Towards a Framework for Software Measurement Validation

Barbara Kitchenham, Shari Lawrence Pfleeger, *IEEE Computer Society*, and Norman Fenton, *Member, IEEE Computer Society*

***Abstract*—In this paper we propose a framework for validating software measurement. We start by defining a measurement structure model that identifies the elementary component of measures and the measurement process, and then consider five other models involved in measurement: unit definition models, instrumentation models, attribute relationship models, measurement protocols and entity population models. We consider a number of measures from the viewpoint of our measurement validation framework and identify a number of shortcomings; in particular we identify a number of problems with the construction of function points. We also compare our view of measurement validation with ideas presented by other researchers and identify a number of areas of disagreement. Finally, we suggest several rules that practitioners and researchers can use to avoid measurement problems, including the use of measurement vectors rather than artificially contrived scalars.**

***Index Terms*—Measurement theory, software measurement, software metrics validation.**

I. Introduction

As software engineering matures, software measurement plays an increasingly important role in understanding and controlling software development practices and products. Consequently, it is essential that the measures we use are valid. That is, measures must represent accurately those attributes they purport to quantify. So validation is critical to the success of software measurement.

In the past few years, there have been a number of papers addressing the issue of validating software metrics. A feature of these papers is their diversity. Schneidewind [16] recommends an empirical validation process in which a software metric is validated by showing that it is associated with some other measure of interest. Weyuker [20] restricts her discussion to complexity metrics and suggests that evaluation be performed by identifying a set of desirable properties that measures should possess and determining whether or not a prospective metric exhibits those properties. Fenton [4] and Melton et al. [13] suggest that a valid metric must obey the Representation condition of measurement theory, so that intuitive understanding of some attribute is preserved when it is mapped to a numerical relation system. Fenton and Kitchenham [6] discussed two different views of validation: one based on identifying the usefulness of a measure for predictive purposes, the other on identifying the extent to which a measure characterizes a stated attribute.

What has been missing so far is a proper discussion of relationships among the different approaches, and how they should be used in practice. Furthermore, it is not clear which approaches actually lead to a widely accepted view of validity. For example, Chidamber and Kemerer [3] refer to Weyuker's properties to discuss their measures for object oriented designs, while Zuse [21] claims that at least two of Weyuker's properties are inconsistent. This situation means that new measures are being justified according to disputed criteria, and some commonly-used measures may not in fact be valid according to any widely accepted criteria.

It is our intention to overcome these problems by proposing a validation framework. Such a framework can help researchers and practitioners to understand:

- how to validate a measure;
- how to assess the validation work of others;
- when it is appropriate to apply a measure in a given situation.

A full, practical framework is an ambitious goal that requires input from practitioners and the research community. To encourage a broad discussion, this paper considers some of the concepts needed to develop a validation framework. Our framework is based on identifying the elements of measurement and their properties, identifying how we define those elements when we construct a measure, and defining appropriate theoretical and empirical methods of validating those properties and definition models. In Section II, we describe a structure model of software measurement intended to introduce the elements of measurement and their relationships with one another. This section identifies the properties of individual elements and what those properties mean for measurement validation. In Section III, we discuss the models we use to define the elements in the structure model when we create/apply a measure. We need to be sure our definition models are valid if we want our measures to be valid. This is a particular problem for indirect measures and much of the discussion in Section III concentrates on such measures. In particular we discuss the issue of vector and scalar attributes. In software measurement we are usually keen to convert a set of simple measures into a new indirect measure; however such measures are extremely difficult to validate. This section looks in detail at the issue of the construction of function points and con-

Manuscript received March 1995; revised September 1995.

B. Kitchenham is with National Computing Centre, Oxford House, Oxford Rd., Manchester M1 7ED, England; e-mail: barbara.kitchenham@ncc.co.uk.

S.L. Pfleeger is with Systems/Software, Inc., 4519 Davenport St. NW, Washington, DC 20016-4415, USA; e-mail: slpfleeger@aol.com.

N. Fenton is with the Centre for Software Reliability, City University, Northampton Sq., London EC1V OHB, England; e-mail: n.fenton@city.ac.uk.

IEEECS Log Number S95043.

cludes that both the Albrecht [1] and Mark II [19] versions are invalid. In Section IV, we attempt to summarise the issues involved in validation from the viewpoint of the properties of measurement elements identified in Section II and properties of definition models introduced in Section III. We identify a number of validation methods and the aspects of measurement they apply to. In addition, we review other work on measurement validation and compare it with our own.

Throughout this paper we illustrate our points by referring to measurement activities both in software engineering and in other domains, because we believe that software measurement must be consistent with measurement in other disciplines. Indeed, any validation framework that contradicts measurement principles that apply to other disciplines would itself be invalid. In addition, we apply the framework to a number of measures used in the software domain. This shows that a range of simple measures are valid within well-defined contexts, but also shows that certain measures cannot be deemed to be valid according to any reasonable scientific notion. Thus, the framework has an important and practical role to play in moderating our use of software measures.

We encourage researchers and practitioners to respond critically to our ideas, because, as a community, we must arrive at a common consensus about proper validation. Currently, some software measurement researchers choose a validation method by reference to authority (i.e., choosing the approach they want to use with reference to a specific author's viewpoint) rather than to standard scientific principles. In our view, unless the software measurement community can agree on a valid, consistent, and comprehensive theory of measurement validation, we have no scientific basis for the discipline of software measurement, a situation potentially disastrous for both practice and research.

II. The Structure of Measurement

A structural model of software measurement allows us to describe the objects involved in measurement and their relationships. Such a model is shown in Fig. 1. In this section we describe all the elements of the model and discuss how they contribute to measurement. We use an existence argument to confirm that the stated relationships are necessary for measurement in general and software measurement in particular. However, we make no claims that the model is sufficient.

A. Entities, Attributes, and Their Relationships

A.1. Entities

Entities are the objects we observe in the real world. One of the goals of measurement is to capture their characteristics and manipulate them in a formal way. Software entities may be products, processes, or resources of different types.

A.2. Attributes

Attributes are the properties that an entity possesses. For a given attribute, there is a relationship of interest in the empirical world that we want to capture formally in the mathematical world. For example, if we observe two people we can say that one is taller than the other. A measure allows us to captures the "is taller than" relationship and map it to a formal system, enabling us to explore the relationship mathematically. In physics properties are often multi-dimensional. Multi-dimensional attributes can be *vectors*, e.g., velocity which involves speed and direction or *scalars*, e.g., speed which is measured in terms of distance per time period. These distinctions are often ignored in software measurement, but we believe that many measurement problems could be overcome if they were properly understood.

