards OO languages and metrics. Most of the important properties, such as property 5, 6, and 9, are based on the principles of measurement theory. Once we evaluate the proposed metric with the principles of measurement theory, it is automatically satisfied by Weyuker's properties. Additionally, we have observed that Weyuker's properties do not address language independency [24], which a complexity measure should account for; i.e. a complexity measure should be language-independent. This property has been proved as an essential feature for procedural language. Let us evaluate this property for its applicability to object-oriented metrics.

2.2 Evaluation of the Applicability of Measurement Theory to OO Metrics

The evaluation of complexity measures via measurement theory is established by several researchers [2, 6-10, 15, 17, 18]. However, proper applicability of these measures to OO metrics is not evaluated properly except by a few [15]. If we review the literature to find the relation between evaluation criteria for software complexity metrics and measurement theory, we can find three important proposals: Briand, Morasca, and Basili's [2] 'property based software engineering measurement', Kitchenham, Pfleeger, & Fenton's [6] proposal on 'towards a framework for software measurement validation' and Zuse's proposal for software measurement [15, 17]. Although many other proposals based on measurement theory are available in the literature, most of them are either related to these three, in developing stages [8] or not used by the software community [7]. In the following paragraph, we will see how effective these different theoretical evaluation criteria are in evaluating OO measures.

Amongst available validation criteria, the framework given by Briand, Morasca and Basili [2] is used by several researchers [38-40]. Briand, Morasca and Basili [2] have proposed measurement properties of various measurement concepts, such as size, length, complexity, cohesion and coupling. The authors have tried to make the differences in these different measurement concepts by proposing a set of properties. For calculating the complexity of a system, they proposed five properties, which include nonnegative (Complexity property 1), null value (Complexity property 2), symmetry (Complexity property 3), module monotonicity (Complexity property 4) and disjoint module additive (Complexity property 5). The first property states that the complexity of a system should not be a negative value. The second property states that if a system has no elements, then its complexity value should be zero. The third property states that the complexity of a system should not depend on the convention chosen to represent the relationships between its elements. The forth property states that complexity of a system should be no less than the sum of the complexities of any two of its modules with no relationships in common. The fifth property states that the complexity of a system composed of two disjoint modules is equal to the sum of the complexities of the two modules. The fourth and fifth properties are related to the additive nature of the metric. Furthermore, by satisfying all these properties (Complexity.1 – Complexity.5), the measure will be on the ratio scale. It is important to note that the first three properties are common in other measures also, i.e. for length and size. In summary, by satisfying these properties by complexity measures, it proves the ratio scale and additive nature of the measures.

The second method to evaluate complexity measures is through representation conditions [4, 6]. To satisfy the representation conditions, initially there should be an empirical relation system (ERS), numerical relation systems (NRS), and a complexity metric, which is defined as the mapping between ERS to NRS. Furthermore, a measure/metrics must be satisfied by the two conditions called representation conditions. The first part of the representation condition says that any empirical observation should be measurable and any measurement result should be empirically observable. The second part says that the complexity of the whole should be definable in terms of the complexities of its parts. Again, the conclusion of the representation condition states that the measure should be additive.

Zuse [15] has introduced properties of OO software measures. This is the only work which especially emphasizes the properties of OO metrics. In this work, Zuse has proved that OO metrics do not assume an extensive structure and hence he has proposed some new measurement principles for OO metrics. He used terms such as the Dempster-Shafer Function of Belief, the Kolmogoroff axioms and DeFinetti axioms. However, it is a common observation that these theories for OOM evaluation have neither been applied nor have they been used by most of the developers of OO metrics.

The scale measurement is also an important issue for evaluating software complexity metrics and proved by several researchers. The third evaluation method which is normally applied to the complexity measures is to investigate the scale of the measure [17]. In fact, it is not easy to separate scale measurement with the first two methods, but there are also different ways to achieve the scale. There are two ways for scale measurement in software complexity measurement: admissible transformation and extensive structure [17]. It is assumed that a complexity measure should be on ratio scale. We have included the scale measurement in our model.

Conclusion for the Applicability of Measurement Theory: In all of the above measurement criteria applied for evaluation of software complexity measurements, all of them recommend that the measure should be additive, hence on ratio scale. The way of achieving this goal may be different; for example through admissible transformation, extensive structure or representation condition, but the goal of all aforementioned criteria is the same, i.e. to achieve ratio scale.

By keeping this issue in mind, in the following paragraphs we evaluate the additive nature of the existing object-oriented measures. Based on the evaluation, we will discuss the applicability and relevance of measurement theory to OO measures.

2.3 Additive Property and Language Independency of OO Measures

Based on the observation in Sections 2.1 and 2.2, we observed two important points:

1) From measurement theory perspective, all of the different criteria suggest that a measure should be additive. The additive property is not directly addressed by Weyuker's.
2) In addition, Weyuker does not discuss anything regarding the language independency of a proposed measure.

