Refining the Axiomatic Definition of Internal Software Attributes

Sandro Morasca
Dipartimento di Scienze Politiche, della Cultura e dell'Informazione
Università degli Studi dell'Insubria
Via Carloni 78, I-22100, Como, Italy
sandro.morasca@uninsubria.it

ABSTRACT

Several internal software attributes, like size, structural complexity, cohesion, coupling, have been introduced and used to reason about software engineering artifacts, and many measures have been proposed for them. Internal software attributes are important because they are believed to be related to quantities of industrial interest, like the number of defects or the development effort. However, the definition of internal software attributes still needs to be made more precise and formal, so measures can be defined that really quantify the attributes they purport to measure. In this paper, we extend, simplify, and refine an existing axiomatic approach that characterizes each internal attribute rigorously via a different set of axioms. This paper makes three specific contributions. First, the new proposal captures a larger set of aspects of software artifacts that may be relevant for internal software attributes than the original proposal did. Second, we identify the basic, foundational sets of axioms for each internal attribute studied, from which the other properties of the attribute can be derived, so the intrinsic properties of the attribute and their implications can be understood. Third, we investigate some relevant relationships among internal software attributes, so their similarities and differences, which are sometimes not well identified, are made more explicit.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, product metrics*

General Terms

Measurement, Theory

Keywords

Internal software attributes, Size, Complexity, Cohesion, Coupling

1. INTRODUCTION

The quality of software artifacts is a central aspect in software engineering, even more than one may usually realize. Techniques are introduced and used because they are supposed to improve the resulting artifact quality. For instance, Object-Oriented techniques were originally introduced with the aim of building modular software systems with reduced coupling and increased cohesion so as to improve the maintainability of the systems. Decisions are made during the software engineering lifecycle with some software quality in mind. For instance, a specific architecture is chosen because it is believed that it may provide satisfactory results as for the complexity, performance, or usability of the system. It is also believed that techniques and decisions may affect positively some software qualities and adversely some others. So, the trade-offs among these qualities should always be kept in mind when using a technique or making a decision.

It is usual in software engineering to distinguish between internal and external software attributes (i.e., qualities). Internal software attributes, like size and structural complexity, can be defined and measured based only on the software artifact. External software attributes, like usability and maintainability, depend on the software artifact, the "users" of the software artifact, and the context in which the "users" use the software artifact. There are several categories of "users" of a software artifact, including: the final users, who are interested in and influence the assessment of, say, the artifact's usability, as different users have different perceptions of the artifact's usability; the developers, who are interested in and influence the artifact's maintainability, for instance, as the maintainability of a software artifact may well depend on the maintainer; and the operational environment, which affects the artifact's performance, for instance. External software attributes are the really interesting ones from an industrial perspective. Internal software attributes are not interesting *per se*, but they are interesting because they are believed to be related to external software attributes or process attributes. For instance, software coupling is believed to be related to software maintainability and software size to lifecycle effort. Thus, techniques and decisions are often introduced and used with the idea that they affect internal software attributes, which, in turn, affect external software attributes or process attributes.

However, until a clear understanding of these attributes is reached, the claims about the impact of techniques or decisions on internal software attributes cannot be tested. For instance, it is not really possible to check whether a new technique decreases the coupling among software components until a clear and widely accepted definition of coupling is reached. The rigorous definition of an internal software attribute requires that the intuition about it be formalized so it can be subject to review, revision, and eventual acceptance. The rigorous definition of the attributes of the objects of study is a precondition for the development of any fields that follows a truly scientific approach, since it allows for the definition of sensible measures for these attributes. A measure that agrees with the definition of the software attribute it purports to measure is said to be "theoretically valid," and can be used and understood by researchers and practitioners. Thus, only after rigorous definitions are available for software attributes will it be possible to run rigorous empirical studies that allow researchers and practitioners to assess the truthfulness of claims. The lack of theoretical bases has led in the last few years to the definition of a plethora of measures, the vast majority of which have not survived the proposal phase.

Two main approaches have been used in the last few years to study the theoretical bases of software attributes and measures. Measurement Theory [21] is the general framework that has been used for providing the theoretical foundations for measurement in a number of disciplines, including psychology. However, Measurement Theory has mostly been used in software engineering to show some theoretical pitfalls of existing software measures and has provided little guidance for the definition of new measures. It is this author's belief that a theory is more useful if it also provides guidance for the definition of new solutions. To this end, Axiomatic Approaches have proven to be successful in guiding the definition of internal software attributes and their measures (see [1, 6, 11, 14], for instance) and the classification of cohesion and coupling measures see [2, 3]. A few axiomatic proposals exist for defining internal attributes, most of which address a specific internal software attribute: complexity. Prather [20] and then Weyuker [22] defined axioms for complexity measures of software code. Lakshmanian et al. [12] introduced axioms for software complexity measures based on control flow graphs. The approach by Poels and Dedene [19] is based on the notion of distance for the definition of properties for internal software attributes.

The approach of Briand, Morasca, and Basili [4] has the goal of providing sets of axioms for a number of different software attributes, namely size, length, complexity, cohesion, and coupling, instead of complexity alone. The goal of [4] was to establish a way for discussing about internal software attributes in a precise way, and evaluate the similarities and differences among them. The various axiom sets were introduced for software artifacts in general, i.e., not only for software code. So, the axiom sets were defined based on a graph-based representation used to model the elements of software artifacts and the relationships among them. The axiom sets were also introduced as necessary ones, i.e., they were already expected to be open to revisions and modifications (see [18, 5, 11]). In general, sets of axioms formalize intuitive knowledge, so it is not possible to formally show that a set of axioms is complete for an attribute, since completeness can be claimed only when a widespread consensus is reached among researchers and practitioners alike. So, being a relatively new proposal in software engineering measurement, Axiomatic Approaches still need to be further refined to better capture the nature of software artifacts and make the various aspects of internal software attributes more precise and explicit. At any rate, the axiomatic approach, which has been successfully used well beyond the formalization of software attributes, may be also used in the formalization of other software attributes than the ones of this paper.

A first extension of these axiom sets was carried out in [15], where relationships could take place among more than two elements and an additional axiom for coupling was introduced. In addition, issues related to scale type were investigated, since the axiom sets of [4] only allowed measures on the ratio level of measurement to be defined, i.e., measures for which the ratio between measurements makes sense, like measures of length or weight in everyday life. However, other types of measures may be defined, like ordinal ones, which simply order a set of entities with respect to a specific attribute, but provide no information about the "distance" or the ratio between the measurement values. For instance, a software engineering measure like failure criticality is usually defined on an ordinal level, with values like: critical, serious, medium, suggestion, cosmetic. The "distance" in criticality between, say, "critical" and "serious" is not known, nor is it known whether it is the same as the "distance" between "serious" and "medium." In [15], the definition of axiom sets for defining measures at the standard levels of measurement, i.e., nominal, ordinal, interval, ratio, and absolute ones was investigated and axiom sets were provided for these different levels of measurement for some software attributes.

