SOFTWARE METRICS VALIDATION METHODOLOGIES IN SOFTWARE ENGINEERING

K.P. Srinivasan[1] and T. Devi[2]

[1]Associate Professor in Computer Science,C.B.M. College, Kovaipudur, Coimbatore – 641 042, India
[2]Professor and Head, Department of Computer Applications,School of Computer Science and Engineering,Bharathiar University, Coimbatore – 641 046, India

ABSTRACT

In the software measurement validations, assessing the validation of software metrics in software engineering is a very difficult task due to lack of theoretical methodology and empirical methodology [41, 44, 45]. During recent years, there have been a number of researchers addressing the issue of validating software metrics. At present, software metrics are validated theoretically using properties of measures. Further, software measurement plays an important role in understanding and controlling software development practices and products. The major requirement in software measurement is that the measures must represent accurately those attributes they purport to quantify and validation is critical to the success of software measurement. Normally, validation is a collection of analysis and testing activities across the full life cycle and complements the efforts of other quality engineering functions and validation is a critical task in any engineering project. Further, validation objective is to discover defects in a system and assess whether or not the system is useful and usable in operational situation. In the case of software engineering, validation is one of the software engineering disciplines that help build quality into software. The major objective of software validation process is to determine that the software performs its intended functions correctly and provides information about its quality and reliability. This paper discusses the validation methodology, techniques and different properties of measures that are used for software metrics validation. In most cases, theoretical and empirical validations are conducted for software metrics validations in software engineering [1-50].

KEYWORDS

Result Based Software Metrics (RBSM), Software Metrics Validations, Theoretical Validations, Empirical Validations, Software Measurement, Object-Oriented Metrics, Software Engineering

1. INTRODUCTION

The software metrics is an essential research subject in software engineering [1-50]. The software metrics researchers proposing a new metric have the trouble of proof to show that the metric is adequate for measuring the software [41, 44]. This is provided through the process of software metrics validation. Over the last four decades, however, researchers have debated what constitutes a "valid" metric. The debate over what constitutes valid metric centres on software metrics validation criteria [10, 24, 47]. According to eminent researcher Jones, C., "Better measures and better metrics are the stepping stones to software engineering excellence" [22]. Recently,

Srinivasan, K.P., and Devi, T., proposed a set of six Result Based Software Metrics (RBSM) suites called "Comprehensive Metrics" for measuring object-oriented design quality effectiveness [44, 45, 46]. And further, they also introduced a new kind of software metrics for software coding measurement in software engineering metrics called "Program Keyword Metrics (PKM) (RBSM)" [41]. This program keyword metrics eliminates the ambiguity criticism of most referred "Halstead Metrics" and "Lines Of Coding (LOC) metrics" in software engineering. Further, they eliminated the main criticism of "accuracy on results" in software quality measurement by "Program Keyword Metrics" in software engineering [41]. Today, the software metrics gives the appearance of being more influenced by "object-oriented metrics [31, 33, 44, 45]" and "software metrics validations [3, 34, 35]". At present, many researchers are involved in proposing software metrics for software process and product measurement [12, 13, 25, 40, 44, 45]. The most important events of software metrics show that in the past many metrics had been proposed and validated by eminent researchers. This paper discusses the concepts related to theoretical and empirical software metrics validation methodologies.

Basili, V.R., and Weiss, D.M., proposed a methodology for collecting valid software engineering data and an effective data collection method for evaluating software development [9]. In the year 1985, Shen,V.Y., Yu, T.Y., Thebaut,M., and Paulsen.L.R., conducted an empirical study for identifying error-prone software. They studied and analyzed three program products and their error histories and they discovered that simple metrics related to the amount of data and the structural complexity of programs may be useful in identifying at an early stage those modules most likely to contain errors [39]. Software metrics are computed for the purpose of evaluating certain characteristics of the software developed. Li, H.F., and Cheung, W.K., conducted an empirical study of software metrics in 1987. They studied 31 metrics, including a new hybrid metric, and applied them to a database of 255 programs. It is also concluded that many volume metrics have similar performance while some control metrics surprisingly correlate well with typical volume metrics in the test samples used [27]. Cherniavsky, J.C., and Smith, C.H., studied Weyuker's axioms for software complexity measure and properties for software complexity measures. It is shown that a collection of properties suggested by Weyuker are inadequate for determining the quality of a software complexity measure. They concluded that Weyuker is not proposing the properties as axioms**,** but only as rules of thumb for evaluating complexity measures [11].

Schneidewind, N.F., proposed the metrics validation methodology that has six validity criteria. The proposed validation methodology supports the quality function for assessment, control, and prediction. The Schneidewind, N.F. validation criteria for software measures are: association, consistency, discrimination, power tracking, predictability and repeatability. Further, it shows that nonparametric statistical methods play an important role in evaluating metrics against the validity criteria [37]. Alshayeb, M., and Li, W., conducted an empirical validation of object-oriented metrics in two different iterative software processes and found that object-oriented metrics are reasonably effective in predicting evolution changes of a system design in the short-cycled process, but they are largely ineffective in the long-cycled framework process. Further, they concluded that predictive capability of object-oriented metrics in the short-cycled iterative process is only effective when the design accumulates to a certain critical mass, often toward the last several iterations in the process and this study provided the evidence on the abilities and limitations of object-oriented metrics in predicting maintenance effort [2]. Bandi, R.K., Vaishnavi, V.K., and Turk, D.E., conducted an empirical study for predicting maintenance performance using object-oriented design complexity metrics. Researchers agreed that maintenance may turn out to be easier for object-oriented systems. The approach to controlling software maintenance costs is the utilization of software metrics during the development phase and to help identify potential problem areas. They conducted empirical validation for three

existing design complexity metrics and it is used to assess their ability to predict maintenance time. The empirically validated three metrics are interaction level, interface size, and operation argument complexity. The experiment conducted on the effect of design complexity on maintenance time and each of the three metrics by itself was found to be useful in the experiment in predicting maintenance performance [8]. Deligiannis, I., Shepperd, M., Roumeliotis, M., and Stamelos, I., conducted an empirical investigation of an object-oriented design heuristic for maintainability. They investigated the impact of a design heuristic on the maintainability of object-oriented designs and the relationship between that object-oriented design heuristic and metrics [15]. Badri, L., and Badri, M., conducted an empirical study for new class cohesion metrics [6]. Badri L, Badri, and Gueye, A.B., conducted an empirical investigation on several systems. Class cohesion is considered as one of the most important object-oriented software attributes. They introduced a new cohesion criterion based on common objects parameters. They analyzed the real systems to validate class cohesion assessment by exploring cohesion empirically [7]. Kaur, A., Singh, S., Kahlon, K.S., and Sandhu, P.S., conducted empirical analysis of CK and MOOD metrics suite. The empirical validations of these metrics in real world software development setting are limited. Various flaws and inconsistencies have been observed in the suite of six class-based metrics [23]. Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements or not and validation should establish confidence that the software is fit for its purpose and does what the user really requires [16]. Section 2 explains different types of validations used in software metrics and Section 3 discusses the Weyuker's properties of measures. Further, Section 4 discusses the Kitchenham's properties of measures. Section 5 explains Briand's properties of measures and conclusion includes future directions of the research.