A.3. The Relationship Between Entities and Attributes

Fig. 1 suggests that an entity possesses many attributes, while an attribute can qualify many different entities. These relationships can be confirmed by example. To see that an entity can have many attributes, consider a program as a software entity which can exhibit attributes such as length, structure, and correctness.

In addition, an attribute may apply to one or more different entities. Just as height applies to human beings, mountains, and houses, productivity can apply to several different software entity classes such as individual software developers, teams or projects.

B. Units and Scale Types and Their Relationships

B.1. Units

A measure maps an empirical attribute to the formal, mathematical world. A measurement unit determines how we measure an attribute, and Fig. 1 implies that an attribute may be measured in one or more units. For example, you may use different units to measure temperature (e.g., Fahrenheit, Celsius, or Kelvin). Likewise, code length might be measured by counting the lines of code or the lexical tokens in a program listing. Fig. 1 also states that the same unit may be used to measure more than one attribute. For example, fault rate may be used to measure to program correctness or test case effectiveness.

B.2. Scale Types

When we consider measurement units, we need to understand the different measurement scale types implied by the particular unit. The most common scale types are: nominal, ordinal, interval and ratio. A unit's scale type determines the admissible transformations we can apply when we use a particular unit [5].

In classical measurement theory, units are only applicable to ratio and interval scale measures. We have extended the use of units in our structure model to allow for the definition of the scale points for ordinal scale measures and the categories used for nominal scale measures. For example, if we are defining fault categories as major, minor and negligible, we need to define these terms in more detail if different data collectors are going to use the terms consistently. In this case our "units" would be the description of major (e.g., a fault that result in a software failure), minor (e.g., a fault that results in misleading or unhelpful outputs to the user), and negligible (e.g., a code structure that conflicts with standard coding practices). Thus, in the context of nominal and ordinal scale measures where our measures are mappings to arbitrary labels, we suggest a "unit" is needed to ensure that such measures are used consistently.

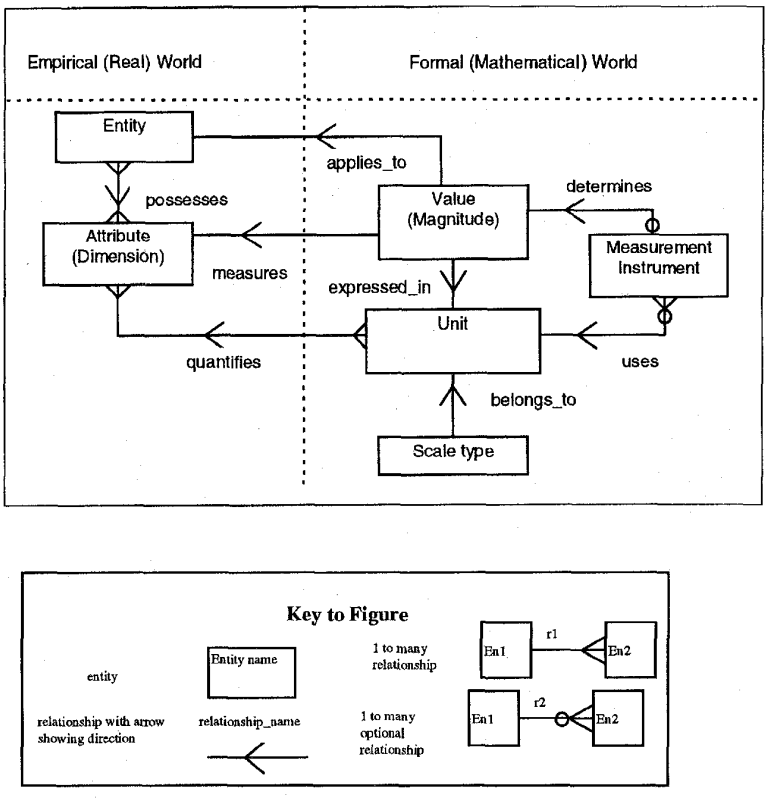

Fig. 1. A structural model of measurement.

B.3. The Relationship Between Units and Scale Types

Fig. 1 identifies a one-to-one relationship between unit and scale type; however we cannot offer a proof of this relationship. Rather we can confirm by example that the scale type is inherent in the unit not the attribute. For example, Fahrenheit and Celsius are interval scale units of temperature, whereas Kelvin is a ratio scale unit of temperature. Thus, although the different units lead to different scale types, they do not affect the attribute.

C. Values

When we measure an attribute, we do so by applying a specific measurement unit to a particular entity and attribute to obtain a value. This value is often numerical, but it does not have to be. For example, a module can be labeled "inspected" or "not inspected," or a defect can be categorised as "requirements fault," "design fault," "code fault," or "documentation fault". However, for convenience, we often map (and always *can* map) to numbers. Thus, the set {not inspected, inspected} can be mapped to the set {0, 1} or equivalently to the set {101, 100} and the fault categories can be mapped to the set {0, 1, 2, 3}. However, since these values represent nominal scale measures, they are *arbitrary labels*; so they cannot be summed or averaged.

A measured value cannot be interpreted unless we know to what entity it applies, what attribute it measures and in what unit. Just as a price is always associated with a specific item, and a unit of currency (e.g., dollars, guilders, pounds), so must a software attribute have both an entity and a unit of measure; one or two without the third are meaningless.

D. Properties of Values

We expect valid measures to be defined over a set of permissible values; for example, length in lines of code is defined on the non-negative integers. A set of permissible values may be finite or infinite, bounded or unbounded, discrete or continuous. Nominal and ordinal scale measures are usually discrete (i.e., map to integers), whereas interval and ratio measures can be continuous or discrete. For example, length in lines of code is a discrete ratio-scale measure, and temperature in degrees Kelvin is a continuous ratio-scale measure.

E. Measurement Instrument

Our model shows that an instrument may optionally be used to obtain the measured value of an attribute. For example, we can use a thermometer to measure temperature, or a software program to count the number of lines of code in a program. Fig. 1 indicates that there may be many different measurement instruments available for a particular unit. For example, we can measure height by using either a tape measure or variations in air pressure. Measurement instruments usually detect a single (unit) value of an attribute in a particular unit of measurement and accumulate units into a value for a particular en-

tity. However, measurement instruments are also used to classify entities. For example, we might use a genetic test as a measurement instrument to determine the sex of an athlete.