Along these lines, this paper provides further advances for the axiomatic approach of [4], by extending, simplifying, and refining it. Specifically, in this paper we

- extend the graph-based representation of [4] into a multigraph-based representation of software artifacts, to better mirror the structure of existing modeling notations and techniques
- identify the basic sets of axioms (and in one case also refine one of the original axioms) for each internal software attribute, and show their consequences as derived properties, to check how well the axioms agree with intuition and what kind of additional features of each internal software attribute can be identified; the measurement level needed for a measure to satisfy each basic axiom and derived property will also be shown
- study the similarities, differences, and relationships between internal software attributes.

The remainder of this paper is organized as follows. Section 2 introduces an extended definition of the graph-based model of [4]. Section 3 describes the refined sets of base axioms and the derived properties. Section 4 explores the relationships among software attributes. An outline of future work is in Section 5.

2. EXTENDED DEFINITIONS OF SYSTEMS AND MODULES

We now provide a definition that extends the definition of system of [4]. The definition is based on the idea that a system is basically a graph, where each arc is associated with a multiset of relationships, and each relationship has a type.

Definition 1. System. A system S is a pair $S =< E, R >$, where

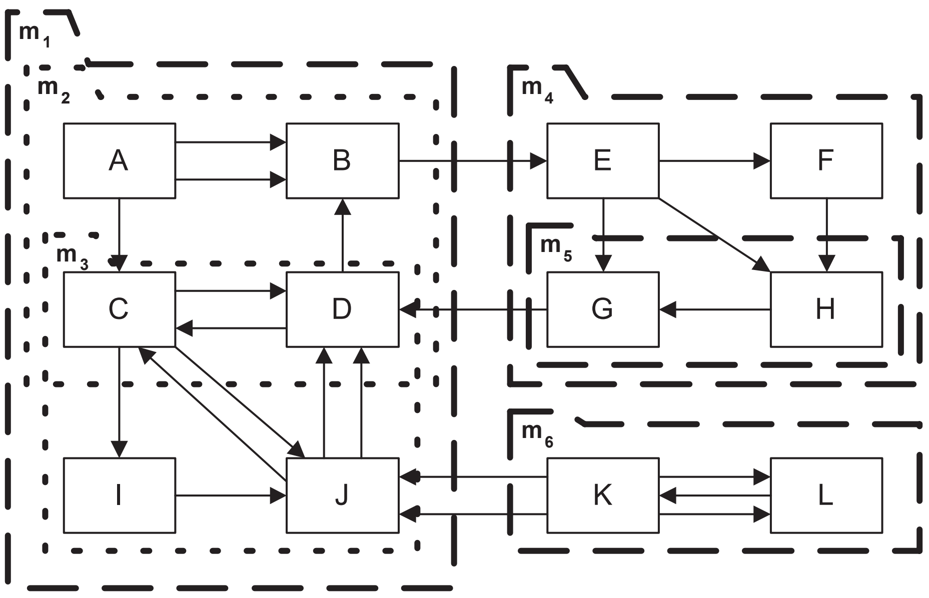

Figure 1: Representation of Systems and Modules.

- E represents the set of elements of S
- $R \in N^{E \times E \times T}$

where T is a finite set of types of relationships (N is the set of natural numbers, including 0).

In this general representation, system elements (which play the part of the nodes of the graph) may denote different things, depending on the application at hand. For instance, elements may be code statements, methods, classes, etc. Each arc of the graph connecting two elements, is associated with a multiset of typed relationships, to take into account the different kinds of connections that may exist between system elements. Formally, if $a \in E$, $b \in E$, $t \in T$, and $n \in N$, the expression $<< a, b, t >, n >$ associates $< a, b, t >$, i.e., a typed relationship between elements a and b, with a natural number n, which describes how many times the typed relationship $< a, b, t >$ occurs. For completeness, $<< a, b, t >, 0 >$ denotes the fact that the typed relationship $< a, b, t >$ does not belong to the system.

The definition above extends the definition of [4] in two ways: (1) each element of R is a multiset, while $R \subseteq E \times E$ was simply a binary relation in the original proposal [4]; (2) each relationship is typed. These extensions make it possible to model software artifacts and the characteristics of software attributes more closely. For instance, take the UML-like class diagram in Fig. 1, which is provided here for explanation purposes. Note that, rigorously speaking, Fig. 1 does not represent a fully proper UML class diagram because, in UML, a class, like C or D, may not be included in two packages. We use a UML-like example for even greater generality, and the arcs represent associations that may well have different types, e.g., aggregation, containment, inheritance, use, etc. As for having a multiset, two classes may be connected by several relationships, even of the same kind. Classes K and L are two elements of what we call a system according to Definition 1 and they are linked by three relationships, so a multiset of relationships may be more appropriate than a set.

As for typing the relationships, the choice of having several types of relationships in the definition of system is due to the coexistence of different kinds of relationships in actual software systems. For instance, in the UML-like class diagram of Figure 2, there are inheritance relationships between M and N, M and O, M and P, Q and R, and there are aggregation relationships between M and Q, P and Q (with cardinality 2 on Q's side). The aggregation between M and Q gets inherited by N, O, and P. The inheritance relationship between Q and R, by effect of the aggregation relationship between M and Q also establishes aggregation relationships between N and R, O and R, P and R. So, there may be several different types of relationships, which may affect each other in various ways, and cannot be studied independently of each other, in general.

In what follows, the same symbol (e.g., $\cup$ for union) may denote an operation between sets when sets of elements are involved, or an operation between multisets when multisets of typed relationships are involved. Even though these operations are different, no confusion will arise because there will never be an operation whose operands are a set of elements and a multiset of typed relationships.

For completeness, we provide the meaning of these operations when they occur between two multisets of typed relationships R_1, R_2, which are the usual definitions of operations used for multisets.

Inclusion. $R_1 \subseteq R_2 \Leftrightarrow \forall << a, b, t >, n_1 > \in R_1$, $\exists << a, b, t >, n_2 > \in R_2 \wedge n_1 \leq n_2$, i.e., R_2 contains at least all the occurrences of the typed relationships in R_1.

Union. $R_3 = R_1 \cup R_2 \Leftrightarrow \forall << a, b, t >, n_3 > \in R_3$, $\exists << a, b, t >, n_1 > \in R_1, << a, b, t >, n_2 > \in R_2, n_3 = n_1 + n_2$, i.e., R_3 gathers all the occurrences of the typed relationships in R_1 and R_2.