Following the above mentioned points, we evaluate the applicability of additive property and language independency to OO metrics.

a) Additive Property

The additive nature of the complexity measure is proved as one of the desirable properties from measurement theory perspective. Misra [24] has presented the additive property in mathematical form. In fact this property was suggested by several researchers and in [24], the author has suggested modifying Weyuker's property 9. First, we explain this property [24] and later we check all the considered OO complexity metrics against this property.

Property 1: $(\forall P)(\forall Q)(|P; Q|) \geq |P| + |Q|$).

This property states that the complexity of a program ($|P; Q|$) composed of two components cannot be less than the sum of the complexities of its component bodies.

In fact, this property is not a new one but rather a modified version of Weyuker property 9. Weyuker herself rejected this property, arguing that the effort needed to implement or understand the composition of a program body P with itself is probably not twice as much as the effort needed for P alone. However, it seems reasonable that the complexity of a program body can be related to the complexity of all of its parts. That is to say, it is necessary requirement for any complexity measure. The additive nature is also related to the scale of the measure [41]; that is to say, if any measure is additive, then it is also assumed to be on ratio scale.

b) Language Independency Property

Misra has [24] observed that Weyuker's properties do not address language independency, which a complexity measure should account for; i.e. a complexity measure should be language-independent. The author has presented this property as;

Property 2: $(\exists P)(\exists Q)(\exists R)(P \equiv Q \equiv R \,\&\, |P| = |Q| = |R|)$; Where P, Q, and R are the classes written in different languages

This property states that if P, Q, and R are the classes for the same algorithm in a different programming language, then the complexity should be the same. In other words, this property states that a measure should rank the complexity of the same algorithm in a different language as equally complex.

Now the applicability of these properties is checked against the OO metrics under consideration.

c) Applicability of language Independency and Additive Properties to OO Metrics

Weighted method per class (WMC). If WMC is calculated only by counting the number of methods, (In [33] the authors have suggested taking the weight of each method as 1 unit.) the language independent property is satisfied by this measure. This is because the numbers of methods for classes in different languages are assumed to be the same. In this respect, this property is satisfied by this measure. On the other hand, if we calculate the complexity of the each method independently, by any procedural metric, it depends on the characteristics of the applied procedural metric.

For the additive nature of WMC, if we combine two classes, it is possible that the number of methods in the combined class may reduce in comparison to the sum of the methods of independent classes because there may be some methods in both classes. As a result, WMC is not an additive measure.

Depth of inheritance tree (DIT) is satisfied by the language independent property because the tree structure for the same problem is assumed to be same for all object-oriented languages.

DIT is not an additive measure because when we combine the two classes, the DIT value of the resultant class will not be the sum of the DIT values of the independent classes. It is because of the fact that when two OO programs with multiple hierarchies combine with each other, they combine in parallel way. This is to say, if DIT value of two OO programs are 3 and 2, then it is not equal to 5 for the combined class.

Number of children (NOC) also satisfies the language-independent property. A similar argument for DIT is applicable to NOC.

NOC does not satisfy the additive property due to similar arguments to those given above (for DIT).

Coupling between object (CBO) also satisfies the language-independent property because it depends on the number of messages through which the classes are coupled.

The numbers of message calls will not change by a change in the language.

CBO is the measure which depends on the number of message calls to the other classes. If we combine two classes, in which one of them has a message call for the other class, then naturally, after combining these classes, there is no coupling in the combined class. In this respect, CBO also does not satisfy the additive property.

Response for the Class (RFC) is defined as the total number of methods that can be executed in response to a message to a class. This count includes all the methods available in the class hierarchy. Since class hierarchy is the same for all languages, RFC satisfies the language-independent property.

RFC also does not satisfy the additive property because after the combination of two classes, the total number of methods that can be executed in response to a message to a class will not be the sum of the total number of called messages of two independent classes.

Lack of cohesion in method (LCOM), which is related to the counting of methods using common attributes, does not satisfy the language-independent property. This is because the use of attributes inside the methods depends on the programming language.

LCOM does not satisfy the additive property. This is because the use of attributes inside the methods depends on the programming language and on the fact that, when we combine two classes, the number of methods using the common attributes of the independent class will not be the sum of the combined class.

Complexity measure for OO program based on entropy (CMBOE) is a measure based on entropy. The entropy of an OO program is computed on a per-class basis dependent on the name strings, methods, messages, and their architecture. For a given problem, the different OO languages may have different representation of the classes and its contents, and therefore the entropy of the same problem in different OO languages may be different. From this point of view, CMBOE does not satisfy the language independency.

CMBOE is not satisfied by the additive property. In the original paper [34] the authors have proved that the complexity of the combined class is greater than the sum of the complexities of independent ones.

Component complexity (CC) is based on classes, methods, attributes and interfaces. The authors [36] have used the coefficients for classes, methods and attributes, which are dependent on the nature of the component-this means on the nature of programming language. As a result, CC does not satisfy language-independent properties.