F. Indirect Measures

We often obtain measures from equations involving other measures. Such measures are called indirect measures. The model presented in Fig. 1 is appropriate for direct measures but cannot cope with indirect measures. Fig. 2 and Fig. 3 are needed to represent indirect measures. The equation defining an indirect measure acts as a form of measurement instrument. Fig. 2 shows an equation that is based on an empirically observed association between attributes that we formalise as a mathematical equation (e.g., when we use program size in an equation to predict project effort). The attribute(s) we use in an equation may relate to an entity or entities different from the entity whose attribute we want to measure indirectly. For example, although we may use an equation to predict effort from size, size is a product attribute whereas effort is a process or a resource attribute.

Fig. 3 shows a measure that is based on a compound measurement unit (e.g., when we use lines of code per hour as a unit for measuring productivity). In this case, the equation is derived from the unit of the new attribute.

G. Compound Units

In the case of scalar measures that are expressed in compound units, it is usually not possible to measure the multi-dimensional attribute directly (e.g., we cannot measure productivity in lines of code per hour except by dividing size by effort using appropriate units for size and effort). Fig. 3 shows that a multi-dimensional attribute is derived from several other attributes and measured in a compound unit constructed from relevant base units (e.g., "lines of code per hour" is constructed from the unit "lines of code" and the unit "hour"). The equation used to calculate the indirect attribute value is derived from the nature of the multi-dimensional attribute not from any empirical association among the attributes.

H. Properties of Indirect Measures

Valid indirect measures should not exhibit unexpected discontinuities; that is, they should be defined in all reasonable or expected situations. Thus:

$$Measure1 = \frac{Count1}{Count2}$$

may present problems if $Count2 = 0$ or

$$Measure1 = \frac{Count1}{Count2 - n}$$

may be invalid if $Count2 = n$.

Fig. 2 and Fig. 3 imply that a multi-dimensional attribute is represented as a scalar indirect measure. Although many compound measures in software are treated as scalars, this is not the case in other disciplines. For example, *volume* is a scalar value but *shape* is not; so we can speak of a box with *shape* 2 metres high by 4 metres wide by 6 metres long as having a *volume* of 48 cubic metres. Similarly, we measure *position* as a vector (e.g., latitude and longitude for position on the Earth, or x- and y- coordinates in Cartesian space), but *distance* as a scalar. A point with coordinates (x, y) is the same distance from the origin as the point (y, x) but their positions differ.

The significant issue for software measurement is that a vector *cannot* be turned into a scalar value by some *arbitrary* mathematical function. Simply multiplying position coordinates together would be meaningless; we would obtain a value but we would not know what attribute was being measured. Any relationship between vector attributes and scalar attributes must be based on a *model*; for example position is converted to distance using a model derived from Euclidean geometry (see Section III). In addition, the two elements of a position vector are *independent*, even though they may yield the same scalar value for distance. This implies attribute relationship models are not restricted to models that postulate an association among attributes.

We observe these sorts of models in the software domain. For example, Henry and Kafura's Information Flow Measure [10] involves multiplying fan-in measures by fan-out measures, and the same value is obtained when fan-in and fan-out values are transposed. Shepperd [17] criticised this model because it confuses calling structure with information flow. He proposed a pure information flow measure to avoid this problem. However, we can question the validity of Henry and Kafura's model on other grounds. Kitchenham et al. [11] pointed out that structural fan-out refers to the modules controlled by a given module (i.e., control coupling), whereas structural fan-in relates to the extent to which a module is reused (i.e., internal reuse). Thus, a calling structure value based on multiplying fan-in by fan-out could yield a large value in one of several ways. For example, a large value could be caused by one module controlling many other modules, by a module that is reused extensively, or by a module that controls a moderate number of modules and is reused a moderate amount. To retain this information and enable the measure to show cause as well as effect, a calling structure measure should be represented as a vector rather than a scalar.

I. Implications for Measurement Validation

Our structural measurement model has several implications for validation:

1) Different objects may share the same attribute. This situation implies that any measurement framework derived by classifying attributes as belonging to only one type of software entity is invalid. In practice, measurement frameworks that group attributes against particular entity types provide a useful means of presenting an overview of software measurement, but they should not imply one-to-one relationships.
2) Since an attribute may be measured in many different ways, attributes are independent of the unit used to measure them. Thus, any definition of an attribute that implies a particular measurement scale is invalid. Furthermore, any property of an attribute that is asserted to be a *general* property but implies a specific measurement scale

must also be invalid. Thus, while Zuse criticises Weyuker's complexity measure properties as contradictory because one (property 5) implies a ratio scale and another (property 6) explicitly excludes a ratio scale, we would regard both the properties as inadmissible *because* they imply a scale type. We also regard Weyuker's property 9 as inadmissible because it involves the relation "<" and thereby excludes nominal scale units.

3) A unit may apply to different attributes and different entities. However, a *specific* measurement unit need not be defined independently of the attribute and entity to which it applies. For example, we can refer to the temperature of a liquid on an informal ordinal scale: icy, cold, tepid, warm, hot, scalding. Furthermore, parents bathing infants use a nominal scale to measure bath water: body temperature, or not body temperature. These units of temperature refer to our perception of liquids in general and water in particular. Clearly, the specific units are derived with reference to how we want to use our measures. Moreover, the scale type reflects the use; we do not always need a ratio measure, since nominal measures can convey useful information.
4) Since a direct measure maps attributes to values, we should consider the characteristics of the domain (i.e., the set of possible attribute instances) and the range (i.e., the possible values to be assigned). We should be able to state whether the domain and range are finite or infinite, countable or uncountable, bounded or unbounded. We should also be able to determine if the mapping is discrete, discontinuous or continuous.
5) For an indirect measure, we need additional information. We must have not only an entity and attributes but also a model that describes how the indirect measure relates to the attribute it captures and defines how the indirect measure is to be calculated.
6) Often, a measure maps attributes to a numerical domain: a subset of the integers, rational numbers, or real numbers. If the Representation Condition is satisfied (as it should be for all valid measures) then the behaviour of the attribute in the domain (in the real world) should be reflected in the behaviour of the measure in the range (the numerical world). However, there is a danger that the reverse of the Representation Condition is used to make inappropriate judgments about real world behaviour. For example, if subsystems are categorised as type 1, 2, or 3 where the scale is intended to be nominal, then it is tempting but incorrect to apply the ranking of the integers to the subsystems. That is, it makes no sense to say that a subsystem of type 3 is more important or complex than that of type 1 if the scale is nominal (because there is no ordering on a nominal scale).
7) For compound attributes, we need to understand whether an attribute is best represented as a scalar or a vector. An attribute represented by a vector will not be the same as one constructed from the elements of the vector. Thus, converting a vector of measures to a scalar must be justified by a model that links the different attributes.