Intersection. $R_3 = R_1 \cap R_2 \Leftrightarrow \forall << a, b, t >, n_3 > \in R_3, \exists << a, b, t >, n_1 > \in R_1, << a, b, t >, n_2 > \in R_2, n_3 =$

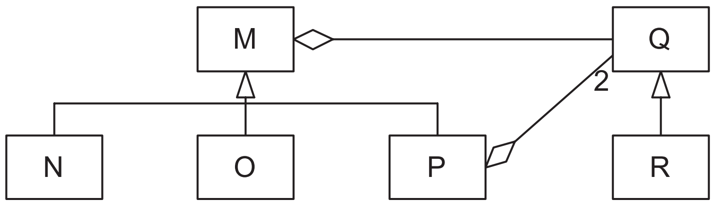

Figure 2: A UML-like Class Diagram

$min\{n_1, n_2\}$, i.e., R_3 contains all the occurrences of typed relationships in common to R_1 and R_2.

The original definition of system was extended in [15], where R was defined as a set of tuples $< e_1, e_2, , e_i, , e_n >$ with possibly different arities. However, an n-ary relation can be represented by binary relations. A graphical example of this is given by Entity-Relationship diagrams. Each ER relationship is represented by a specific node. (We use the term "ER relationship" here to distinguish it from a "relationship" in a set- or multiset-theoretic sense.) Thus, an entity that participates in a n-ary ER relationship is connected to the node representing the ER relationship via an edge. An ER Diagram is actually a graph whose nodes represent either entities or ER relationships, and whose edges represent the fact that an entity participates in a relationship. An ER Diagram represents a system in which $E = \{entities, ERrelationships\}$, and whose relationships $R \subseteq E \times E$ are depicted by the edges of the ER diagram. So, the definition of [15] made it easier to represent the relationships among system elements in general, but was not substantially more general than that of [4].

Some software attributes require that parts of a system be identified. For instance, cohesion and coupling may be defined for a specific part of the system (e.g., a class or a package) or an entire system. These parts of a system are actually subsystems that we call *modules*.

Definition 2. Module. Given a system $S =< E, R >$, a module $m =< E_m, R_m >$ is a system such that $E_m \subseteq E \wedge R_m \subseteq R$.

Thus, for maximum generality and simplicity, a module is simply a subsystem, with no additional characteristics (e.g., an interface).

Given a module m of a system $S =< E, R >$, the multiset of those relationships that involve one element within m and one element outside m is called the set of outer relationships of m and is denoted as $OuterR(m)$, with $OuterR(m) \subseteq R$. In Fig. 1, UML-like packages m_1, m_2, m_3, m_4, m_5, m_6, may be interpreted as modules. For notational convenience, it will be implied in the remainder of the paper that the set of elements and the multiset of relationships of a system or a module have the same subscript as the system or module, unless otherwise explicitly specified. For instance, it will be implied that m_1 has the set of elements E_1 and the multiset of relationships R_1. If no subscript appears in the name of the system or module, then no subscript appears in the name of its set of elements and multiset of relationships.

A few operations and definitions may be provided for modules based on set and typed multiset operations, as follows.

Inclusion. Module m_1 is said to be included in module m_2 (notation: $m_1 \subseteq m_2$) if $E_1 \subseteq E_2 \wedge R_1 \subseteq R_2$. In Fig. 1, $m_5 \subseteq m_4$.

Union. The union of modules m_1 and m_2 (notation: $m_1 \cup m_2$) is the module $< E_1 \cup E_2, R_1 \cup R_2 >$. In Fig. 1, $m_1 = m_2 \cup m_3$.

Intersection. The intersection of modules m_1 and m_2 (notation: $m_1 \cap m_2$) is the module $< E_1 \cap E_2, R_1 \cap R_2 >$. In Fig. 1, $m_2 \cap m_3$ is the module whose elements are classes C and D and whose relationships are $<< C, D, t >, 1 >$ and $<< D, C, u >, 1 >$ (assuming that they have type t and u, respectively).

Empty module. Module $< \oslash, \oslash >$ (denoted by $\oslash$) is the empty module.

Disjoint modules. Modules m_1 and m_2 are said to be disjoint if $m_1 \cap m_2 = \oslash$. In Fig. 1, m_3 and m_6 are disjoint.

Unconnected modules. Two disjoint modules m_1 and m_2 of a system are said to be unconnected if $OuterR(m_1) \cap OuterR(m_2) = \oslash$. In Fig. 1, m_4 and m_6 are unconnected, while m_3 and m_6 are not unconnected.

3. BASIC AXIOM SETS AND DERIVED PROPERTIES

Here, we refine the axiom sets of [4] in three ways.

- First, the base axioms need to be extracted from the existing axiom sets. This simplification allows us to remove redundancies from the core sets of basic ideas behind the modeling of internal software attributes, so similarities and differences among software attributes can be assessed.

- Properties are derived and proven as implications of the base axioms. These derived properties can be used to further check whether the modeling of an internal software attributes is consistent with the intuition on it. The derived properties are "weaker" than the base ones and are often satisfied by measures that are ordinal or nominal and not necessarily ratio ones. This is not just a theoretical exercise, but can guide the building of ordinal or nominal measures, instead of only ratio ones.

- One axiom for coupling has been extended (Section 3.4) to make the axiom set more adequate for modeling coupling.

For space reasons, we do not deal here with one of the original attributes, which was called *Length* in [4].

The decision as to which properties are the base ones and so they are taken as axioms and which are derived properties is somewhat subjective. We mostly use properties satisfied by ratio measures as the base axioms and, often, properties satisfied by ordinal measures are derived. Each axiom and property is annotated by the level of measurement of the measures to which the axiom or property can be applied to. Also, the axiom sets presented here show a possible formalization of the measures for each attribute, and their acceptance as the appropriate axioms can only come through a consensus. As mentioned in the Introduction, other formalizations of the properties of the measure for some of the same attributes have been proposed. Our approach, along with the others, will help clarify what can be expected of measures for internal software attributes until a widely accepted set of axiom is identified for each attribute.

3.1 Size

The set of axioms of [4] included three axioms. Here, we show that only two of them need to be used as base ones, and the third one and several properties can then be derived. Size is based on the set of elements of a system or a module (which is a subsystem anyway), as the following axioms show.

The first axiom is based on the idea that the size of a module composed of two possibly overlapping modules is not greater than the sum of the sizes of the two modules, each taken in isolation.

Size Axiom 1. Union of Modules (ratio measures). The size of a system S is not greater than the sum of the sizes of two of its modules m_1 and m_2 such that each element of S is an element of either m_1 or m_2 or both

$$E = E_1 \cup E_2 \Rightarrow Size(S) \leq Size(m_1) + Size(m_2)$$

For instance, $Size(m_1) \leq Size(m_2) + Size(m_3)$ in Fig. 1. When the two modules are disjoint, size is additive.