2. THEORETICAL AND EMPIRICAL VALIDATION METHODOLOGIES IN SOFTWARE ENGINEERING

In the software measurement, each metric must be validated for its validity. There are two types of validations for validating software metrics: "theoretical" and "empirical" validations. These two types of software metrics validations are given in Figure 1.

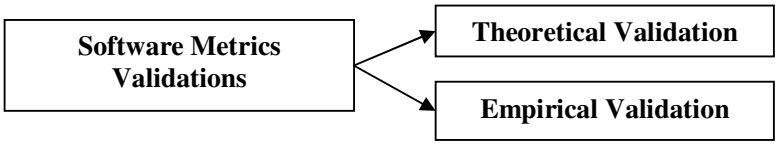

Figure 1. Software Metrics Validation Methodologies in Software Engineering

The structural measurement model defines validations for software measurement. There are two basic methods of validity of measures that are given in measurement models. The theoretical validation confirms that the measurement does not violate any necessary properties of the elements of measurement. The empirical validation confirms that measured values of attributes are consistent with values predicted by models involving the attribute. The theoretical methods of validation allow valid measurements with respect to certain defined criteria and empirical methods are corroborating evidence of validity or invalidity [24, 30, 42].

These two types of validation methods are arrived under internal and external validations. The internal validation is a theoretical methodology that ensures that the metric is a proper numerical characterization of the property it claims to measure. Demonstrating that a metric measures what

it purports to measure is a form of theoretical validation. The external validation methodology involves empirical study of software metrics. Further, internal and external validations are commonly referred to as "theoretical and empirical validations". Both types of validation are necessary for software metrics in order to prove their metric validity. Theoretical validation requires the properties for the software metrics validations. The theoretical validation is based on the analysis of the properties of the attribute to be measured. The theoretical validation provides information about mathematical and statistical operations that can be performed with the measure, which is essential when working with the measure. There are two main approaches followed in theoretical validation and they are representational theory of measurement and property based approaches (Figure 2).The main goal of theoretical validation is to assess whether a metric actually measures what it purports to measure and theoretical validation of metrics establishes their construct validity, i.e., it proves that they are valid measures for the constructs that are used as variables in the study. There is not yet a standard, accepted way of theoretically validating software metric. The representational theory-based approach is explained by Kitchenham, B., Pfleeger, S.L., and Fenton, N., properties of measures [24]. The property-based approach, called as axiomatic approach is proposed by Weyuker, E.J. [3, 14, 47] and and Briand, L.C., Morasca, S., Basili, V.R. [10, 35].

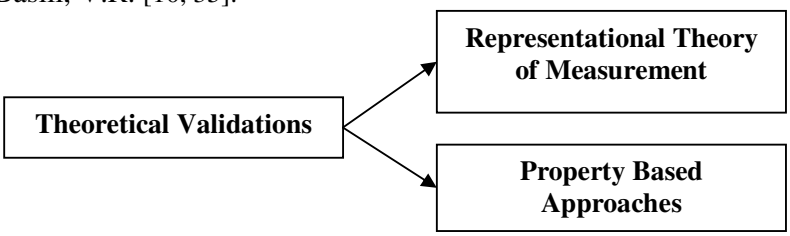

Figure 2. Theoretical Validations Techniques in Software Measurement

The representational theory of measurement is based on a homomorphism between the empirical world and the numerical world and the representation condition, which attests that the properties of the attributes in the real world should be maintained by the measures in the numerical world. This implies the definition of an empirical and a numerical world and the construction of a mapping between the two. This kind of validations is called internal validation of measures for assessment and theoretical validation. The property-based theoretical approaches are used with a number of axioms for theoretical validations and this category of theoretical validation is also called "axiomatic, analytical, and algebraic validation". This paper discusses three types of properties of measures that are shown in Figure 3.

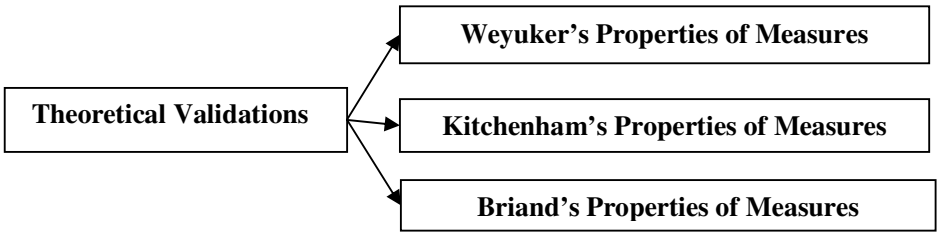

Figure 3. Theoretical Properties of Measures in Metrics Validations

The set of axioms are defined for software attribute. Weyuker, E.J., [47] proposed nine properties for evaluating syntactic software measures and they are discussed in Section 3. These properties are used on the applicability of object- oriented measures and properties alone are necessary to prove the validity of measure. Briand, L.C., et al. [10] describe properties of measures for size,

complexity, length, coupling, and cohesion. They state that the representation condition is an obvious prerequisite to measure validation. A broader approach to theoretical validation is taken by Kitchenham, B., et al. [24].

The procedure to follow for experimental validation varies significantly depending on the purpose of the measures, i.e., evaluation or prediction, the type, and the amount of data collected. One measure can be used to predict the value of another measure [17, 24]. Currently proposed metrics are validated using theoretical and empirical validations for software metrics and measures are validated for validity. The three types of empirical validations are surveys, case studies and experiments (Figure 4).

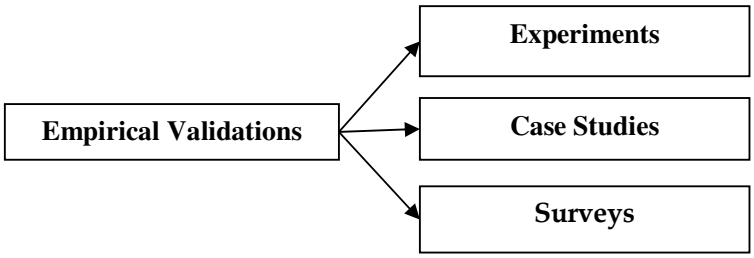

Figure 4. Empirical Validation Techniques in Software Engineering

According to properties of measures, not all the measures satisfy validation properties. And the measures that do not satisfy the properties can safely be excluded. However, there is evidence for a validated measure used in software metrics theoretically and empirically. The proof of the validation is that hundreds of measures have been defined and validated in the past. Many of the defined metrics are proved using both theoretical and empirical validations and they are useful for empirical evidence of software [48]. The following section explains the most referred and used software metrics validation methodology proposed by Weyuker's properties of measures.

3. WEYUKER'S PROPERTIES OF MEASURES IN TRADITIONAL AND OBJECT-ORIENTED METRICS VALIDATIONS

This section explains the validation properties of measure proposed by the eminent researcher Weyuker, E.J. It is most referred properties of measure and most of the object-oriented metrics are theoretically validated using these properties [1, 3, 14, 34, 35]. The evaluation is performed by identifying a set of desirable properties that measures should possess and determining whether or not the prospective metric exhibits those properties [47].