Points 3, 4, 5, 6, and 7 above make it clear that when we are constructing measurement units we often need to understand or *model* the way in which a specific type of entity exhibits a specific attribute. Thus, we next consider the place of models in measurement.

III. Models and Measurement

This section considers how we define the elements of measurement we introduced in Section II. In most case our definitions are based on models. We believe that measurement validation must be concerned with the validation of these definition models.

We also identify the need for an additional measurement element that is necessary when we consider a set of measures taken from a set of entities. This we refer to as an entity population.

A. Unit Definition Models

We use definition models to define our measurement units. As noted previously, classical measurement theory restricts the discussion of units to ratio and interval scale measures, but we suggest that nominal and ordinal measures need proper definition of the categories and scales points which is equivalent to defining a unit.

Although unit definition models may be influenced by a desire to measure a specific attribute of a specific entity type, they usually include some concept of the attribute that is *commonly* measured by the scale and may refer to a particular entity. For example, consider the definition of a "metre". Initially a metre was defined to be the distance between two marks on a platinum bar kept in Sèvres. Later it was redefined to be the distance traveled by light in $3.335440952 \times 10^{-9}$ seconds measured by a caesium clock. Both definitions refer to particular entity (a platinum block in the first case and light in the second) and a specific attribute (i.e., distance). However, in the first case the metre is defined by reference to a standard (i.e., a direct representation of the unit); in the second case the metre is defined by reference to a theory of the speed of light. The reason for moving to the second definition was that it leads to more accurate measures. Thus, accuracy is a criterion for selecting one unit definition rather than another. In addition, we must expect our measures to be subject to measurement error determined by the accuracy of our units.

Another feature of many ratio and interval scale units (e.g., of length, mass, time) is that it is possible to convert from one unit to another (e.g., metres to yards) using a conversion constant. We may therefore define a unit by reference to another unit (e.g., a yard can be defined as 0.914 metre).

Finally, many units are constructed from other units. For example, speed can be measured in metres per minute, or miles per hour. Attributes expressed in compound units are usually measured indirectly (see Fig. 3).

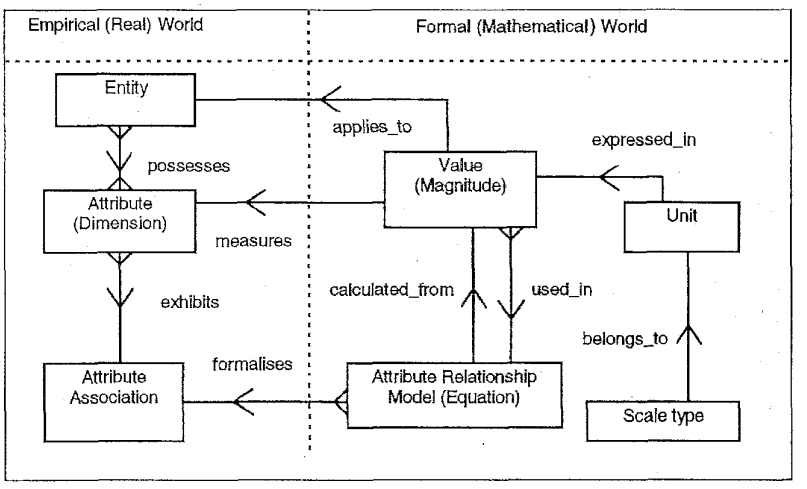

Fig. 2. A structural model for indirect measures of simple attributes.

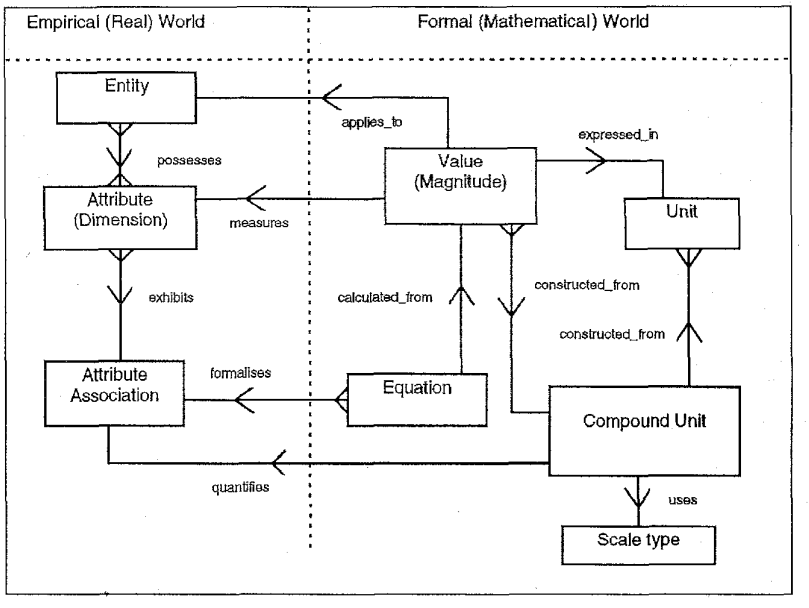

Fig. 3. A structural model for measuring multi-dimensional attributes.

These examples demonstrate that there are at least four types of unit definition model:

1) A definition by reference to a standard or example. For example, a "line of code" is often defined as a non-blank, non-comment line in a code listing.
2) A definition by reference to a wider theory involving the way in which an attribute is observed on a particular entity. For example, an "executable statement" can be defined by reference to the manner in which a compiler handles particular elements of a specific programming language.
3) A definition by reference to conversion from another unit. For example, a "yard" can be defined as three feet, or "system size" as the sum of its module sizes. This type of definition is controlled by the scale type of the units. The scale type determines the appropriate mappings from one unit to another. (See [5] for a discussion of this issue.)
4) A definition by reference to a model involving several attributes. For example, the unit "hours per line of code"

can be used to measure productivity because we define productivity to be the effort to produce a given amount of software. This is discussed further in Section III.C below. (This category might be regarded as a special case of a type 2 definition model.)