CC does not satisfy the additive property because it is based on classes, methods, attributes, and interfaces. Once we combine the two classes, and if there are some common methods and attributes in both, then the complexity of the combined class reduces.

Number of Catch Blocks per Class (NCBC) is defined as the ratio of catch block in a class (the sum of the catch blocks of each method) to the total number of possible catch blocks in a class [35]. Since NCBC depends on the internal architecture of the method, which varies from language to language, NCBC does not satisfy the language independency.

NCBC does not satisfy the additive property because it depends on the sum of the catch blocks of each method, and methods may be common in both classes, which reduces the complexity of the combined class.

Weighted Class Complexity (WCC) depends on the attributes and the internal architecture of the methods, which are not same for all programming languages. From this point of view, WCC is not a language-independent complexity measure.

WCC does not satisfy the additive property. It depends on the attributes and the internal architecture of the methods. After combining the two classes, their number may reduce due to the common methods and attributes.

Exception-Handling Factor (EHF) is defined as the ratio of the number of exception classes to the total number of possible exception classes in software. EHF depends on the classes, which should be the same for all programming languages. As a result, EHF satisfies the language independency.

EHF does not satisfy via the additive property because the number of exception classes is the count of the exceptions covered in a system [35]. If we combine the two classes, the EMF values do not combine with each other due to the nature of the metric; EMF is also not an additive measure.

The applicability of language independency and the additive property by different complexity measures are summarized in Table 2.

Table 2

Language independency and additive properties of OO languages

Properties ---------------------- Metrics	Language Independency	Additive Nature
WMC	Y	N
DIT	Y	N
NOC	Y	N
CBO	Y	N
RFC	Y	N
LCOM	N	N
CMBOE	N	N
CC	N	N
NCBC	N	N
WCC	N	N
EMF	Y	N

From Table [2], we can easily observe that none of the OO metrics are additive in nature, which is not compatible with the principles of measurement theory. All the three proposals in measurement theory strongly suggest that a measure should be additive in nature [2, 6 and 17]. However, our experiments show the negative results and prove that OO measures do not satisfy the additive property. For the language independencies of OO metrics, we have found a mixed response. Six out of eleven OO metrics are language independent measures. Also language independency is a reasonable requirement for any metric hence OO metrics.

Based on the evaluation measurement theory, scale measurement, language independency, additive property, and Weyuker's properties, we propose our model in next section.

3 Proposed Evaluation Criteria/ Model

The proposed validation and evaluation criteria have the following four stages:

1. Identifying the basic requirement for proposing OO measure
2. Evaluation through the measurement theory and Scale measurements.

3 Empirical Validation

4 Identifications of Thresholds and limitations

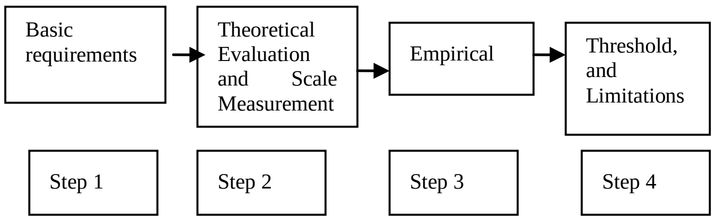
Basic requirements
Theoretical Evaluation and Scale Measurement
Empirical
Threshold, and Limitations
Step 1
Step 2
Step 3
Step 4

Figure 1
Proposed model for evaluation of OO metrics

3.1 Identifying the Basic Requirement for OO Measure

Based on the properties suggested by different scientists, the following set of simpler and essential properties is summarized, against which an OO measure/metric should be evaluated For this purpose, studies by Kitchenham, Pfleeger, & Fenton's [6], Linda [14] and Morasca [18] are used to extract these features and present them in a formal way.

1) Identifying the OO features under investigation.

- The first step in proposing an OO metric is to decide what to measure. It is important that a metric or measure evaluates one of the following features of OO: methods, classes, cohesion, coupling, inheritance, information hiding, and polymorphism.
- An OO metric should focus on an internal object structure.
- An OO metric should measure the efficiency of an algorithm and the use of machine resources.
- An OO metric should evaluate the psychological measures that affect a programmer's ability to create, comprehend, modify, and maintain software.

All of our metrics under consideration are specific to OO features. For example, WMC is related to the internal architecture of method and class, DIT with inheritance, NOC also with inheritance, CBO with coupling, RFC with class, LCOM with cohesion, CC, NCBC WCC,EHF and CMBOE with class.

2) Identifying the quality factors/attributes

The attribute(s) under investigation by the proposed OO metric should be identified in the beginning. If the metric or suite of metrics is capable of evaluating more than one of the attributes (also called quality factors), then they should be priorities. Some of the attributes are:

Performance

Reliability

Maintainability

Portability

Efficiency

Understandability

Reusability

Testability

A number of developers of complexity measures do not care for this part. Although one can guess the attributes under investigation from their original papers, in most papers this part is not clearly defined. For example, in the original proposal of NCBC and EHF, nowhere in the entire paper is it discussed for which attributes they are proposing metrics. They stated that two metrics were developed to measure the amount of robustness included in the code. It can be guessed that NCBC and EHF are representative of maintainability. The attributes measured by other metrics are: maintainability by WMC, DIT, NOC, CBO, RFC, LCOM and CMBOE, and understandability, maintainability and reliability by WCC.