The popular and often used Weyuker's nine properties are shown in the Table 1. The properties are: **Property 1 Noncoarseness:** Given a class P and a metric μ another class Q can always be found such that μ (P) ≠ μ (Q). This implies that not every class can have the same value for a metric; otherwise it loses its value as a measurement. **Property 2 Granularity:** There should be a finite number of cases having the same metric value. Since the universe of discourse deals with almost a finite set of applications, each of which has a finite number of classes, this property will be met by any metric measured at the class level. **Property 3 Nonuniqueness (Notion of Equivalence):** There can exist distinct classes P and Q such that μ (P) = μ (Q). This implies that two classes can have the same metric value, i.e., the two classes are equally complex. **Property 4 Design Details are Important:** Given two class designs, P and Q, which provide the same functionality, does not imply that μ (P) = μ (Q). The specifics of the class must influence

the metric value. The intuition behind property 4 is that even though two class designs perform same functions, the details of design matter in determining the metric for the class. **Property 5 Monotonicity:** For all classes P and Q the following must hold: μ (P) ≤ □□μ (P+Q) and μ (Q) ≤ □□μ (P+Q), where P+Q implies combination of P and Q. This implies that the metric for the combination of two classes can never be less than the metric for either of component classes. **Property 6 Nonequivalence of Interaction:** For classes P, Q, R, μ (P) = μ (Q) does not imply that μ (P+R) = μ (Q+R). This suggests that interaction between P and R can be different from the interaction between Q and R resulting in different complexity values for P + R and Q + R. **Property 7 Permutation:** A measure is sensitive to the permutation of classes. This property requires that permutation of elements within the item being measured can change the metric value. This property is meaningful in traditional program design. **Property 8 Renaming Property:** If a class P is a renaming of a class Q, then μ (P) = μ (Q). The renaming property requires that when the name of measured entity changes, the metric should remain unchanged. **Property 9 Interaction Increases Complexity:** □ P and Q are classes such that: μ (P) + μ (Q) < μ (P+Q). The principle behind the property is that when two classes are combined, the interaction between classes can increase the complexity metric value.

Table 1. Weyuker's Properties of Measures

Property 1	**Noncoarseness**
Property 2	**Granularity**
Property 3	**Nonuniqueness (Notion of Equivalence)**
Property 4	**Design Details are Important**
Property 5	**Monotonicity**
Property 6	**Nonequivalence of Interaction**
Property 7	**Permutation**
Property 8	**Renaming Property**
Property 9	**Interaction Increases Complexity**

Weyuker's second property "granularity" and eighth property "renaming property" are applicable to all object-oriented metrics. The seventh property "permutation" is only for traditional programming and not required for object-oriented programming.

The eminent researchers Chidamber, R., and Kemerer, C.F. validated object-oriented design metrics using Weyuker's properties of measure [14]. Archer, C., and Stinson, M. applied Weyuker's properties on Halsted metrics and McCabe's metrics. They suggested that Weyuker's properties can be used as a basis for selecting software metrics properties [4]. Weyuker's properties have been widely used and they are well established compared to other validation criteria. It is a formal approach and serves as an important measure to evaluate metrics [36]. Aggarwal, K.K., et al. analytically evaluated a set of design metrics proposed for object-oriented software using Weyuker's set of all axioms [1]. Anbumani, K., and Srinivasan, K.P., evaluated their metrics using Weyuker's properties of measures [3]. Sharma, A, et al. evaluated their complexity metrics using Weyuker's properties of measures [38]. Misra, S., and Akman, I. conducted the application of Weyuker's properties of measures on object-oriented metrics [29]. Malik, N., and Chillar, R.S. proposed the new design metrics for complexity and validated metrics using Weyuker's properties of measures [28]. At present, Weyuker's properties are

mainly used for software metrics validation by various researchers and these properties are very useful for object-oriented design metrics validations. The following section discusses another significant validation methodology called Kitchenham properties of measure.

4. KITCHENHAM'S PROPERTIES OF MEASURES IN TRADITIONAL AND OBJECT-ORIENTED METRICS VALIDATION TECHNIQUES

The eminent researchers Kitchenham, B., Pfleeger, S.L., and Fenton, N., have proposed properties for validating software metrics and they explained how to validate metrics for their validity of measure [21, 24]. They proposed a basic assumption for measure for validation on satisfying the two conditions: Measure must not violate and necessary properties and each model used in the measurement process must be valid (Figure 5).

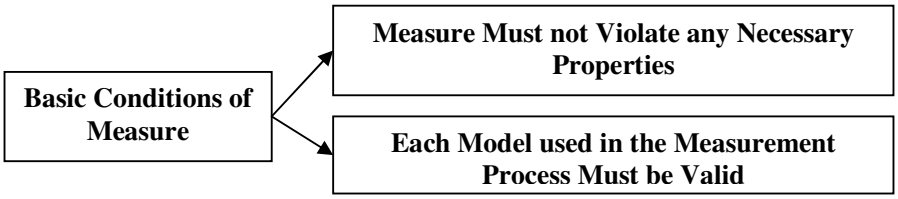

Figure 5. Basic Conditions of Measure

Formally, Kitchenham, B., et al. defined valid measure as it cannot prove a theory but can only falsify it. So, a valid measure is one that it cannot be invalidated. In order to decide the measurement validity, it is necessary to confirm different types of measurement validity proposed by Kitchenham, B., et al. and they are illustrated in Table 2. The types of software measurement validity are attribute, unit, instrument and protocol validity [24, 49]. ① **Attribute validity -** The attribute is actually exhibited by the entity. Attribute validity must be considered both for directly measurable attributes and for indirectly measureable attributes that are derived from other attributes; ②**Unit validity -** The measurement unit being used is an appropriate means of measuring the attribute;③ **Instrument validity -** Any model underlying a measuring instrument has to be valid and the measuring instrument is properly calibrate; and ④**Protocol validity -** An acceptable measurement protocol is adopted.

Table 2. Measurement Validity

Validity	Measurement
Validity 1	Attribute
Validity 2	Unit
Validity 3	Instrument
Validity 4	Protocol

Kitchenham,B., et al. defined properties for measure called as "Kitchenham's properties of measures" and stated that the software metrics should exhibit the properties. The Kitchenham's properties for metrics are shown in Figure 6. ① **Property 1**: For an attribute to be measurable, it must allow different entities to be distinguished from one another; ② **Property 2:** A valid measure must obey the representation condition, that is, it must preserve the intuitive notions about the attribute and the way in which it distinguishes different entities; ③ **Property 3:** Each

unit of an attribute contributing to a valid measure is equivalent; and ④ **Property 4:** Different entities can have the same attribute value within the limits of measurement error.

The four properties of Figure 6 are necessary properties for direct measures, but they are not sufficient for indirect measures. For indirect metrics, they proposed another four additional properties and they are shown in Figure 7. ⑤**Property 5:** The metric should be based on an explicitly model of the relationship between certain attributes; ⑥**Property 6:** The model must be dimensionally consistent; ⑦**Property 7:** The metric must not exhibit any unexpected discontinuities; and ⑧**Property 8:** The metric must use units and scale types correctly.