Size Axiom 2. Module Additivity (ratio measures). The size of a system S is equal to the sum of the sizes of two of its modules m_1 and m_2 such that any element of S is an element of either m_1 or m_2 but not both

$$E = E_1 \cup E_2 \wedge E_1 \cap E_2 = \oslash \Rightarrow$$
$$Size(S) = Size(m_1) + Size(m_2)$$

Thus, $Size(m_1 \cup m_6) = Size(m_1) + Size(m_6)$ in Fig. 1.

Several properties can be derived from the base axioms, so the characteristics of size can be better clarified. The following property is Axiom Size.2 of [4].

Size Property 1. Null Value (ratio measures). *The size of the empty system is zero. To show this, based on Size Axiom 2, we have $Size(S) = Size(S \cup \oslash) = Size(S) + Size(\oslash)$, from which we obtain $Size(\oslash) = 0$.*

The following property is Axiom Size.1 of [4].

Size Property 2. Nonnegativity (ratio measures). *Given any system S, we have $Size(S) \geq 0$. To prove this, based on Size Axiom 1, we have $Size(S) + Size(S) \geq Size(S \cup S) = Size(S)$, so $Size(S) \geq 0$.*

The empty system has the lowest size.

Size Property 3. Lower Bound (ordinal measures). *Given any system S, $Size(S) \geq Size(\oslash)$. This follows immediately from Size Properties 1 and 2.*

Even though Size Property 3 is somewhat obvious based on the previous ones, it a first example of a property that can be applied to a larger class of measures than ratio scales.

Adding elements to a system cannot decrease its size.

Size Property 4. Monotonicity (ordinal measures). *If the set of elements of a module (or system) m_2 includes the set of elements of another module (or system) m_1, then the size of m_2 is not less than the size of m_1*

$$E_2 \supseteq E_1 \Rightarrow Size(m_2) \geq Size(m_1)$$

To prove this property, since $E_2 \supseteq E_1$, then there must exist a module m_3 such that $E_2 = E_1 \cup E_3$ and $E_1 \cap E_3 = \oslash$. By applying Size Axiom 2, we have that $Size(m_2) = Size(m_1) + Size(m_3)$, which implies that $Size(m_2) \geq Size(m_1)$, since Size Property 2 shows that $Size(m_3) \geq 0$.

This is a second example of a property that applies to ordinal measures. So, one might very well build an ordinal measure that satisfies Size Properties 3 and 4 alone, if one's goal is to build an ordinal measure. Later, this measure may be refined and, based on it, a ratio measure compatible with it and Size Axioms 1 and 2 can be built, for instance when one achieves a greater understanding of how to measure size in a specific context.

In addition, relationships play no role in size.

Size Property 5. Relationships Have No Impact (nominal measures). *Two modules m_1, m_2 with the same set of elements have the same size, i.e., $E_1 = E_2 \Rightarrow Size(m_2) = Size(m_1)$. This result derives from Size Property 4, since $E_1 \supseteq E_2 \Rightarrow Size(m_1) \geq Size(m_2)$ and $E_2 \supseteq E_1 \Rightarrow Size(m_2) \geq Size(m_1)$, so $Size(m_2) = Size(m_1)$.*

In Fig. 3, $Size(m_1) = Size(m_2)$. This property is applicable to nominal measures, and can also be used as an additional "sanity check" of a size measure. If a measure depends on the relationships, then it cannot be classified as a size measure (according to the axioms defined in this paper).

A measure of size is computed as the sum of the "sizes" of its elements.

Size Property 6. Sum (ratio measures). *Given an element e of a system S, let $m^e = < \{e\}, \oslash >$, i.e., m^e is a module with e as its only element and no relationships. Then, we have $Size(S) = \sum_{e \in E} Size(m^e)$. This property is a direct consequence of Size Axiom 2, since all the m^e's are disjoint.*

The axiomatic definition of size we provided is closely related to the axiomatic definition of what is known as "measure" in Measure Theory [17], which is a part of the basis of the theory of differentiation and integration in Calculus.

A real-valued function μ defined on the set of subsets of a set X, on which a σ-algebra Σ is defined, is a measure if it associates a real number with each subset $sub \subseteq X$, with sub belonging to Σ, in such a way that the following three axioms are satisfied.

Measure Axiom 1. $\mu(sub) \geq 0$.

Measure Axiom 2. $\mu(\oslash) = 0$.

Measure Axiom 3. If $sub_1, sub_2, sub_3, ...$ is a countable sequence of pairwise disjoint subsets of X belonging to Σ, $\mu\left(\bigcup_{i=1}^{\infty} sub_i\right) = \sum_{i=1}^{\infty} \mu(sub_i)$.

It can be shown that a function that satisfies these three axioms (i.e., "measure" according to Measure Theory) is a size measure according to our axioms, where set E plays the role of set X and any subset of E belongs to Σ, and, conversely, a size measure according to our axioms is a "measure" according to Measure Theory. To this end, notice that Measure

Axiom 1 mirrors Size Property 2, Measure Axiom 2 mirrors Size Property 1, and Measure Axiom 3 mirrors Size Axiom 2 (note that we always deal with a finite set of elements and therefore subsets of elements, unlike Measure Theory).

According to our approach, it can be shown that these are size measures: LOC, #Statements, #Modules, #Procedures, Halstead's Length , #Occurrences of Operators, #Occurrences of Operands, #Unique Operators, #Unique Operands [9], WMC [7]. Instead, these are not size measures: Halstead's Estimator of length and Volume [9].

3.2 Complexity

By complexity, we here mean "structural" complexity, which is an internal software attribute, and not "psychological" complexity, which is an external software attribute. Complexity is based on the relationships among system elements. In the original proposal, five axioms were used to model complexity. Here, we show that two can be used as the base ones and show how other two become derived properties. We also remove one of the original axioms, as we will discuss a bit in more detail at the end of this Section.

We start with the axiom that characterizes complexity the most. The basic idea is that the complexity of a system is never lower than the sum of the complexities of its parts when they are taken in "isolation," i.e., when they have no relationships in common.

Complexity Axiom 1. Module Composition (ratio measures). The complexity of a system S is not lower than the sum of the complexities of any two of its modules m_1, m_2 whose union is included in S

$$S \supseteq m_1 \cup m_2 \Rightarrow$$
$$Complexity(S) \geq Complexity(m_1) + Complexity(m_2)$$

In Complexity Axiom 1, the two modules m_1 and m_2 may have elements in common. Thus, in the overall system S to which m_1 and m_2 belong, the elements of m_1 and m_2 may still be linked via "paths" of relationships, i.e., transitive relationships if R is a binary relation. This is the rationale for the '$\geq$' sign, since these additional, transitive relationships cannot be accounted for by the relationships that exist in the single modules in isolation. For instance, in Fig. 1, $Complexity(m_1) \geq Complexity(m_2) + Complexity(m_3)$.