B. Instrumentation Models

We use instrumentation models when we take measurements. Closely related to unit definition models, the instrument models tell us how to capture the data we seek. For example, we can use a tape measure to measure a person's height, or a thermometer to measure the temperature of a liquid or a person's body temperature. A tape measure is simply an entity possessing the attribute in which the unit is defined (i.e., distance from one end to the other) with the unit marked repeatedly on the tape. A thermometer, however, is much more sophisticated. It is based on a model of how the temperature of the surrounding environment affects the length of a column of mercury.

Software structure offers a similar example, as structure metrics can be extracted by constructing a control flow graph equivalent to the program. Structure metrics can then be defined by reference to properties of the decomposition tree constructed from the flow graph [7].

These examples suggest that there are two types of measurement instrument model: a direct representational model and an indirect, theory-based instrument. The latter is often based on a model of how an attribute manifests itself in a particular entity. A measure derived from a theory-based instrument is valid only if the underlying theory is valid.

An assumption that is inherent in both types of instrument model is that any two units contributing to a particular value are equivalent. For example, when we measure someone's height, we are making an implicit assumption that a metre of a person's leg is equivalent to a metre of torso. Similarly, when we measure code length in terms of lines of code, we assume that for the purposes of establishing code length all lines of code are equivalent. The equivalence of units is what allows us to compare entities; we can say that Jane, at 1.8 metres in height, is taller than John, at 1.5 metres in height, even though a metre of John is different from a metre of Jane.

C. Attribute Relationship Models

An attribute relationship model is used when we consider an attribute that is derived from a functional relationship involving one or more other attributes. An example of this is productivity, often defined as the effort needed to produce a given amount of software. The unit of productivity depends on the units used to measure effort and software size, e.g., hours per line of code, or hours per transaction. Attribute models are of two types: definition models and predictive models.

C.1. Definition Models

Definition models define a multi-dimensional attribute by expressing a relationship among attributes based directly on our understanding of the desired attribute (see Fig. 3). In this type of model, there is no requirement for a constant term to convert from one unit or dimension to another. Thus, an attribute definition model would be invalid if the unit and dimension of the constructed attribute could not be derived from the input attribute(s) unit and dimension.

C.2. Predictive Models

In predictive models, the values of two or more attributes are believed to be associated; that is, the value of one or more attributes can be used to predict the value of another (see Fig. 2). For example, a cost model often uses size to predict effort. A feature of this type of model is that the unit of the output variable is different from the unit of the input variable(s). Thus, the model must include constants with appropriate units to convert between the different units. For example, the Basic COCOMO model [2] uses size in terms of thousand lines of code (KLoC) as an input variable and effort in terms of staff months (SM) as an output variable, in a formula of the type:

$$effort = a \times size^{b}$$

The multiplicative constant a must have the unit "staff months per KLoC" for the equation to be dimensionally consistent.

D. Practical Problems With Attribute Models

D.1. Attribute Definition Models

The major problem with attribute definition models concerns whether or not a compound attribute is a vector or a scalar. If the attribute is a scalar attribute the next problem is to decide how it should be composed from its component attributes.

The relationship between position and distance offers some insight into the construction of scalar measures from vectors. If we consider a point (x, y) in a Cartesian plane, Euclidean geometry tell us that the distance d of the point from the origin can be calculated from the formula:

$$d = \sqrt{x^2 + y^2}$$

This formula illustrates several constraints that apply to attribute relationships:

1) The conditions when the equation holds are defined by the theory from which it is derived (e.g., the equation holds in a Cartesian plane but not on the surface of a sphere).
2) The equation can be validated (i.e., proven) only by reference to Euclidean geometry.
3) Dimensional analysis tell us that the equation is dimensionally consistent. Distance is measured in a unit of length and the (x, y) coordinates are also measured in units of length. The dimension derived from the equation is:

$$\sqrt{(length \times length) + (length \times length)} = length$$

Thus, the dimension implied by the equation is the dimension we associate with distance.
4) The equation can be corroborated empirically by comparing measured distance with calculated distance.

5) The equation does not hold if we measure the x and y in different units (e.g., the equation will not hold if x is measured in centimetres and y in inches).

D.2. Example: Definition Problems With Function Points

There has always been some dispute about what function points are. Some software engineers assumed a function point was a unit for measuring functionality; others thought it was a unit for measuring product size as perceived by a product user. Function points are calculated from the weighted sum of a number of different elements (the number of elements depending on the type of function point being used: Albrecht or Mark II). Thus, analogous to distance and position, function points appear to be an attribute in their own right derived from an attribute relationship model. If so, the term "function point" does not seem appropriate; function points might be better renamed as functionality, or user requirement size.

However, more serious implications must be addressed. Albrecht function points [1] are the sum of five elements derived from the number of inputs, outputs, queries, logical master files and interfaces to other systems. The input element is based on the number of data elements involved in each system input. If we simply sum the number of data elements involved in all inputs, the sum would be an acceptable measure of input data size. However, Albrecht's model involves classifying each input using an ordinal scale (simple, average, complex) according to the number of data elements and logical files involved, mapping those values to numbers and summing the numbers. Thus, Albrecht's model violates basic scale type constraints (i.e., you cannot sum ordinal scale measures). In addition, the Albrecht function point counting rules mean that the smallest non-null system (e.g., a single inquiry comprising a single input, and a single output) has the value 3, so function point values are discontinuous (i.e., the possible values for a system are 0, 3, 4, 5, etc.) and there is no "unit" value. Thus, Albrecht function points violate our measurement framework in a variety of ways.

Mark II function points [19] take a different approach and merit closer inspection. The elements used in Mark II function points are inputs, entities and outputs for each business transaction. The transaction input size is the sum of the data elements that are input to the system; the transaction output size is the sum of the data elements output from the system; the transaction data processing size is the sum of the number of entities referenced when the transaction is processed. These values are summed for each transaction. Thus, we can identify three different size attributes in the Mark II function point model: input data size, entity usage, and output data size.

The Mark II model requires the attribute values to be weighted and summed. The weights are different for each attribute and are intended to represent the relative amount of development effort devoted to handling inputs and internal entity references, and producing reports. According to our framework, this is permissible if:

1) We have a theoretical model that justifies adding these elements together.
2) The weights themselves have units that convert the three size attributes to a common unit.