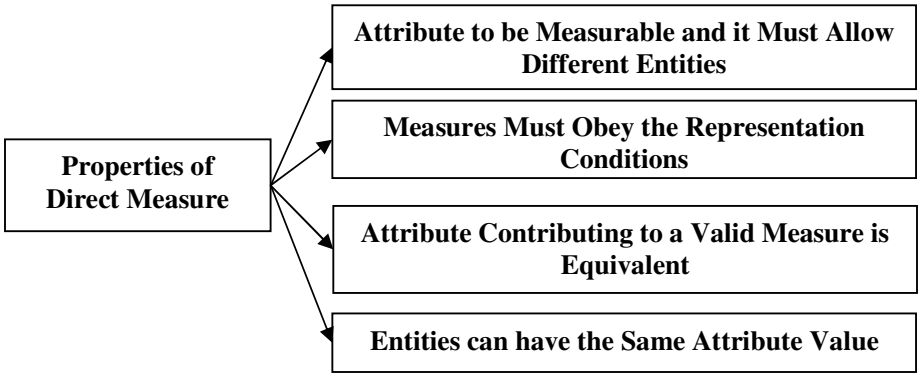

Figure 6. Properties of Direct Measure

The properties of measures of Kitchenham, B., et al. were used by Harrison et al. for validating MOOD metrics [21]. Kitchenham, B., et al. agrees with property 1 of Weyuker and they rejected the properties 5, 6 and 9 of Weyuker because they imply scale type. They concluded that property 7 is inappropriate as it contradicts with standard measurement scale and property 8 as unnecessary, due to representation conditions. Weyuker's properties 1, 7 and 8 are related with properties of measures of Kitchenham, B., et al. Weyuker's properties 2, 3 and 4 are not considered by Kitchenham, B., et al.

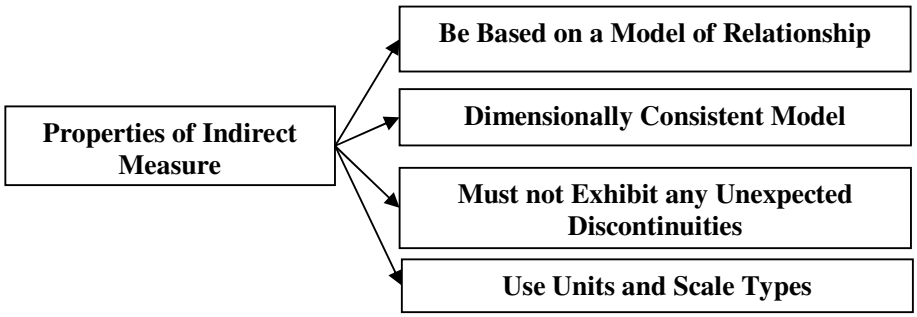

Figure 7. Properties of Indirect Measure

A valid measure must obey the representation condition, that is, it must preserve the intuitive notions about the attribute and the way in which it distinguishes different entities. Weyuker's eighth property is related to this issue. It has accepted the representative condition and Kitchenham's properties do not require Weyuker's eighth property. Weyuker's property 7 relates to this issue and in fact, it asserts the converse of this assumption by claiming that program complexity should be responsive to the order of statements in a program. Weyuker's fourth

property states that programs that deliver the same functionality can have different complexity values. This property asserts that function does not prescribe form. Thus, it seems to be an assertion about complexity rather than a property of a complexity measurement. The following section discusses other important properties of measures proposed by the eminent researchers Briand, L.C., Morasca, S., Basili, V.R.

5. BRIAND'S PROPERTIES AND METHODS IN SOFTWARE METRICS VALIDATION TECHNIQUES

In software measurement, defining properties of measure is an important task and it requires skill for defining properties for software metrics. Briand, L.C., Morasca, S., Basili, V.R. have proposed property-based approach for software measurement [10, 35]. They proposed a set of mathematical properties and formalized important internal software attributes for "size, length, complexity, cohesion, and coupling" measurement. Their framework is based on a graph-theoretic model of a software metrics, which is seen as a set of elements linked by relationships. The properties provided by them can be applied to any software object produced along the software process. Their framework includes definitions of systems, modules and modular systems and operations between the elements [10, 35, 50]. Their properties of measure for internal attributes are shown in Figure 8. The properties proposed by Weyuker in the Section 3 and Kitchenham in the Section 4 are syntactically based properties of measure and their properties are "generic". Their properties do not characterize specific measurement concepts but properties of measures of Briand et al. are only for size, length, complexity, cohesion and coupling (Figure 8).

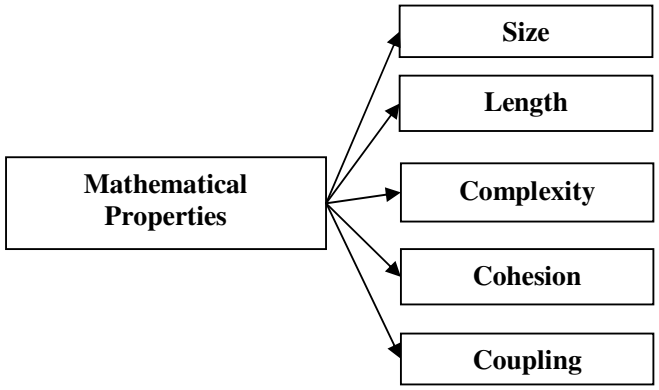

Figure 8. Mathematical Properties for Internal Attributes

The concepts of 'size and complexity' are related to system and modules. The possible system representation and modules and relationships can be defined as: A system S will be represented as a pair < E, R>, where E represents the set of elements of S, and R is a binary relation on E (R ⊆ E * E) representing the relationships between elements of S. Given system S = <E, R>, a system m = <E_m, R_m> is a module of S and only if E_m, ⊆ E, R_m⊆ E_m*E_m and R_m⊆ R. The concepts cohesion and coupling are meaningful only with reference to system that is provided with modular decomposition. The representation of modular system can be defined as a 3-tuple - MS = < E, R, M> represents a modular system in S = <E, R> is a system according to previous definition and M is a collection of modules of S.

Mathematical Properties of Measures for Size: The size is recognized as being an important measurement concept in software engineering. According to Briand's framework, 'size cannot be

negative. When a system does not contain any elements and when modules do not have elements in common then the size is expected to be additive. The size of a system S is a function Size(S) that is characterized by the following properties (Figure 9).