Additivity holds when a system is made up of two unconnected modules.

Complexity Axiom 2. Unconnected Module Additivity (ratio measures). The complexity of a system S composed of two unconnected modules m_1, m_2 is equal to the sum of the complexities of the two modules

$$S = m_1 \cup m_2 \wedge$$
$$m_1 \cap m_2 = \oslash \wedge OuterR(m_1) \cap OuterR(m_2) = \oslash \Rightarrow$$
$$Complexity(S) = Complexity(m_1) + Complexity(m_2)$$

Thus, $Complexity(m_4 \cup m_6)$ is equal to $Complexity(m_4) + Complexity(m_6)$ in Fig. 1.

We now describe properties that can be derived from the previous Complexity Axioms. No relationships imply zero complexity (which was Axiom Complexity.2 of [4]).

Complexity Property 1. Null Value (ratio measures). *Let $m_1 =< E_1, \oslash >$ denote a module with no relationships. We have $Complexity(m_1) = 0$. To prove this property, we first prove the spacial case $Complexity(\oslash) = 0$. Given any system S, we have $S = S \cup \oslash$. By applying Complexity Axiom 2, since S and $\oslash$ are obviously unconnected, we have $Complexity(S) = Complexity(S \cup \oslash) = Complexity(S) + Complexity(\oslash)$, so $Complexity(\oslash) = 0$. Now, thanks to Complexity Axiom 1, since $m_1 \supseteq \oslash \cup \oslash$, we have $Complexity(m_1) \geq 0$. On the other hand, $m_1 \supseteq m_1 \cup m_1$, so we have $Complexity(m_1) \geq 2 \cdot Complexity(m_1)$, thanks to Complexity Axiom 1. So, $0 \geq Complexity(m_1)$, and $Complexity(m_1) = 0$.*

The lower bound of complexity is attained when there are no relationships in a system (Axiom Complexity.1 of [4]).

Complexity Property 2. Lower Bound (ordinal measures). *The complexity of a system $S_2 =< E_2, R_2 >$ is not lower than the complexity of a system $S_1 =< E_1, \oslash >$ with no relationships, i.e.,*

$$Complexity(S_2) \geq Complexity(S_1)$$

To prove this, based on Complexity Axiom 1, $S_2 \supseteq \oslash \cup \oslash$, so $Complexity(S_2) \geq 2 \cdot Complexity(\oslash) = 0 = Complexity(S_1)$.

We can also show the monotonic behavior of complexity with the multiset of relationships.

Complexity Property 3. Monotonicity (ordinal measures). *Adding relationships to a system cannot decrease its complexity. To show this, suppose that a new relationship $<< e_1, e_2, t >, 1 >$ is added between two elements e_1, e_2 of an existing system S, to obtain a new system $S' = S \cup m \supset S$, where $m =< \{e_1, e_2\}, \{<< e_1, e_2, t >, 1 >\} >$, i.e., it is the module whose elements are e_1 and e_2 with the only relationship. As $S' \supseteq S \cup m$, Complexity Axiom 1 shows that $Complexity(S') \geq Complexity(S) + Complexity(m)$, and Complexity Property 2 shows that $Complexity(m) \geq 0$, we have*

$$Complexity(S') \geq Complexity(S)$$

We now show that complexity depends on relationships and not on elements.

Complexity Property 4. Adding Elements (Nominal Measures). *Adding elements to a system does not change its complexity. To show this, suppose that a new element e is added to the existing ones of a system S_1, so that the new system S_2 is obtained. Adding this new element means that $S_2 = S_1 \cup m^e$, where $m^e =< \{e\}, \oslash >$ is a module with e as its only element and no relationships. We also have that $S_1 \cap m^e = \oslash$, so Complexity Axiom 2 shows that $Complexity(S_2) = Complexity(S_1) + Complexity(m^e)$. Since $Complexity(m^e) = 0$ because of Complexity Property 1, we have that $Complexity(S_2) = Complexity(S_1)$.*

These measures may be classified as complexity measures, according to the above axioms: Oviedo's data flow complexity measure DF [16], $v(G) - p$, where $v(G)$ is McCabe's cyclomatic number and p is the number of connected components in a control-flow graph [13]. These measures conflict with the above axioms: Henry and Kafura's information flow complexity measure [10], RFC and LCOM [7].

As a final remark, it must be noted that Axiom Complexity.3 in [4] does not appear here. That axiom stated that the complexity of a system does not change if all the binary relationships between the elements of the system are reversed. For instance, it is simply a notational convention that is used in Fig. 2 to state that class N "inherits from" class M and that class M "aggregates" class Q. Equivalently, one could have reversed the relationships (and, graphically, the arrows) by saying that M "generalizes" class N and that class Q "is aggregated in" class M without influencing the structural complexity of the system. However, this axiom does not seem to influence the building or verification of complexity measures, so we drop it here.

3.3 Cohesion

Cohesion is related to the *degree* with which the elements of a module are tied to each other. For space reasons, unlike in [4], we only deal with the cohesion of a single module, and not also with the cohesion of an entire system composed of disjoint modules.

Cohesion Axiom 1. Upper Bound (ordinal measures). The cohesion of a module m is not greater than a specified value Max, i.e., $Cohesion(m) \leq Max$.

Adding relationships does not decrease module cohesion.

Cohesion Axiom 2. Monotonicity (ordinal measures). Let modules $m_1 =< E, R_1 >$, $m_2 =< E, R_2 >$ be two modules with the same set of elements E, and let $R_1 \subseteq R_2$. Then, $Cohesion(m_1) \leq Cohesion(m_2)$.

The cohesion of a module obtained by putting together two unrelated modules is not greater than the maximum cohesion of the two original modules.

Cohesion Axiom 3. Unconnected Modules (ordinal measures). Let m_1 and m_2 be two unconnected modules, then,

$$max\{Cohesion(m_1), Cohesion(m_2)\} \geq Cohesion(m_1 \cup m_2)$$

Thus, in Fig. 1, we have

$$Cohesion(m_4 \cup m_6) \leq max\{Cohesion(m_4), Cohesion(m_6)\}$$

All of the above axioms may be used for ordinal measures, not necessarily interval or ratio ones. For instance, one may define a discrete cohesion measure (e.g., Constantine and Yourdon's [23]) as an ordinal scale.

The original set of axioms also includes the following one, which may be satisfied by ratio measures.

Cohesion Axiom 4. Null Value (ratio measures). The cohesion of a module with no relationships $m =< E, \oslash >$ is null, i.e., $Cohesion(m) = 0$.

The following property derives from the above axioms.