3) A basis for the proposal of a new metric should be developed

The foundation for the metric development should be built. It should include the literature survey, motivations, comparison and quality references which will prove the worth and need for a new proposal. Based on discussions of related work published, it should be clearly explained why it is important and how much the new metric will add to the field. Furthermore, a clear-cut proposal for the system should be developed, and the way, method or instrument by which it is to be measured should also be identified. Also the relationship between attribute and metric should be determined.

Although most of the metrics under consideration have been clearly defined in their original papers, one can easily find a number of metrics in literature that were developed without a clear-cut purpose and aim.

3.2 Theoretical Evaluation and Scale Measurement

A metric and the prediction system are acceptable only when their usefulness has been proved by a theoretical validation process; otherwise it may mislead its users. In Section 2.2, we observed that the conclusion of most of the validation criteria based on measurement theory is related to the additive property of the measure.

However, most of the OO metrics under experimentation are not additive in nature. This result forces us to rethink the applicability of measurement theory to OO metrics. According to our point of view, we should not neglect the principles of measurement theory; instead, we should concentrate on some basic requirements and important features (from the measurement theory perspective) which are required to form the scientific basis of the metric. The next important issue in terms of measurement theory is the scale measurement of the proposed measure. Keeping this issue in mind, we propose simple and important features required by measurement theory and the investigation of the scale measurement.

From the measurement theory point of view, there should be an entity, a property and measurement mapping. The measurement mapping and rules are called metric [18].

1) Entity: An entity is an object or an event in reality. For example, the entity for all the OO metrics under consideration is class.

2) Property: The property of the class which is under investigation is different for different measures. For WMC, CC and the WCC the property is the complexity, for DIT, NOC, RFC and CBO the property is coupling, for LCOM the property is cohesion. The property for NCBC and EMF is complexity.

3) Metric: Metrics should be defined by a function(s) which assigns a value to the attribute. All the metrics under consideration are properly defined in their original papers. Although some of them, such as NCBC and EHF, do not map their values to attributes, their definitions are clear.

4) Attributes: An attribute is a feature or property of an entity. Furthermore, the attributes can be classified as internal and external attributes [42].

4.1) Internal Attributes: Internal attributes are those which can be measured purely in terms of the product, process or resources [42]. Since the entity for all the metrics under consideration is class, then we can examine the attributes which are related to class.

The internal attributes for the class may be the size in terms of the number of lines of code, the number of methods, the number of attributes/variables, coupling in terms of the number of message calls to other classes, and cohesion in terms of the number of common attributes used in different methods of the same class. The internal attributes for the considered OO metrics are summarized in Table 3.

4.2) The external attributes: External attributes are those which relate the product, process or resources with the external environment. For example, for object-oriented measures, the external attributes are reliability, maintainability, usability, and understandability. The external attributes of all the metrics under consideration are also summarized in Table 3.

Evaluation of Measure based on Scale

After the identifying entity, attributes and defining the metric, one must investigate the type of scale. Normally, there are five different types of scales: absolute, ratio, interval, ordinal and nominal. The definitions of all these scales are given in brief [17]:

1 Nominal Scale: any one to one

2 Ordinal scale: g: strictly increasing functions

3 Interval scale: g(x)= ax+b, a> 0

4 Ratio Scale: g(x)= ax, a> 0

5 Absolute Scale: g(x) = x.

These scales are classified according to admissible transformations [17].

The scale of most of the complexity measures under consideration are not evaluated in their original paper, except WCC. The scale for CK metric suites were evaluated in [43, 44]. The possible scale for all the complexity metrics are summarized in Table 3.

It is worth mentioning here that most of the OO metrics are not additive in nature. This is also because they do not satisfy the extensive structure [17]. In other words, most of the OO metrics are either on an interval or ordinal scale. The scale of all the OO measures under investigation is given in Table 3.

Table 3

Attributes and type of scale for Different measures

	Metrics ------------------ Attributes For entity class	WMC	DIT	NOC	CBO	RFC	LCOM	WCC	CMBOE	CC	NCBC	EMF
Internal Attributes:	Size	N	Y	Y	N	N	N	Y	N	Y	N	N
	reuse	N	Y	Y	Y	N	Y	Y	N	N	N	N
	coupling	N	Y	Y	Y	Y	N	Y	N	Y	N	N
	cohesiveness	N	N	N	N	N	Y	Y	N	N	N	N
	Functionality	Y	N	N	Y	N	N	N	N	Y	Y	
External Attributes:	Maintainability	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	reliability	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	usability	N	Y	Y	Y	N	Y	Y	N	N	N	N
	Understandability	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Scale		Ns	Ns	Is	Is	0s	Is	Ns	Is	Is	Ns	Is