Property Size1 - Nonnegativity: The size of a system S = <E, R> is nonnegative, i.e., Size(S) ≥ 0. **Property Size 2 - Null Values:** The size of a system S = <E, R> is null if E is empty, E = ∅ ⇒ Size(S) = 0. **Property Size 3 - Module Additive:** The size of a system S = <E, R> is equal to the sum of the sizes of two of its modules m1 = <E_{m1}, R_{m1}> and m_2 = <E_{m2}, R_{m2}> such that any element of S is an element of either m_1 or m_2. ($m_1 \subseteq S$ and $m_2 \subseteq S$ and $E = E_{m1} \cup E_{m2}$ and $E_{m1} \cap E_{m2} = \emptyset$) ⇒ Size(S) = Size ($m_1$) + Size ($m_2$). The additional three properties for size from the above properties are shown below and they are named as property 4, 5 and 6. **Property Size 4 - Size of a System:** Property Size 4 provides the means to compute the size of a system S = <E, R> from the knowledge of the size of its – disjoint - modules m_e = < {e}, R_e> whose set of elements is composed of different elements e of E^1. Size(S) = Σ size (m_e). **Property Size 5 - Adding Elements:** Adding elements to a system cannot decrease its size. (S' = <E', R'> and S" = <E", R"> and E' ⊆ E") ⇒ Size (S') ≤ Size (S"). **Property Size 6 - Sum of Modules:** The properties, Size.1 - Size.3, follow the size of a system S = <E, R> is not greater than the sum of the sizes of any pair of its modules m_1 = <E_{m1}, R_{m1}> and m2 = <E_{m2}, R_{m2}>, such that any element of S is an element of m_1, or m_2, or both, i.e., ($m_1 \subseteq S$ and $m_2 \subseteq S$ and $E = Em_1 \cup Em_2$) ⇒ Size(S) ≤ Size (m_1) + Size (m_2). The size of a system built by merging such modules cannot be greater than the sum of the sizes of the modules, due to the presence of common elements. Properties Size.1 to Size.6 holds when applying the admissible transformation of the ratio scale. Therefore, there is no contradiction between Briand, et al. concept of size and the definition of size measures on a ratio scale.

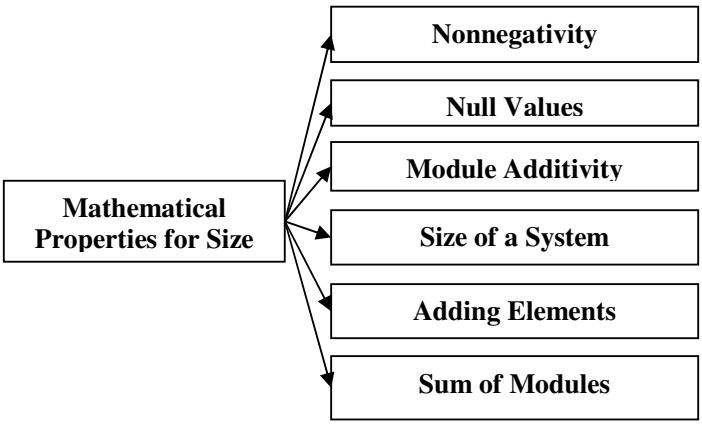

Figure 9. Mathematical Properties for Size

Mathematical Properties of Measures for Length: The properties Size.1 to Size.6 characterise the concept of size as commonly proposed in software measurement. In order to differentiate the measurement concept of length from size, other properties are proposed for "length". The length of a system S is a function, Length(S) characterised by the following properties Length.1 to Length.5 as shown in Figure 10.

Property Length 1 - Nonnegativity: The length of a system S = <E, R> is non-negative i.e., Length (S) ≥ 0. **Property Length 2 - Null Value:** The length of a system S = <E, R> is null if E is empty. (E = ∅) ⇒ (Length(S) = 0). **Property Length 3 - Nonincreasing Monotonicity for Connected Components:** Let S be a system and m be a module of S such that m is represented by a connected component of the graph representing S. Adding relationships between elements of m does not increase the length of S. (S = <E,R> and m = <E_m, R_m> and m ⊆ S and m " is a

connected component of S" and S' = <E,R'> and R' = R ∪ {<e_1,e_2>} and <e_1,e_2> ∉ R and e1 ∈ Em_1 and e_2 ∈ Em_1) ⇒ Length (S) ≥ Length (S'). **Property Length 4 - Nondecreasing Monotonicity for Non-connected Components:** Let S be a system and m_1 and m_2 be two modules of S such that m_1 and m_2 are represented by two separate connected components of the graph representing S. Adding relationships from elements of m_1 to elements of m_2 does not decrease the length of S. (S = <E,R> and m_1 = <E_{m1},R_{m1}> and m_2 = <Em_2,Rm_2> and m_1 ⊆ S and m2 ⊆ S " are separate connected components of S" and S' = <E,R'> and R' = R ∪ {<e1,e2>} and <e_1,e_2> ∉ R and e1 ∈ Em1 and e2 ∈ Em2) ⇒ Length(S') ≥ Length(S). **Property Length 5 - Disjoint Modules:** The length of a system S = <E, R> made of two disjoint modules m_1, m_2 is equal to the maximum of the lengths of m_1 and m_2. (S = m_1 ∪ m_2 and m_1 ∩ m_2 = ∅ and E = E_{m1} ∪ E_{m2}) ⇒ Length(S) = max {Length (m_1), Length (m_2)}. Properties Length 1 to Length 5 hold the admissible transformation of the ratio scale.

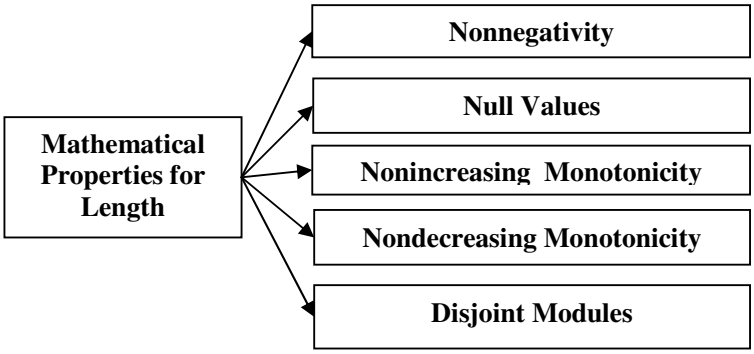

Figure 10. Mathematical Properties for Length

Mathematical Properties of Measures for Complexity: The complexity is a measurement concept that is considered extremely relevant to system properties. In Briand's framework, complexity of a system S is a function *Complexity (S)* that is characterized by the properties Complexity 1 to Complexity 5 as shown in Figure 11. **Property Complexity 1-Nonnegativity:** The complexity of a system S = <E, R> is non-negative i.e., Complexity (S) ≥ 0. **Property Complexity 2 - Null Value:** The complexity of a system S = <E, R> is null if R is empty. R = ∅ ⇒ Complexity (S) = 0. **Property Complexity 3 - Symmetry:**The complexity of a system S = <E, R> does not depend on the convention chosen to represent the relationships between its elements. (S=<E, R> and S_{-1}=<E, R_{-1}>) ⇒ Complexity(S) = Complexity(S^{-1}).