Cohesion Property 1. Lower Bound (ordinal measures). *The cohesion of a module m_2 is not lower than the cohesion of a module $m_1 =< E_1, \oslash >$ with no relationships, i.e., $Cohesion(m_2) \geq Cohesion(m_1)$. By contradiction, suppose that $m_2' =< E_2, \oslash >$ is a module with the same set of elements as m_2, such that $Cohesion(m_2) < Cohesion(m_2')$. This is impossible, since Cohesion Axiom 2 states that adding relationships to a module (m_2' in our case) cannot decrease its cohesion.*

These measures may be classified as cohesion measures, according to the above axioms: PRCI, NRCI, ORCI [6].

3.4 Coupling

Coupling is about the *amount* of relationships between the elements of a module and the elements of the other modules. For space reasons, unlike in [4], we only deal with the coupling of a single module, and not also with the coupling of an entire system when it is composed of disjoint modules.

The following axiom for coupling measures replaces and integrates an existing one (Axiom Coupling.3 of [4]). The rationale is that there are two components to coupling. One is related to the connections between a module and the rest of the system, and that was already captured by Axiom Coupling.3 of [4]. The other component of coupling is related to the connection among the elements within the module. For instance, m_2 in Fig. 3(b) is more coupled with the remainder of the system (which is not shown), as its elements (specifically C) are more connected to the remainder of the system than the elements of m_1 in Fig. 3(a).

Coupling Axiom 1. Monotonicity (ordinal measures). Adding a new relationship to a module m_1 or to its set of outer relationships $OuterR(m_1)$ does not decrease its coupling. So, if m_2 is a module such that $E_2 = E_1$, we have

$$OuterR(m_2) \supseteq OuterR(m_1) \wedge R_2 \supseteq R_1 \Rightarrow Coupling(m_2) \geq Coupling(m_1)$$

However, a module with no outer relationships has zero coupling, regardless of its internal relationships.

Coupling Axiom 2. Null Value (ratio measures). The coupling of a module with no outer relationships is null.

When two modules are merged in a single module, the relationships that connect them to each other become internal ones, so the coupling of the new module is not higher than the sum of the couplings of the two modules. In Fig. 1, when modules m_2 and m_3 are merged into module m_1 the relationships to and from m_4, m_5, and m_6 are still outer relationships for m_1, but the relationships between m_2 and m_3 have become internal relationships for m_1 (so, they may also contribute to the cohesion of m_1).

Coupling Axiom 3. Merging of Modules (ratio measures). The coupling of the union of two modules m_1, m_2 is not greater than the sum of the couplings of the two modules

$$Coupling(m_1 \cup m_2) \leq Coupling(m_1) + Coupling(m_2)$$

Additivity holds when a system is made up of two unconnected modules.

Coupling Axiom 4. Unconnected Modules (ratio measures). The coupling of the union of two unconnected modules is equal to the sum of their couplings

$$m_1 \cap m_2 = \oslash \wedge OuterR(m_1) \cap OuterR(m_2) = \oslash \Rightarrow$$
$$Coupling(m_1 \cup m_2) = Coupling(m_1) + Coupling(m_2)$$

So, $Coupling(m_4 \cup m_6) = Coupling(m_4) + Coupling(m_6)$ in Fig. 1.

The following coupling properties can be derived.

Coupling Property 1. Nonnegativity (ratio measures). *The coupling of a module m_1 is nonnegative. To this end, let m_2 denote any module such that $m_2 \supseteq m_1$, so $m_1 \cup m_2 = m_2$. Then, we have $Coupling(m_1) + Coupling(m_2) \geq Coupling(m_2)$, from which $Coupling(m_1) \geq 0$.*

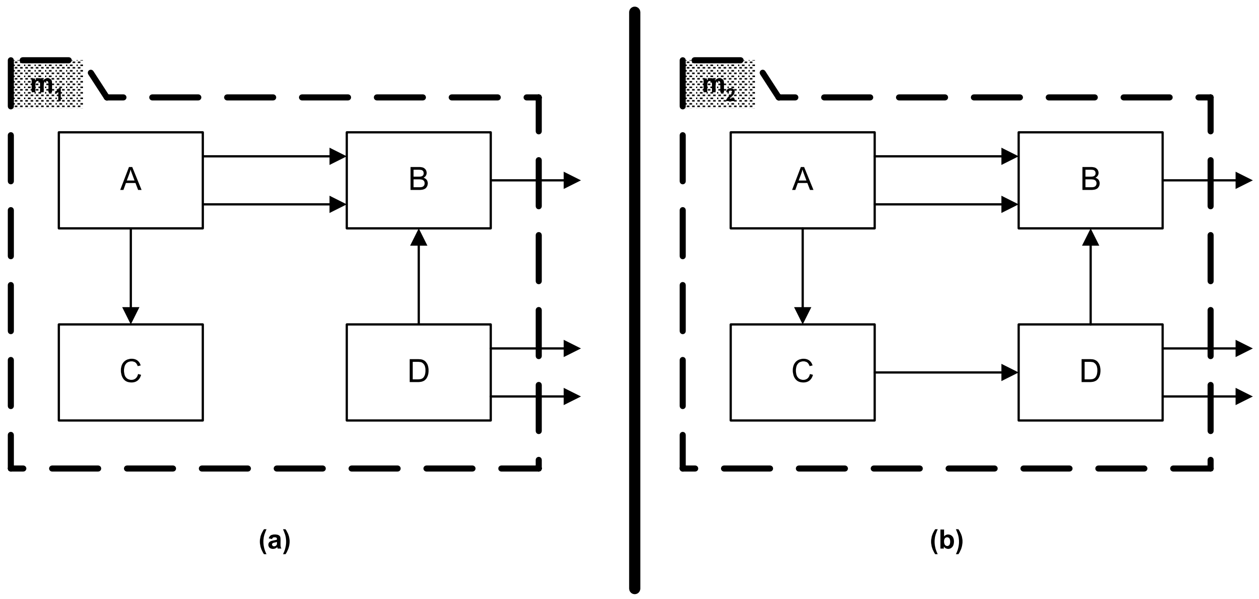

Figure 3: Adding Relationships to a Module

As a direct further implication, we have the following property, which may be satisfied by measures that are on an ordinal level of measurement.

COUPLING PROPERTY 2. LOWER BOUND (ORDINAL MEASURES). *The coupling of a module is not less than the coupling of a module with no outer relationships.*

These measures may be classified as coupling measures, according to the above axioms: TIC and DIC [6], CBO and RFC [7]. Fenton's coupling measure [8] does not satisfy the above axioms.

4. RELATIONSHIPS BETWEEN SOFTWARE ATTRIBUTES

Here, we examine the relationships between three relevant pairs of attributes, to show their similarities, differences, and relationships.