Y: Yes, N: No, In last row, Ns: Nominal Scale, Is: Interval Scale, Os: ordinary scale

3.3 Empirical Validation

Several researchers [45-48] have pointed out the importance of empirical validation for software metrics. It is the way through which the academician and scientist can assist industry in selecting new technology. On the other hand, it is a common observation that the standard of empirical validations for software metrics is not up to the required level [45]. There is no match in content between the increased level of metrics activity in academia and industry [49]. This is because of the fact that there is no direct link between researchers and industry, and this makes it difficult to validate the metrics against real projects. As a consequence, researchers try to validate their metrics through other means, such as experimentations in labs, classrooms or the available data/programs from the internet. Most of the time, they fulfill only partial empirical validation. However, for complete empirical validation, one must apply the new technology/metric to real data/projects from industry.

Considering the above pitfalls, we have suggested the application of empirical validation in two stages [27]. The first stage of empirical validation includes the initial validation of the metrics by applying them to different test cases and examples. In the second stage of the empirical validation, the new metric is tested by using real projects from the industry. The details of all the steps of the empirical validation process are discussed in [27]. In the following paragraphs we provide them in abstract form.

The first stage of the empirical validation includes the case study, applying it to a small project available on the web or in literature. In this stage, the proposed metrics can be validated through experimentations in labs, classrooms or the available data/programs from the internet. In parallel, similar metrics should also be applied to all these different ways of validation. This will be helpful for comparative study and finally in proving the usefulness of the proposed metrics.

The complexity measures under consideration, WMC, CC, WCC, DIT, NOC, RFC, CBO, LCOM, NCBC and EMF, have undergone these initial phases validation. One should bear in mind that the initial validation is not a guarantee of success or of the usefulness of the proposed metric without the second stage of validation.

The second stage of the empirical validation proves the actual applicability of the proposed metric. This stage of validation is a must, and it evaluates the validity of the metric by applying it to a real project from industry. The difference between the first and second stage is that, in first stage, the examples/case studies may be small in size and collected from literature and web; in the second stage, the examples/case studies are real data from industry. As in the first stage, similar metrics should also be applied to the real project(s), for comparative purposes. In fact, the second stage of empirical validation is the real proof for a new measure.

We recommend the first stage only in the cases when the data from the industry is not immediately available.

For the metrics under examination, with the exception of the CK metric suites, none of the metrics have been applied to real projects from industry. This is also the reason that none of them, again with the exception of the CK metric suites, are very popular and accepted in industry.

3.4 Thresholds for the Metrics should be Identified

After preliminary and advanced empirical validations, the thresholds for the metrics should be developed. Although it is possible to propose the thresholds for a new metric before empirical validations, empirical validations can change the threshold values. This is because of the fact that, after the analysis of the results obtained from a real project, the developer can change their thresholds. Furthermore, the initial proposal only gives the basic idea of the proposed measure, which may fail in real life applications. This is one of the reasons for the lack of acceptance of the majority of OO metrics from the industry which are available in the literature.

The importance of thresholds is discussed by several researchers. Lorenz and Kidd defined threshold as [50] "heuristic values used to set ranges of desirable and undesirable metric values for measured software. These thresholds are used to identify anomalies, which may or may not be an actual problem." Henderson-Sellers [51] states the importance of thresholds as, "An alarm would occur whenever the value of a specific internal metric exceeded some predetermined threshold." In fact, threshold values are the best indicator for the rating of the complexity values for an OO system. For example, in WMC measurement, if the number of methods for a class exceeds 100 (the weight of each method is assumed to be 1), then this class automatically becomes more error-prone and less understandable, which also increases the maintenance efforts. Also, the importance of thresholds is supported by cognitive theory [52, 53]. The authors in [52, 53] use a human memory model and suggest that more complex classes will overflow short term memory, which results in more errors. Contrary to these results, some of the authors presented some experimental results which show that there is no impact of threshold values on the fault tolerance. There is a continuous relationship between measures and fault tolerance and errors [28]. However, in our opinion, for any new model or theory, contradictory cases exist. Threshold values are only indicators and act as an alarm which tells you that over this limit there is a high chance of errors. It is possible that one can build a system whose complexity values cross the threshold values and is nevertheless error free.

If we evaluate our metrics under consideration, we observe that most developers do not propose thresholds values. In particular, for the CK metric suite, the authors gave some hints of these numbers but did not clearly define the threshold values.

For example, they observed that the maximum values of DIT were 10 or less. This value was observed based on the empirical validation study. Later, due to the high popularity and acceptance of Chidamber et al.'s metrics, the thresholds were investigated by other authors [54]. The threshold for WMC is 100, the inheritance nesting level (another form of DIT and NOC) is 6 [50]), for CBO it is 5, and for RFC 100. For other metrics, WCC, NCBC, EMF and CMBOE, no threshold values were investigated. If no threshold is defined, how can one guess with numbers whether these numbers are either a good or bad predictor of complexity?