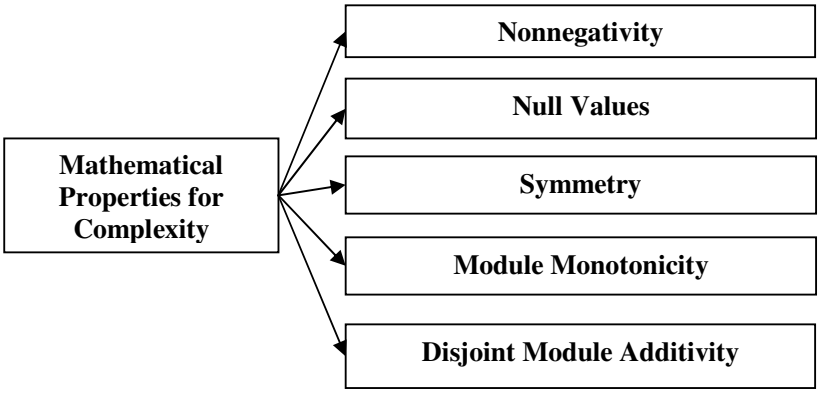

Figure 11. Mathematical Properties for Complexity

Property Complexity 4 - Module Monotonicity: The complexity of a system S = <E, R> is no less than the sum of the complexities of any two of its modules with no relationships in common. (S = <E, R> and m_1 = <E_{m1}, R_{m1}> and m_2 = <E_{m2}, R_{m2}> and $m_1 \cup m_2 \subseteq S$ and $R_{m1} \cap R_{m2} = \emptyset$) ⇒ Complexity (S) ≥ Complexity (m1) + Complexity (m2). **Property Complexity 5 - Disjoint Module Additivity:** The complexity of a system S = <E, R> composed of two disjoint modules m_1, m_2 is equal to the sum of the complexities of the two modules. (S = <E,R> and S = $m_1 \cup m_2$ and $m_1 \cap m_2 = \emptyset$) ⇒ Complexity(S) = Complexity (m_1) + Complexity (m_2).

Mathematical Properties of Measures for Cohesion: The concept of cohesion has been used with reference to modules or modular systems. The cohesion of a module [module m = < $E_{m,}$ R_m>] is a function Cohesion (m) characterized by the properties Cohesion 1to Cohesion 4 as shown in Figure 12. **Property Cohesion 1 - Nonnegativity and Normalization:** The cohesion of a module m = <E_m,R_m> of a modular system belongs to a specified interval. Cohesion (m) ∈ 0, Max. **Property Cohesion 2 - Null Value:** The cohesion of a module m = <$E_{m'}$,R_m> of a modular system is null if [Rm | IR] is empty. Rm = ∅ ⇒ Cohesion (m) = 0. **Property Cohesion 3 - Monotonicity:** Let MS' = <E,R',M'> and MS" = <E,R",M"> be two modular systems with the same set of elements E such that there exist two modules m' = <E_m,R_m'> and m" = <E_m,R_m> with the same set of elements E_m belonging to M' and M", respectively, such that R' - $R_{m'}$ = R" - $R_{m"}$, and $R_{m'} \subseteq R_{m"}$ (which implies IR' ⊆ IR"). Adding intra-module relationships does not decrease module cohesion.

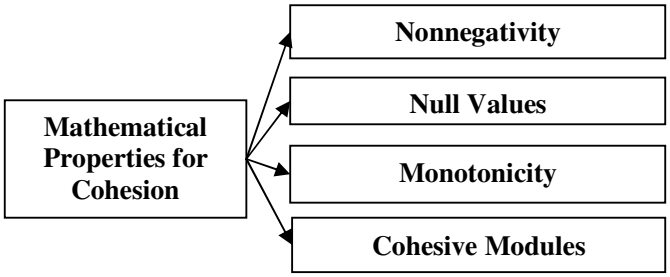

Figure 12. Mathematical Properties for Cohesion

Property Cohesion 4 - Cohesive Modules: Let MS' = <E,R,M'> and MS" = <E,R,M"> be two modular systems with the same underlying system <E,R> such that M" = M' - {m'_1,m'_2} ∪ {m"}, with $m'_1 \in M'$, $m'_2 \in M'$, m" ∉ M', and m" = $m'_1 \cup m'_2$. The cohesion of a module obtained by combining together two unrelated modules is not greater than the maximum cohesion of the two original modules. Properties Cohesion 1 to Cohesion 4 hold when applying the admissible transformation of the ratio scale. Therefore, there is no contradiction between concept of cohesion and the definition of cohesion measures on a ratio scale.

Mathematical Properties of Measures for Coupling: The concept of coupling has been used with reference to modules or modular systems. The coupling captures the amount of relationship between the elements belonging to different modules of a system. Given a module m, two kinds of coupling can be defined: inbound coupling and outbound coupling. The former captures the amount of relationships from elements outside m to elements inside m; the captures the amount of relationships from elements inside m to elements outside m. The coupling of a module [module m = <$E_{m'}$,R_m> of a modular system MS | modular system MS] is a function [Coupling(m) | Coupling(MS)] characterised by the properties Coupling 1 to Coupling 5 are shown in Figure 13. **Property Coupling 1 - Nonnegativity:** The coupling of a module m = <E_m,R_m> of a modular system (MS) is nonnegative. Coupling (m) ≥ 0 | Coupling (MS) ≥ 0. **Property Coupling 2 - Null Value:** The coupling of a module m = <E_m, R_m> of a modular system MS = <E,R,M>] is null if [Outer R(m)| R-IR] is empty. **Property Coupling 3 - Monotonicity:** Let MS' = <E,R',M'> and

MS" = <E,R",M"> be two modular systems with the same set of elements E such that there exist two modules m' ∈ M', m" ∈ M" such that R' – Outer R(m') = R" – Outer R(m"), and Outer R(m') ⊆ Outer R(m"). Adding inter-module relationships does not decrease coupling. Coupling (m') ≤ Coupling (m") | Coupling (MS') ≤Coupling (MS"). **Property Coupling 4 - Merging of Modules:** Let MS' = <E',R',M'> and MS" = <E",R",M"> be two modular systems such that E' = E", R' = R", and M" = M' - {m'_1,m'_2} ∪ {m"}, where m'_1 = <$E_{m'1}$,$R_{m'1}$>, m'_2 = <$E_{m'2}$,$R_{m'2}$>, and m" = <E_m",R_m">, with m'_1 ∈ M', m'_2 ∈ M', m" ∉ M', and E_m" = $E_{m'1}$ ∪ $E_{m'2}$ and R_m" = $R_{m'1}$ ∪ $R_{m'2}$. The two modules m'_1 and m'_2 are replaced by the module m", whose elements and relationships are the union of those of m'_1 and m'_2 [Coupling (m'1) + Coupling (m'2) ≥ Coupling (m") | Coupling(MS') ≥ Coupling(MS")]. The coupling of a module or modular system obtained by merging two modules is not greater than the sum of the couplings of the two original modules coupling of the original modular system, since the two modules may have common inter-module relationships.