4.1 Size vs. Complexity

The analysis of the base axiom sets shows the differences between size and complexity:

- size is based on elements, while complexity is based on relationships
- the structure of Size Axiom 1 is different from the structure of Complexity Axiom 1, since the size of the union of two modules is not greater than the sum of the sizes of the modules, while the complexity of their union is not smaller than the sum of their complexities. In other words, complexity cannot be interpreted as the amount of relationships, as if it was the "size" of the set of relationships. If this was the case, Complexity Axiom 1 would be of the same kind as Size Axiom 1, while the difference between these two axioms also lies in the direction of the inequality.

On the other hand, the structure of Size Axiom 2 is analogous to the structure of Complexity Axiom 2, i.e., both size and complexity are additive under similar (but not identical) conditions. Summarizing, there is a difference between the two attributes, even though size measures have been interpreted as complexity measures in the past. This misunderstanding has been due to a few reasons.

First, when a new element e is added to an existing system $S =< E, R >$, the newly created system $S' =< E', R' >$ is likely to have a larger multiset of relationships, as S' still has all of the relationships in S, plus additional relationships that may be created between e and the existing elements of system S. Thus, the introduction of e is likely to cause an increase in size and the newly introduced relationships are likely to cause an increase in complexity. There is no difference in the complexities of the two systems if e is totally isolated in S' (see Complexity Axiom 4), i.e., no additional relationships are introduced to relate e with the existing elements. However, this is unlikely to happen, as the purpose of adding a new element to a system is to make it cooperate with the existing elements. No increase in complexity occurs also if new relationships are added in S' to relate e with the existing elements, but none of these relationships are of the kind that may affect the kind of complexity we are interested in measuring. Even in this case, the introduction of e does not cause a decrease of complexity. Conversely, even if e does not cause an increase of size, because e is not an element that contributes to a specific kind of size we are interested in measuring (e.g., data size, executable statement size, functional size), the complexity of S' is not lower than the complexity of S. In this respect, the definition of McCabe's cyclomatic complexity [13] $v(G) = |R| - |E| - 2$ for a control-flow graph may be viewed as an attempt to remove the effect of size (measured by $|E|$) from the complexity (measured by $|R|$). An analysis like the one of this paper helps pinpoint the differences between even related software attributes.

Second, there may be an association or even a correlation between measures of size and measures of complexity in practical applications, and this may have caused a misleading use of the term "indicator." For instance, a size measure like Lines Of Code (LOC) has been used as an indicator of several external software attributes, like maintainability, or internal ones, like complexity. This has often led to calling LOC a measure of maintainability or complexity. While there may very well be a relationship between LOC and a complexity measure, as that complexity measure increases when LOC increases, by the same token, one may say that *it is likely* that the weight of an object increases when the volume the object increases, though one would not call a measure of volume a measure of weight, especially when dealing with the weight of objects made of different materials.

Third, some size measures may have an interpretation that is close to that of complexity measures. For instance, take LOC, which counts the number of lines that are physically present in a software program. Suppose that the program is modeled by a system where each line of code is modeled by an element and the fact that one line of code is physically followed by another line of code is modeled by a relationship. Note that this is not the programs' control-flow graph, but simply a graph that describes the physical sequence among the lines of code. LOC is a size measure according to the axioms of Section 3.1; the number of relationships of this system is a complexity measure according to the axioms of Section 3.2. The number of relationships is also equal to $LOC-1$, so one may interpret LOC as a complexity measure too. Cyclomatic complexity [13] provides a quantification of how much the control flow of a program deviates from a pure sequence. Only decision points contribute to $v(G)$: a k-way decision point contributes $k-1$ to $v(G)$.

4.2 Complexity vs. Cohesion

Both complexity and cohesion of a module

- depend on the relationships within the module
- are null when there are no relationships in the module
- increase when a relationship is added to the relationships of the module.

Based on these similarities, one may wonder whether there is some sort of link between the two attributes, and specifically between the measures of the two attributes. We now show that complexity measures for a module may be used to define cohesion measures. For instance, if $cx(m)$ is a complexity measure for module m and $cx_M(m)$ is its maximum value for module m, then $ch(m) = cx(m)/cx_M(m)$ is a cohesion measure. At any rate, we now prove a more general result. To this end, let $f(x, y)$ denote a real-valued function, which will be here used to build a cohesion measure based on two values x and y of a complexity measure. Suppose that $0 \leq f(x, y) \leq Max$, that $f(0, y) = 0$ and, whenever $x \leq y$, we have

$$x_2 \geq x_1 \Rightarrow f(x_2, y) \geq f(x_1, y) \tag{1}$$

and

$$\begin{aligned} &\Delta x \leq \Delta y \Rightarrow \\ &(f(x+\Delta x, y+\Delta y) \leq f(x, y) \vee \\ &f(x+\Delta x, y+\Delta y) \leq f(\Delta x, \Delta y)) \end{aligned} \tag{2}$$

Then, we have that $ch(m) = f(cx(m), cx_M(m))$ is a cohesion measure, as we now show by proving that the Base Axioms of cohesion hold under this transformation of $cx(m)$ and $cx_M(m)$.

Upper Bound. It is straightforward to prove that

$$0 \leq ch(m) \leq Max$$

Monotonicity. Adding one relationship to module $m_1 =< E_1, R_1 >$ produces a new module $m_2 =< E_1, R_2 >$ such that $cx(m_2) \geq cx(m_1)$, because of Complexity Property 3. As both modules have the same set of elements E_1, they also have the same value of maximum complexity. Thanks to Equation (1), if $cx(m_2) \geq cx(m_1)$, we also have that

$$\begin{aligned} ch(m_2) = f(cx(m_2), cx_M(m_2)) \geq \\ f(cx(m_1), cx_M(m_2)) = ch(m_1) \end{aligned}$$

Unconnected Modules. If m_1 and m_2 are two unconnected modules and $m_{12} = m_1 \cup m_2$, then $cx(m_{12}) = cx(m_1) + cx(m_2)$, because of Complexity Axiom 2. Since $cx_M(m_{12}) \geq max\{cx_M(m_1), cx_M(m_2)\}$, based on Equation (2), we have

$$\begin{aligned} ch(m_{12}) = f(cx(m_1) + cx(m_2), cx_M(m_{12})) \leq \\ max\{f(cx(m_1), cx_M(m_{12})), f(cx(m_2), cx_M(m_{12}))\} \leq \\ max\{f(cx(m_1), cx_M(m_1)), f(cx(m_2), cx_M(m_2))\} = \\ max\{ch(m_1), ch(m_2)\} \end{aligned}$$

Null Value. Based on the definition of function f, it is immediate to prove that $R = \oslash \Rightarrow cx(m) = 0 \Rightarrow ch(m) = 0$.