In our view both these conditions are violated if we regard "function points" as a size or functionality measure. They can only be satisfied if "function points" represent an effort model.

If we assume that Mark II function points measure size or functionality, the weights must convert each element to a unit related to size or functionality. That is, we need to have a theoretical model of the relationship between data elements and entity accesses. However, no such model exists. In addition, since output and input size are measured in the same unit (i.e., data elements), they should have the same weight, but the Mark II standard model gives the input and output counts different weights. Thus, it seems that Mark II function points cannot measure size or functionality.

Alternatively, if weights are related to the proportion of staff effort needed to develop one input, one output, and one entity, then Mark II function points are not a size measure but an *effort measure* of the type:

$$\begin{aligned} effort = {} & input\_size \times W_I' \\ & + output\_size \times W_O' \\ & + entity\_references \times W_E' \end{aligned}$$

where $W_I' + W_O' + W_E' = 1$ and each weight is a conversion factor from the relevant size measure to effort. This corresponds to Symons [19] description of Mark II function points since he based his weights on empirical observations that developers required 1.56 staff hours per input, 5.9 staff hours per entity reference, and 1.36 staff hours per output element. The implication is that the weights convert size measures to a common effort unit, so the official Mark II function point model measures effort. However, this interpretation of Mark II function points implies that we cannot talk about productivity in a unit "function points per hour". If function points are effort measures, dimensional analysis implies that dividing function points by effort leads to a constant.

Thus, there are major problems associated with the construction and meaning of function point measures. This situation should be of considerable concern to software measurement researchers and practitioners. An ISO technical committee is attempting to produce an international standard for function points. Unless we understand what a function point is (i.e., dimension or unit, measure, or model) and can confirm that its use leads to valid measures, we risk permitting invalid measures to become international standards.

Based on the validity concepts presented earlier, we can avoid some of the problems with function points if we treat them as follows:

1) Function points might be viewed as a means of measuring the "shape" of a software product in terms of a vector of significant elements. Thus we might interpret a "shape" vector for a software product to be (input data elements, entity references, output data elements) analogous to the way we can describe a person's shape as (shoulder width, waist size, inside leg length).
2) If we were confident that the elements of such a shape measure influence product development effort, we could

attempt to derive an empirical effort estimation model. For example, we could use data collected on past projects to generate effort prediction models of the form:

$$effort = b_0 + b_1 \times input\_size + b_2 \times entity\_references + b_3 \times output\_size.$$

Clearly, such a view of function points does not lead to a scalar productivity value of the type "function points per hour," but it does avoid many other technical problems with the measurement and construction of function points. However, predictive models pose their own problems as discussed below.

D.3. Predictive Models

The problems with predictive models arise because they are usually empirical, in the sense that they have no solid theoretical basis for assigning values to model constants. For example, a cost model such as COCOMO rests on the "theory" that the more software people have to produce, the more working time they will need to do the job. But this statement gives us no theoretical basis for assigning values to the model constants a or b. However, if our theory states that effort is proportional to size, we might assume that $b = 1$. To compensate for lack of theory, researchers often accumulate data reflecting the relevant variables. Then, using an assortment of analysis techniques, they calculate values for the constants that best fit the data. Thus, models such as COCOMO are "calibrated" to datasets that represent likely future situations.

When we use empirically-based models, we often find that they are not very accurate. For example, when effort and size data are analysed, the amount of variability in effort that is accounted for by size is usually in the order of 40-70% [12]. This inaccuracy does not mean that such models are invalid or that input measures have a large measurement error. Rather, it means that the models are incomplete; that is, there are many factors affecting effort that have not been included in the model. Thus, model incompleteness as well as measurement error contribute to model error. In statistical texts, it is customary to include an error term explicitly in such models to draw attention to their inherent inaccuracy. Establishing the validity of predictive models is discussed in Section IV.

E. Measurement Protocols

Measurement protocols let us measure a specific attribute on a specific entity consistently and repeatably. These characteristics are essential because measures should be, as far as possible, independent of the measurer and the environment. Measurement protocols are also necessary because we often base our understanding of entities on entity abstractions, but measure "real" entities. For example, a protocol for measuring the height of adult humans in metres might be: the person must be standing (not bending over); measurement must start at the top of the head (not from the tip of up-stretched arms); the person must remove his/her shoes; the person must stand on the soles of the feet (not on tiptoe).

Measurement protocols are concerned mainly with how an attribute can be measured consistently on a specific entity. They appear to have little to do with the measurement unit. However, we cannot assume that protocols are independent of the measurement scale. If we were interested only in classifying adults as tall or not tall, we might choose a protocol based on whether an individual can reach an item on a shelf of a given height. In this case, we might allow individual to stretch arms and stand on tiptoe. Thus, the measurement unit influences the protocol, albeit rather indirectly.

A measurement protocol is not simply derived from a definition of the measurement unit or the attribute. It is associated with obtaining a measurement value and the use to which the measure will be put, and therefore can be defined only when you have decided to measure a specific attribute on a specific entity using a specific measurement unit for a specific purpose. Thus, empirical studies must specify the measurement protocols they used; otherwise they cannot themselves be replicated (and replication is the basis of scientific validation). A valid measurement protocol must be unambiguous and must prevent invalid measurement procedures such as double counting.

F. Differences Among the Definition Models

Unit definitions, measurement instruments and measurement protocols are separate concerns, but all may be based on detailed models of the way in which an attribute is observed on specific entities. However, the distinctions are necessary for two reasons:

1) They are separate parts of the measurement process. Defining a line of code is not the same thing as having a computer program that extracts a count of the number of lines of codes in a module. Similarly, having a definition of a line of code, and a means of counting lines of code, does not define the conditions under which the count will be made (for example, whether the measurement should be taken after the first successful compilation or after final system acceptance and whether or not lines of code written in different languages are counted separately).
2) Since they are different but necessary parts of a measurement activity, they represent different points where our measurement process might be in error and so make our measures invalid. We need to be sure that we have identified all the factors that could invalidate our measures. In addition, it is likely that different activities will imply different methods of validation. For example, we might use the Representation condition from measurement theory when we are validating the use of a line of code as a unit of code length, but we might require a formal proof to convince ourselves that a program used to extract a count of lines of code is a valid implementation of the definition of a line of code.

Nonetheless, we cannot ignore the fact that it may be difficult sometimes to know whether a definition refers to a unit, an instrument model, a measurement protocol or even the attribute itself.