Further, there exist limitations and boundaries in a new proposal. It is not easy for a single metric to evaluate all the aspects/attributes of code. From our point of view, the limitations of new measurers can best be described by the developers. Some of the examples include: Are the metrics applicable to the design phase or also applicable to the testing phases? For example, most of the metrics in the CK metrics suite are fit for the design phase; however, WCC is fit for both the design and testing phases. WCC can be applicable in the design phase to reduce the class complexity by limiting the number of complex methods; and in the testing phases it can be applied to reduce bugs. Another example of the limitation is: Can one evaluate the complexity only by simple calculations, or does it require software, and if this is the case, is it then available? If not, the chances for practical use of the proposed metric immediately decrease. It can be easily observed in a number of metrics that they have proved their worth for small codes and examples, but it is not easy for them to fit in the real environment of software developments, where codes are quite large and distributed in different classes and in different files. Also the developer should provide the range of values which gives an indication of the different levels of quality attributes.

4 Observations

We observe the following points in this study:

1. There are no models/frameworks/proposals which state clear-cut guidelines for the properties of object-oriented measures.
2. We have proved that the existing criteria, such as Weyuker's properties and measurement theory, are as such not fit for evaluating OO metrics.
3. It is clear from Table 1 that Weyuker's first, second, third, fourth, sixth and eighth properties are satisfied by all given complexity measures. Weyuker's first property states that no complexity measure can rank all classes as equally complex. Since the universe of discourse deals with a finite set of applications, each of which has at most a finite number of classes, property two will be satisfied by any complexity measure at the class level. Weyuker's second and third properties give the same conclusion.

Weyuker’s eighth property is concerned with the name of the class, which does not affect the complexity of any class. In conclusion, Weyuker’s first, second, and eighth properties are found to be not useful for the evaluation of complexity measures for OO programming. Other properties are compatible with measurement principles.

4 Weyuker’s properties number 3, 5, 6, 7 and 9 are found useful for evaluating OO metrics. On the other hand, all these properties are compatible with measurement principles. If we evaluate our measure through the fundamentals of measurement theory, then we have no need to apply Weyuker’s properties. This is the reason that we have included only evaluation via measurement theory, and not via Weyuker’s properties.

5 All of the different criteria based on measurement theory recommend that a measure (in general) should be additive; it hence should be on a ratio scale. Weyuker’s modified property nine [24] is also a representation of the additive nature of a measure.

6 None of the OO metrics under consideration are found to be additive in nature.

7 No complexity metrics under consideration are on a ratio scale according to measurement theory principles.

8 Further, the existing validation criteria/properties for OO metrics based on measurement theory by Zuse [15], are difficult to understand and hence not easy to apply to OO metrics. The theory requires a sound knowledge of mathematics. This is the reason that most of the proposed OO metrics do not follow these properties.

9 Other measurement criteria, such as representation condition, also do not provide too much information (such as ratio scale and additive nature) for OO measures.

All of these observations indicate that theoretical evaluation of an OO metric through the representation condition [4, 42], extensive structure [15, 17], and complexity property [2] are not effective for evaluating OO metrics. In this respect, the fundamental properties and definitions required by measurement theory should only be the necessary condition for OO metrics, which we summarized in Section 3.1. Furthermore, in the case of software engineering, empirical validation is more important than theoretical validation, and if a metric is properly validated empirically, via data from industry, and evaluated through the given fundamental definition from measurement theory, it proves the worth of the measure. It is also worth mentioning that, although empirical validation is the most important part of the validation process, it does not mean that theoretical validation should be ignored. Theoretical validation proves that the metric is developed according to certain rules and regulations and based on principles of measurement theory.

Conclusion and Future Work

The necessity of evaluation and validation criteria for object-oriented metrics is clear. However, in assessing the existing evaluation criteria, we have observed that most of them consider only specific features for the evaluation of a metric, and, especially, they are not proposed in keeping with the special features of OO metrics. For example, Weyuker's properties only cover the mathematical features of programs (for procedural languages) and do not evaluate any practical aspects of the metric. So Weyuker's properties are not suitable criteria for the theoretical evaluation if applied independently. Further, measurement theory also includes most of the features of Weyuker's properties; so if a measure is evaluated via measurement theory, then Weyuker's properties can be avoided. On the other hand, additive nature and ratio scale are two main requirements for a measure from a measurement theory point of view; however, both are rejected by the majority of OO metrics. This is a constraint in the application of the principles of measurement theory to OO metrics. Further, the original measurement principles proposed by Zuse [9] are difficult to understand. Additionally, the empirical validation process is also not clearly mentioned in the literature. All these issues indicate a need for a unified model, which should be simple to apply, and which should cover the majority of the features required for the evaluation and validation of OO metrics. The presented model is an attempt to achieve this goal in this area. We kept all these issues in our mind before constructing our model. We have proposed a simple four-step model against which a software complexity measure for OO metric should be evaluated. Our first step is to prepare the basis of the proposal. The second step is related to theoretical validation, which includes the principles of measurement theory in a simple way. These first two steps form the scientific basis for proposing a new OO metric. Our third step is related to empirical validation, which is proposed in two steps. The final step is to provide thresholds for the proposed metrics based on real observations, which is intended to provide valuable information regarding the actual analysis of metric values. In fact, it is not easy to achieve completeness through independent existing evaluation criterion. This became a motivation for us to propose a unified model. We hope that our attempt will make a valuable contribution to practitioners and as well to academicians who have the intention of proposing a new metric in the OO environment.