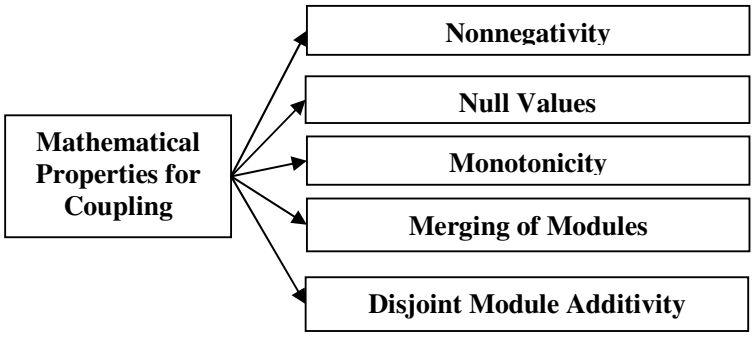

Figure 13. Mathematical Properties for Coupling

Property Coupling 5 - Disjoint Module Additivity: Let MS' = <E,R,M'> and MS" = <E,R,M"> be two modular systems with the same underlying system <E,R> such that M" = M' - {m'_1,m'_2} ∪ {m"}, with m'_1 ∈ M', m'_2 ∈ M', m" ∉ M', and m" = m'_1 ∪ m'_2. The two modules m'_1 and m'_2 are replaced by the module m" and union of m'_1 and m'_2. If no relationship exists between the elements belonging to m'_1 and m'_2, i.e., InputR(m'_1) ∩ OutputR(m'_2) = ∅ and InputR(m'_2) ∩ OutputR(m'_1) = ∅, then Coupling (m'_1) + Coupling (m'_2) = Coupling (m") | Coupling (MS') = Coupling (MS"). The coupling of a module obtained by merging two unrelated modules is equal to the sum of the couplings of the two original modules | coupling of the original modular system. The concepts of Briand, et al. describes the scales are excluded by the Weyuker's Properties [50]. The above discussed theoretical validation methodologies are used for validating software metrics. The Weyuker's properties of validation are more widely adopted in the literature for the theoretical validation of software metrics.

6. CONCLUSION

In software engineering, recently, software metrics researchers have introduced new software metrics and validated metrics using theoretical and empirical techniques and software metrics have been used in decisions-making as well as in various process activities and more researchers are involved in empirical validations. The existing methodologies of software metrics validations which have facilitated the validations of the software metrics have been discussed in detail. This paper discussed the theoretical validations and empirical validations methodology in software measurement. The properties of software measures proposed by eminent researchers Weyuker, E.J., Kitchenham, B., Pfleeger, S.L., and Fenton, N., and Briand, L.C., Morasca, S., Basili, V.R. are discussed and reviewed has been presented. In future, software metrics research work will be

based on using software metrics in software development for the improving the efficiency, cost estimates and quality [22, 35, 41, 44].

REFERENCES

[1] Aggarwal, K.K., Singh, Y., Kaur, A., and Malhotra, R., 2006, "Software Design Metrics for Object-Oriented Software," Journal of Object Technology, Vol. 6, No. 1, January-February, pp. 121-138.

[2] Alshayeb, M., and Li, W., 2003, "An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes," IEEE Transactions on Software Engineering, Vol. 29, No. 11, November, pp. 1043-1049.

[3] Anbumani, K., and Srinivasan, K.P., 2005, "A Set of Object-Oriented Design Metrics," Journal of The Institution of Engineers (India), IE(I), Journal – CP, Volume 86, May, pp. 1-9.

[4] Archer C. and Stinson M.,1995, Object-Oriented Software Measures, Technical Report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, April.

[5] Asad, A.A., Alsmadi, I., 2014, "Evaluating the Impact of Software Metrics on Defects Prediction. Part 2," Computer Science Journal of Moldova, Vol.22, No.1 (64).

[6] Badri, L., and Badri, M., 2004, "A Proposal of a New Class Cohesion Criterion: An Empirical Study," Journal of Object Technology, Vol. 3, No. 4, April, pp. 145-159.

[7] Badri, L., Badri, and Gueye, A.B., 2008, "Revisiting Class Cohesion: An empirical Investigation on Several Systems," Journal of Object Technology, Vol. 7, No. 6, July-August, pp. 55-75.

[8] Bandi, R.K., Vaishnavi, V.K., and Turk, D.E., 2003, "Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics," IEEE Transactions on Software Engineering, Vol. 29, No. 1, January, pp. 77-87.

[9] Basili, V.R., and Weiss, D.M., 1984, "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, Vol. SE-10, No. 6, November, pp. 728-738.

[10] Briand, L.C., Morasca, S., Basili, V.R., 1996, "Property-Based Software Engineering Measurement," IEEE Transactions on Software Engineering, Vol. 22, No.1, January, pp. 68-85.

[11] Chemiavsky, J.C., and Smith, C.H., 1991, "On Weyuker's Axioms For Software Complexity Measures," IEEE Transactions on Software Engineering, Vol. 17, No. 6, June, pp. 636-638

[12] Chhabra, P., Bansal, L., 2014, "An Effective Implementation of Improved Halstead Metrics for Software Parameters Analysis," IJCSMC, Vol. 3, Issue. 8, August, pg.146 – 161.

[13] Chhillar,R,S., Kajla, P., Chhillar, U., and Kumar, N., 2014, "An Access Control Metric Suite for Class Hierarchy of Object-Oriented Software Systems," International Journal of Computer and Communication Engineering, Vol. 4, No. 1, January.

[14] Chidamber, S.R., and Kemerer, C.F., 1994, "A Metrics Suite for Object-Oriented Design," IEEE Transactions on Software Engineering, Vol. 20, No. 6, June, pp. 476-493.

[15] Deligiannis, I., Shepperd, M., Roumeliotis, M., and Stamelos, I., 2003, "An Empirical Investigation of an Object-Oriented Design Heuristic for Maintainability," The Journal of Systems and Software ,65 (2003), pp. 127–139.

[16] Drouin, N., Badri, M., and Touré, F., 2013, "Metrics and Software Quality Evolution: A Case Study on Open Source Software," International Journal on Computer Theory and Engineering, Vol. 5, No. 3, June, pp. 523-527.

[17] Emam, K.E., Benlarbi, S., Goel, N., and Rai, S.N., 2001, "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics," IEEE Transactions on Software Engineering, Vol. 27, No. 7, July, pp. 630-650.

[18] Evanco, W.M., 2003, "Comments on 'The Confounding Effect of Class Size on the Validity of Object-Oriented Design Metrics," IEEE Transactions on Software Engineering, Vol.29, No.7, July, pp. 670-672.

[19] Fenton, N.E., and Pfleeger, S.L., 2004, Software Metrics: A Rigorous and Practical Approach, Thomson Asia, Singapore.

[20] Garg, P., Sangwan, S., Garg, R.K., 2014, "Design an Expert System for Ranking of Software Metrics," International Journal for Research in Applied Science and Engineering Technology, Vol. 2 Issue 8, August, pp. 109-117.

[21] Harrison, R, Counsell, S.J. and Nithi, R.V., 1998, "An Investigation into the Applicability and Validity of Object-Oriented Design Metrics," Empirical Software Engineering, 3, pp. 255–273.