G. Entity Population Models

In Section II we introduced structure models to describe the process of obtaining a measure. However, in practice we usually measure an attribute on a set of entities and need to analyse the resulting dataset. The structure models presented in Section II do not address data analysis or data interpretation. For this we need to consider entity population models. When we measure an attribute on a particular entity, we assume our measurement can be interpreted by reference to the *normal values* of specific attributes for specific entity classes under specific conditions. An entity population model allows us to define the normal values for an attribute. For example, when we measure the body temperature of human adults, we expect that the value we obtain will be close to 98.6 degrees Fahrenheit. Differences from that value may indicate that the individual was ill (when body temperature increases), or that the individual was asleep (when body temperature decreases). If we have no idea what values we expect when we measure an attribute on an entity we cannot interpret what the value means.

If we do not have a population model of the entity we are measuring, we can use statistics to create such a model. For example, we might say that the average length of a set of modules is 200 lines with a standard deviation of 50 lines. If we understand our population model, we can use statistics that represent that population to decide whether particular modules are normal or abnormal with respect to the population norms. That is, the model tells us whether differences in measurement are due to likely variation in the population or are unexpected and unusual. The population modeling activity raises two measurement issues:

1) When we derive statistics from measures on a set of entities, the statistics of *central tendency* and *dispersion* are constrained by the scale type we have used (e.g., averages and standard deviations can be used for ratio and interval scale measures, but medians and percentiles should be used for ordinal scale measures);
2) The distribution of the set of values also affects our choice of statistics. We might prefer to use medians and percentiles for interval and ratio scale measures in datasets that are severely skewed.

However, it is much more important that we have a notion of the population from which we obtained the measured entities. Statistics such as the average and standard deviation are attributes of a *set of entities*, not a single entity. To use these values, we need to know whether the statistics we obtain from our dataset are representative of a larger population of entities. Since it is difficult to confirm that a set of entities is representative, statisticians require that the entities we measure be a random sample of the larger population.

If we do not define a population model, we cannot interpret the values obtained from a new entity. For example, we may be interested in measuring the height of human beings for purposes such as furniture design. However, if we measured any humans, including adult males, adult females, adolescents, and infants, the "average" of such a dataset could yield undesirable results. We must define carefully the set of entities allowed to be in the population we want to model.

A population model can be defined simply, as "modules from a particular product," or "projects of a particular application type". Population models are usually derived from the following considerations:

- conditions or states that affect the entity (e.g., language for programs, education for programmers, tool support for tasks);
- the goals of the measurement program (e.g., investigating the effect of process changes on productivity or quality, or identifying entities with abnormal values).

Population models and their relationship to measurement are essential features of software experimentation. For a more complete discussion of experimentation, see Pfleeger [14].

IV. Software Measurement Validation

We have seen in Section II how the elements of the measurement process and their properties underpin the notion of measurement validity. Thus, a value that does not have an associated unit is a clear sign that a measure is meaningless. Section III considered how we define those elements when we construct or use a measure and found that most definitions rely on an underlying model. This section considers how we should validate our measures. Our basic assumption is that in order for a measure to be valid both of the following conditions must hold:

1) the measure must not violate any necessary properties of its elements;
2) each model used in the measurement process must be valid.

In this section we discuss the difference between theoretical validation and empirical validation and propose several specific validation methods. In this context, we need to define what we mean by a valid measure. Formally, we cannot prove a theory but can only falsify it or corroborate it [15], so a valid measure is one that we are unable to invalidate *yet*.

We saw in Sections II and III that to decide whether a measurement is valid we need to confirm:

- *attribute validity*, i.e., whether the attribute in which we are interested is actually exhibited by the entity we are measuring. Attribute validity must be considered both for directly measurable attributes and for indirectly measurable attributes that are derived from other attributes;
- *unit validity*, i.e., whether the measurement unit being used is an appropriate means of measuring the attribute;
- *instrument validity*, i.e., whether any model underlying a measuring instrument is valid and the measuring instrument is properly calibrated;
- *protocol validity*, i.e., whether an acceptable measurement protocol is adopted.

It is also clear from Section III that we use attribute prediction models and entity population models to support our use of measurements. We need to consider validation of these models, if we are interested in confirming that we are *using* measure-

ments appropriately. In this section we discuss some of the issues involved in validation but we do not claim this is a compete validation framework.

A. Theoretical and Empirical Validation

We have two basic methods of testing the validity of our measures:

- *theoretical validation*, which confirms that the measurement does not violate any necessary properties of the elements of measurement or the definition models (e.g., a compound unit must be constructed by transformations permitted by the scale type of its components);
- *empirical validation*, which corroborates that measured values of attributes are consistent with values predicted by models involving the attribute (e.g., the measured distance from an origin to a point is the square root of the sum of the x and y co-ordinate values).

Theoretical methods of validation allow us to say that a measure is valid with respect to certain defined criteria; empirical methods provide corroborating evidence of validity or invalidity.

B. Theoretical Measurement Validation Issues

B.1. Properties of Measures

In Section II, we identified a number of theoretical criteria that need to be satisfied by a valid measure. We will summarise them and compare them with the properties of complexity measures suggested by Melton et al. [13] and Weyuker [20].

1) *For an attribute to be measurable, it must allow different entities to be distinguished from one another.* This statement is related to Melton et al.'s first assumption that says "There is an order on abstractions of documents; the order is based on relative complexity," where a document might be specification, design or code listing. It is also related to Weyuker's first property, which states that there must exist two entities for which the measure results in different values. We prefer Weyuker's formulation. Melton et al.'s definition is inadmissible because the concept of "order" excludes a nominal scale unit.
2) *A valid measure must obey the Representation Condition*, i.e., it must preserve our intuitive notions about the attribute and the way in which it distinguishes different entities. This statement is the same as Melton et al.'s second assumption. Weyuker's eighth property is related to this issue. It states that if we have two programs that are the same program except that we have given them different labels, their complexity measures must be the same. Assuming complexity relates to structural complexity not psychological complexity, this assertion is implied by the Representation Condition. Our intuitive notion of structural complexity (or length, or modularity) does not allow us to separate or order copies of the same entity with different labels. Thus, if we accept the Representative Condition, we do not require Weyuker's eighth property.
3) *Each unit of an attribute contributing to a valid measure is equivalent.* This seems to be standard measurement practice. Weyuker's property 7 relates to this issue. She, in fact, asserts the *converse* of this assumption by claiming that program complexity should be responsive to the order of statements in a program. It seems here that Weyuker is confusing the attributes program correctness and/or psychological complexity with structural complexity. It is unlikely that a random re-ordering of program will be correct or understandable, but a re-ordering would not necessarily be more structurally complex.