It can be shown that $f(x, y) = x/y$ satisfies conditions 1 and 2. Thus, given any complexity measure with an upper bound, the measure $ch(m) = cx(m)/cx_M(m)$ is actually a cohesion measure, like the ones (NRCI, ORCI, PRCI) in [6].

Note that, given a module, a maximum value for its complexity always exists. Maximum complexity is attained when the module's elements are connected by the largest multiset of relationships that can be had based on the set of elements. Since each module is composed of a finite number of elements, there obviously is a finite number of multisets of relationships in the module. In turn, the multiset of relationships in the module. In turn, the multiset of relationships between two elements is a finite one (as per Definition 1), so the largest set of relationships of a module exists, and it has maximum complexity, thanks to Complexity Property 3.

This analysis and this result are hardly just theoretical ones. The above relationship between complexity and cohesion shows that cohesion may increase when complexity increases. This circumstance might explain why sometimes cohesion measures are not very well related to fault-proneness [2]. On the one hand, cohesion increases, and this is supposed to lead to less error-prone modules. On the other hand, complexity increases as well, and this is supposed to lead to more error-prone modules. Thus, the positive effect of the increase in cohesion on error-proneness is somewhat masked by the negative effect of an increase in complexity. This may explain why cohesion measures have sometimes been weakly related to important, tangible measures like for instance the probability of having a fault in a software module. For instance, in [6], it is shown that a cohesion measure built as the ratio between the number of the so-called "interactions" relationships among the elements of the specification of an Ada module and the maximum number of such interactions is a predictor for module fault proneness, but not for all the Ada systems analyzed and it is the weakest predictor among the ones that were taken into account.

Also, an equation like $ch(m) = cx(m)/cx_M(m)$ may used as a starting point to find relationships among quantities, as is usual in many scientific disciplines.

4.3 Complexity vs. Coupling

Coupling has often been associated with complexity, and sometimes has been called a "component" of complexity. Both complexity and coupling of a module

- are null when there are no relationships in the module and outside it
- increase when a relationship is added to the relationships of the module.

However, there is one major difference between complexity and coupling. When two disjoint modules are merged in a module, the complexity of the resulting module is not less than the sum of the complexities of the original modules, while the coupling of the resulting module is not greater than the sum of the couplings of the original modules.

5. FUTURE WORK

Future work will address

- further relationships among internal software attributes
- investigating the impact of hierarchical graph abstraction/refinement operations on the properties of internal software attributes
- the definition of external software attributes
- the relationships between internal and external software attributes.

6. ACKNOWLEDGMENTS

The research presented in this article was partially funded by the IST project QualiPSo, sponsored by the EU in the 6th FP (IST-034763); the FIRB project ARTDECO, sponsored by the Italian Ministry of Education and University; and the project "La qualità nello sviluppo software," sponsored by the Università degli Studi dell'Insubria.

7. REFERENCES

[1] E.B. Allen, T.M. Khoshgoftaar, "Measuring Coupling and Cohesion: An Information-Theory Approach," IEEE METRICS 1999, pp. 119 - 129, Boca Raton, FL, USA, Nov. 4-6 1999.

[2] L.C. Briand , J.W. Daly , J. Wuest, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems," Empirical Software Engineering, v.3 n.1, p.65-117, 1998

[3] L.C. Briand , J.W. Daly , J. Wuest, "A Unified Framework for Coupling Measurement in Object-Oriented Systems," IEEE TSE, Vol.25 No.1, pp.91-121, Jan. 1999.

[4] L. Briand, S. Morasca, V. Basili, "Property-based software engineering measurement," IEEE TSE, Vol. 22 No. 1, pp.68-86, January 1996.

[5] L. Briand, S. Morasca, V. Basili, "Response to: "Comments on "Property-Based Software Engineering Measurement: Refining the Additivity Properties,"""" IEEE TSE, Vol. 23 No. 3, pp. 196-197, Mar. 1997.

[6] L. Briand, S. Morasca, V. Basili, "Defining and Validating Measures for Object-based High-Level Design," IEEE TSE, Vol. 25, No.5, pp. 722 - 741, Sep.-Oct. 1999.

[7] S.R. Chidamber, C. Kemerer, "A metrics suite for obiect oriented design," IEEE TSE, Vol. 20, No. 6, pp. 476-493, June 1994.

[8] N. Fenton, "Software Metrics, A Rigorous Approach." Chapman and Hall, 1991.

[9] M.H. Halstead, "Elements of Software Science." Elsevier North- Holland, 1977.

[10] S. Henry, D. Kafura, "Software structure metrics based on information flow," IEEE TSE, vol. 7, no. 5, pp. 510-518, Sept. 1981.

[11] S. Kramer, H. Kaindl, "Coupling and cohesion metrics for knowledge-based," ACM TOSEM, Vol. 13 , Issue 3, pp. 332 - 358, July 2004.

[12] K.B. Lakshmanian, S. Jayaprakash, and P.K. Sinha, "Properties of control-flow complexity measures," IEEE TSE, vol. 17, no. 12, pp. 1,289-1,295, Dec. 1991.

[13] T.J. McCabe, "A complexity measure," IEEE TSE, vol. 2, no. 5, pp. 308-320, Apr. 1976.

[14] S. Morasca, "Measuring Attributes of Concurrent Software Specifications in Petri Nets," IEEE METRICS 1999, pp. 100 - 110, Boca Raton, FL, USA, Nov. 4-6 1999.

[15] S. Morasca, L.C. Briand, "Towards a theoretical framework for measuring software attributes," IEEE METRICS 1997, pp. 119 - 126, Albuquerque, NM, USA, Nov. 5-7, 1997.

[16] E.I. Oviedo, "Control flow, data flow and program complexity," Proc. IEEE COMPSAC, pp. 146-152, Nov. 1980.

[17] E. Pap, "Some elements of the classical measure theory," in Handbook of Measure Theory (ed. E. Pap), Elsevier, 2002.

[18] G. Poels, G. Dedene, "Comments on "Property-Based Software Engineering Measurement: Refining the Additivity Properties,"" IEEE TSE, Vol. 23, no. 3, pp. 190-195, Mar. 1997.

[19] G. Poels, G. Dedene, "Distance-based software measurement: necessary and sufficient properties for software measures," Information and Software Technology, Vol. 42, no. 1, pp. 35-46, 2000.

[20] R. E. Prather, "An axiomatic theory of software complexity measure," The Computer Journal, Vol. 27, no. 4, pp. 340-346, 1984.

[21] F. S. Roberts, *Measurement Theory*, Addison-Wesley, Reading, 1979.

[22] E.J. Weyuker, "Evaluating software complexity measures," IEEE TSE, Vol. 14, No. 9, pp. 1357-1365, Sept. 1988.

[23] E. Yourdon, L.L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Yourdon Press, 1979.