[22] Jones, C., 2014,"Evaluating Software Metrics and Software Measurement Practices," Version 4, Namcook Analytics, March, (http://Namcookanalytics.com).
[23] Kaur, A., Singh, S., Kahlon, K. S., and Sandhu, P.S., 2010, "Empirical Analysis of CK and MOOD Metric Suit," International Journal of Innovation, Management and Technology, Vol. 1, No. 5, December, pp. 447-452.
[24] Kitchenham, B., Pfleeger, S.L., and Fenton, N., 1995, "Towards a Framework for Software Measurement Validation," IEEE Transactions on Software Engineering, Vol. 21, No.12, December, pp. 929-943.
[25] Kulkarni, R., Sharma, S.D., 2014, "A Metric Base Calculation for Object Oriented Software Modularization Quality Measurement," International Journal Of Engineering And Computer Science, Vol. - 3 Issue -9 September, pp. 8175-8178.
[26] Laila Cheikhi, Rafa E. Al-Qutaish, Ali Idri1 and Asma Sellami, 2014, " Chidamber and Kemerer Object-Oriented Measures: Analysis of their Design from the Metrology Perspective," International Journal of Software Engineering and Its Applications, Vol.8, No.2, pp.359-374
[27] Li, H.F., and Cheung, W.K., 1987, "An Empirical Study of Software Metrics," IEEE Transactions on Software Engineering, Vol. SE-13, No. 6, June, pp. 697-708.
[28] Malik, N., and Chhillar, R.S., 2011, "New Design Metrics for Complexity Estimation in Object Oriented Systems," International Journal on Computer Science and Engineering, Vol. 3 No. 10, October, pp. 3367-3382.
[29] Misra, S., and Akman, I., 2008, "Applicability of Weyuker's Properties on OO Metrics: Some Misunderstandings," ComSIS, Vol. 5, No. 1, June, pp. 17-24.
[30] Muketha, G.M., Ghani, A.A.A., Selamt, M.H., and Atan, R., 2010, "A Survey of Business Complexity Metrics," Information Technology Journal, 9 (7), pp. 1336-1344.
[31] Nigam, P., Mishra, R., 2014, "Coupling Appraisal in Object-Oriented Systems," International Journal of Engineering and Innovative Technology, Vol. 3, Issue 11, May.
[32] Nyasente, S.M., Mwangi, W., Kimani, S., 2014, "A Metrics-based Framework for Measuring the Reusability of Object-Oriented Software Components," Journal of Information Engineering and Applications, Vol.4, No.4.
[33] Paliwal, N., Shrivastava, V., Tiwari, K., 2014, "An Approach to Find Reusability of Software Using Objet Oriented Metrics," International Journal of Innovative Research in Science, Engineering and Technology, Vol. 3, Issue 3, March.
[34] Radhika Raju, P., and Ananda Rao, A., 2014, "A Metrics Suite for Variable Categorization to Support Program Invariants," International Journal of Software Engineering & Applications, Vol.5, No.5, September, pp. 65-83.
[35] Rajnish, K., 2014, "Theoretical Validation of Inheritance Metrics for Object-Oriented Design against Briand's Property," International Journal of Information Engineering and Electronic Business, 3, pp. 28-33.
[36] Rajnish, K., and Bhattacherjee, V., 2005, "Complexity of Class and Development Time: A Study," Journal of Theoretical and Applied Information Technology, pp. 63-70, 2005.
[37] Schneiderwind, N.F., 1992, "Methodology for Validating Software Metrics," IEEE Transactions on Software Engineering, Vol. 18, No. 5, May, pp. 410-422.
[38] Sharma, A., Kumar, R., and Grover, P. S., 2007, "Empirical Evaluation and Critical Review of Complexity Metrics for Software Components," Proceedings of the 6th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems, Greece, pp. 24-29.
[39] Shen,V.Y., Yu, T.Y., Thebaut, M., and Paulsen, L.R., 1985, "Identifying Error-Prone Software – An Empirical Study," IEEE Transactions on Software Engineering, Vol. SE-11, , No. 4, pp.317-323.
[40] Singh, V., and Bhattacherjee, V., 2014, "Assessing Package Reusability in Object-Oriented Design," International Journal of Software Engineering and Its Applications, Vol.8, No.4, pp.75-84.
[41] Srinivasan, K.P., and Devi, T., 2014, "A Novel Software Metrics and Software Coding Measurement in Software Engineering," International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 4, Issue 1, January, pp. 303-308.
[42] Srinivasan, K.P., and Devi, T., 2009, "Design and Development of a Procedure to Test the Effectiveness of Object-Oriented Design," International Journal of Engineering Research and Industrial Applications, Vol.2, No.6, pp. 15-25.
[43] Srinivasan, K.P., and Devi, T., 2011, "Design and Development of a Procedure for new Object-Oriented Design Metrics," International Journal of Computer Applications, Vol.24, No.8, pp. 30-35.

[44] Srinivasan, K.P., and Devi, T., 2014, "A Complete and Comprehensive Metrics Suite for Object-Oriented Design Quality Assessment," International Journal of Software Engineering and Its Applications, Vol. 8, No. 2, February, pp.173-188. (This Paper is recognized as "Quality Paper" by SERSC, Republic of Korea and Published with Free of Cost).

[45] Srinivasan, K.P., and Devi, T., 2014, "A Comprehensive Review and Analysis on Object-Oriented Software Metrics in Software Measurement," International Journal on Computer Science and Engineering, Vol. 6, No.7, July, pp.30-35.

[46] Vanitha., N., and ThirumalaiSelvi., R., 2014, "A Report on the Analysis of Metrics and Measures on Software Quality Factors – A Literature Study," International Journal of Computer Science and Information Technologies, Vol. 5 (5) , pp. 6591-6595.

[47] Weyuker, E.J., 1988, "Evaluating Software Complexity Measure," IEEE Transactions on Software Engineering, Vol. 14, No. 9, September, pp. 1357-1365.

[48] Yadav, K.P., Singh, Y., and Yadav, S., 2011, "Software Project Metrics and Measurement," International Journal of CS and IT, Vol. 1(10), pp. 711-719.

[49] Yao, H., Orme, A.M., and Etzkorn, L., 2005, "Cohesion Metrics for Ontology Design and Application," Journal of Computer Science, 1(1), pp. 107-113.

[50] Zuse, H., 1997, "Reply to: Property-Based Software Engineering Measurement," IEEE Transactions on Software Engineering, Vol. 23, No. 8, August, pp. 533.

AUTHORS

Dr. K.P. SRINIVASAN received his Master of Computer Applications Degree from Bharathiar University, Coimbatore, India in 1993. He completed his M.Phil. Degree in Computer Science from Bharathiar University, Coimbatore, India in 2001 and Ph. D. Degree from School of Computer Science and Engineering, Bharathiar University, Coimbatore, India in 2014. Presently, he is working as an Associate Professor in Computer Science in C.B.M. College, Kovaipudur, Coimbatore under Bharathiar University, Coimbatore, India since 1997. He has published five conference papers and seven journal papers. He has received the best paper award from a conference and "quality paper" recognition from a reputed journal. His current research interests are in the areas of Software Engineering and Object-Oriented Systems.

Prof. Dr. T. DEVI received Master of Computer Applications Degree from P.S.G. College of Technology, Coimbatore, India in 1987 and the Ph.D. Degree from the University of Warwick, United Kingdom in 1998. Presently, she is working as a Professor and Head of the Department of Computer Applications, School of Computer Science Engineering, Bharathiar University, Coimbatore, India. Prior to joining Bharathiar University, she was an Associate Professor in Indian Institute of Foreign Trade, New Delhi, India. She has contributed more than 140 papers in various Journals and Conferences. Her current research interests are in the areas of Software Engineering and Concurrent Engineering.