References

[1] Fenton N. (1993) New Software Quality Metrics Methodology Standards Fills Measurement Needs', IEEE Computer, April, pp. 105-106

[2] Briand L. C., Morasca S., Basili V. R. (1996) Property-based Software Engineering Measurement, IEEE Transactions on Software Engineering, 22(1), pp. 68-86

[3] Kaner C. (2004) Software Engineering Metrics: What do They Measure and How Do We Know?’ In Proc. 10th Int. Software Metrics Symposium, Metrics, pp. 1-10

[4] Fenton N. (1994) Software Measurement: A Necessary Scientific Basis’, IEEE Transactions on Software Engineering, 20(3), pp. 199-206

[5] IEEE Computer Society (1998) Standard for Software Quality Metrics Methodology. Revision IEEE Standard, pp. 1061-1998

[6] Kitchenham B., Pfleeger S. L., Fenton N. (1995) Towards a Framework for Software Measurement Validation. IEEE Transactions on Software Engineering, 21(12), pp. 929-943

[7] Morasca S. (2003) Foundations of a Weak Measurement-Theoretic Approach to Software Measurement. Lecturer Notes in Computer Science LNCS 2621, pp. 200-215

[8] Wang Y. (2003) The Measurement Theory for Software Engineering. In Proc. Canadian Conference on Electrical and Computer Engineering, pp. 1321-1324

[9] Zuse H. (1991): Software Complexity Measures and Methods, Walter de Gruyter, Berline

[10] Zuse, H. (1992) Properties of Software Measures. Software Quality Journal, 1, pp. 225- 260

[11] Weyuker, E. J. (1988) Evaluating software complexity measure. IEEE Transaction on Software Complexity Measure, 14(9) pp. 1357-1365

[12] Marinescu, R. (2005) Measurement and Quality in Object –oriented design, In Proceedings 21st IEEE International Conference on Software Maintenance, pp. 701-704

[13] Reißing R. (2001) Towards a Model for Object-oriented Design Measurement, Proceedings of International ECOOP Workshop on Quantitative Approaches in Object-oriented Software Engineering, pp. 71-84

[14] Rosenberg L. H. (1995) Software Quality Metrics for OO System environment. Technical report, SATC-TR-1001, NASA

[15] Zuse, H (1996) Foundations of Object-oriented Software Measures. In Proceedings of the 3rd International Symposium on Software Metrics: From Measurement to Empirical Results (METRICS '96) IEEE Computer Society, Washington, DC, USA, pp. 75-84

[16] Misra S. (2010) An Analysis of Weyuker’s Properties and Measurement Theory, Proc. Indian National Science Academy, 76(2), pp. 55-66

[17] Zuse, H. (1998) A Framework of Software Measurement, Walter de Gruyter, Berline

[18] Morasca S (2001) Software Measurement, Handbook of Software Engineering and Knowledge Engineering, 2001, World Scientific Pub. Co. pp. 239-276

[19] Gursaran, Ray G. (2001) On the Applicability of Weyuker Property Nine to OO Structural Inheritance Complexity Metrics, IEEE Trans. Software Eng., 27(4) pp. 361-364

[20] Sharma N., Joshi P., Joshi R. K. (2006) Applicability of Weyuker's Property 9 to OO Metrics" IEEE Transactions on Software Engineering, 32(3) pp. 209-211

[21] Zhang L., Xie, D. (2002) Comments on 'On the Applicability of Weyuker Property Nine to OO Structural Inheritance Complexity Metrics. IEEE Trans. Software Eng., 28(5) pp. 526-527

[22] Misra S., Kilic, H. (2006) Measurement Theory and validation Criteria for Software Complexity Measure, ACM SIGSOFT Software Engineering Notes, 31(6), pp. 1-3

[23] Poels G., Dedene G. (1997) Comments on Property-based Software Engineering Measurement: Refining the Additivity Properties, IEEE Trans. Softw. Eng. 23(3) pp. 190-195

[24] Misra. S. (2006) Modified Weyuker's Properties, In Proceedings of IEEE ICCI 2006, Bejing, China, pp. 242-247

[25] Misra S., Akman I. (2008) Applicability of Weyuker's Properties on OO Metrics: Some Misunderstandings, Journal of Computer and Information Sciences, 5(1) pp. 17-24

[26] Tolga O. P., Misra S. (2011) Software Measurement Activities in Small and Medium Enterprises: An Empirical Assessment', In press, Acta Polytechnica, Hungarica, 4

[27] Misra S. (2011) An Approach for Empirical Validation Process of Software Complexity Measures, In press, Acta Polytechnica, Hungarica. Issue 4

[28] Benlarbi S., Emam K. E., Goel N., Rai S. (2000) Thresholds for Object-oriented Measures, In Proc. 11th International Symposium on Software Reliability Engineering (ISSRE'00) p. 24

[29] Rojas T., Perez M. A., Mendoza L. E., Mejias A. (2002) Quality Evaluation Framework:The Venezuelan case, In Proc. AMCIS 2002. Dallas, USA. 3-5 August

[30] Jacquet J. P., Abran A. (1999) Metrics Validation Proposals: A Structured Analysis, Dumke, R., and Abran, A. (eds.):Software Measurement, Gabler, Wiesbaden, pp. 43-60

[31] Misra. S. (2009) Weyuker's Properties, Language Independency and OO Metrics, In Proceedings of ICCSA 2009, Lecture Notes in Computer Science, Volume 5593/2009, pp. 70-81

[32] Lincke R. (2006) Validation of a Standard- and Metric-Based Software Quality Model. In Proceedings of 10th Quantative approach in OO software engineering (QAOOSE), pp. 81-90

[33] Chidamber S. R, Kemerer C. F. (1994) A Metric Suite for OO Design, IEEE Transactions on Software Engineering, 20(6) pp. 476-493

[34] Kim K., Shin Y., Chisu W. (1995) Complexity Measures for Object-oriented Program Based on the Entropy, In Proc. Asia Pacific Software Engineering Conference, pp. 127-136

[35] Aggarwal K. K., Singh Y., Kaur A., Melhotra, R. (2006) Software Design Metrics for OO Software, Journal of Object Technology, 6(1), pp. 121-138

[36] Sharma A., Kumar, R, Grover, P. S. (2007) Empirical Evaluation and Critical Review of Complexity metrics for Software Components, In Proceedings of the 6th WSEAS Int. Con. on SE, Parallel and and disributed systems, pp. 24-29

[37] Misra S., Akman I. (2008) Weighted Class Complexity: A Measure of Complexity for OO Systems, Journal of Information Science and Engineering, 24(5), pp. 1689-1708

[38] Costagliola G., Tortora G. (2005) Class points: An Approach for The Size Estimation of Object-oriented Systems, IEEE Transactions on Software Engineering, 31, pp. 52-74

[39] Chhabra J. K., Gupta V. (2009) Evaluation of Object-oriented Spatial Complexity Measures. SIGSOFT Softw. Eng. Notes, 34(3), pp. 1-5

[40] Chhabra J. K., Gupta V. (2009) Package Coupling Measurement in Object-oriented Software, Journal of Computer Science and Technology, 24(2), pp. 273-283

[41] Stevens, S. S. (1946) On the Theory of Scale of Measurement Science, 103, pp. 677-680

[42] Fenton, N. (1997) Software Metrics: A Rigorous and Practical Approach', PWS Publishing Company, Boston, MA, 1997

[43] Neal R. D., Weis R. (2004) The Assignment of Scale to Object-oriented Software Measures Technical report, NASA-TR-97-004

[44] Kanmani S, Sankaranarayanan V, Thambidurai P, (2005) Evaluation of Object-oriented Metrics, IE(I) Journal-CP, 86(November), pp. 60-64

[45] Kitchenham B. A., Pfleeger S. L., Pickard L. M., Jones P. W., Hoaglin D. C., El-Emam K., Rosenberg J. (2002) Preliminary Guidelines for Empirical

Research in Software Engineering, IEEE Transaction on Software Engineering, 28(8), pp. 721-734

[46] Singer J., Vinson N. G. (2002) Ethical Isuue in Empirical Studies of Software Engineering, IEEE Transaction on Software Engineering, Vol. 28, 12, pp. 1171-1180

[47] Brilliant S. S., Kinght J. C. (1999) Empirical Research in Software Engineering, ACM Sigsoft. 24(3), pp. 45-52

[48] Zelkowitz M. V., Wallace D. R. (1998) Experimental Models for Validating Technology, IEEE Computer, May, pp. 23-40

[49] Fenton N. E. (1999) Software Metrics: Sucess, Failure and New Directions, J. of System and Software, 47(2-3) pp. 149-157

[50] Lorenz M., Kidd J. (1994) Object-oriented Software Metrics. Prentice-Hall

[51] Henderson-Sellers, B. (1996) Object-oriented Metrics:Measures of Complexity. Prentice-Hall

[52] Hatton L. (1997) Re-examining the Fault Density-Component Size Connection, IEEE Software, pp. 89-97

[53] Hatton L. (1998) Does 00 Sync with How We Think?, IEEE Software, pp. May/June, pp. 46-54

[54] Rosenberg L., Stapko R., Gallo A (1999) Object-oriented Metrics for Reliability, In Proc. of IEEE International Symposium on Software Metrics, 1999