Of Weyuker's nine properties, we agree with property 1, we reject properties 5, 6, and 9 because they imply a scale type, we suggest property 7 is inappropriate because it contradicts standard measurement practice, and discard property 8 as unnecessary if we accept the Representation condition. Weyuker identifies three other properties (2, 3, and 4) that we have not yet considered. These properties are discussed below.

Weyuker's second property says that there should be only a finite number of programs with a given complexity value. This finiteness requirement does not seem to be needed for measures of physical attributes that map to the Rational or the Real numbers (for example, there is an infinite number of points at a distance

$$d = \sqrt{x^2 + y^2}$$

from the origin of a Cartesian plan). It is difficult to see why program complexity values require this property.

Weyuker's third property states that a valid complexity measure allows different programs to have the same complexity value. Although for continuous measures we can refer to measures as equal only within the constraints of measurement error, the third property is a reasonable requirement. It is also consistent with measurements of other attributes in other domains (e.g., temperature of liquid, weights of animals etc.), so we regard it as a general property of a measure, not a property specific to complexity measures. Melton et al., however, seem inclined to dispute this assumption. They note that a Relation such as $\leq$ applied to the length of a set of documents (D) measured in terms of lines of code is reflexive and transitive, and therefore (D, $\leq$) is a preordered set. However, different programs can have the same length, so (D, $\leq$) is not a partially ordered set for program length. Later they state "On the intuitive level, there may be problems with preorders that are not partial orders". However, since most valid measures in other domains are not based on partial orders, we are unable to identify the problem to which Melton et al. are referring.

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.

From Weyuker's properties we believe we need to add only one more property to the three we have defined above.

4) *Different entities can have the same attribute value (within the limits of measurement error).*

The above four properties seem reasonable and necessary properties for all measures to possess, but they are not suffi-

cient to cope with indirect measures.

B.2. Properties of Indirect Measures

Sections II and III suggested that measures calculated from a model must exhibit a number of properties:

- be based on a model concerning the relationship among attributes defined on specific abstract entities;
- be based on a dimensionally consistent model;
- exhibit no unexpected discontinuities;
- use units and scale types correctly.

These properties imply that an indirect measure is invalid if any of the following conditions hold:

- there is no underlying model to justify its construction (e.g., it is constructed from various attributes but the attribute it is supposed to measure cannot be defined, or cannot be measured directly);
- there is an underlying model but the measure can be shown to be invalid in the circumstances when it is being applied (e.g., the distance from the origin *on a sphere* cannot be calculated from the square root of square of the x and y co-ordinates);
- the measurement fails a dimensional analysis (e.g., it measures effort when it is supposed to measure size);
- the measure is discontinuous within its defined numerical bounds (e.g., it is infinite or undefined for certain values of its component attributes);
- the measure uses scale types incorrectly (e.g., it adds nominal scale attributes);
- the measure uses units inconsistently (e.g., it mixes effort in hours with effort in days).

B.3. Alternative Units

If we have a measurement unit that is accepted as a valid means of measuring an attribute, an alternative unit will be valid if the new unit is an admissible transformation from the original unit. The scale type of the unit determines which transformations are admissible. For example, if we accept "lines of code" as a valid measure of program length, "thousand lines of code" is also valid.

B.4. Validating Instrument Models

When we validate a measurement instrument, we need to confirm that the measurement instrument accurately measures attribute values in a given unit. If our measurement instrument is based on a model of the attribute on a different entity, we need to validate that the attribute behaves in a directly comparable way on the new entity.

It is possible to define a measurement instrument formally. For example, Fenton [5] considers the concept "depth of nesting" and demonstrates how to derive a formal measure of depth of nesting, α, in terms of its effect on prime flowgraphs, nesting and sequencing as follows:

- *Primes*: The depth of nesting of the prime P_0 (a graph comprising a single node) and of the prime P_1 (a graph comprising two nodes and a single edge) is zero; the depth of nesting of any other prime F is equal to 1. Thus:

$$\alpha(P_1) = 0 \text{ and } \alpha(P_0) = 0 \text{ and}$$

$$\text{if } F \neq P_1 \text{ and } F \neq P_0 \text{ then } \alpha(F) = 1.$$

- *Sequence*: The depth of nesting of the sequence $F_1, \ldots, F_n$ is the maximum of the depth of nesting of the F_is. Thus:

$$\alpha(F_1, \ldots, F_n) = max\,(\alpha(F_1), \ldots, \alpha(F_n)).$$

- *Nesting*: The depth of nesting of the flowgraph $F(F_1, \ldots, F_n)$, where F is the flowgraph made up of a prime F with the sequence $F_1, \ldots, F_n$ nested in it, is the maximum of the depth of nesting of the F_is plus one because of the extra nesting level of F. Thus:

$$\alpha(F(F_1, \ldots, F_n)) = 1 + max\,(\alpha(F_1), \ldots, \alpha(F_n))$$

where $F \neq P_1$ and $F \neq P_0$.

In terms of our structure model, depth of nesting is the attribute of interest, the unit used to measure depth of nesting is "non-trivial primes" and the axioms represent the measurement protocol. Primes themselves are defined in the context of the theory of flowgraphs. The measurement instrument can be invalidated if it can be shown that the axioms are incomplete or inconsistent.

B.5. Validating Measurement Protocols

Measurement protocols must be unambiguous, self-consistent and prevent problems such as double counting. A protocol that does not violate these criteria is usually validated by peer acceptance rather than logic or empirical studies.

C. Corroborating Measures Empirically

C.1. Direct Measures

To corroborate a measure, we can perform experiments to see whether people agree that an attributes exists or whether a mapping to a value captures their understanding of the attribute. For example, if we want to know whether an entity exhibits a particular attribute we can ask a random selection of individuals to classify a set of entities according to a set of possible categories (for nominal scale measures) or to rank the set of entities with respect to the attribute of interest (for ordinal, interval or ratio scale measures). We can then use measures of association to assess how consistently a group of people categorise or order a set of entities.

Siegel and Castellan [18] identify a number of different measures of association that can be used:

1) The Kendall Coefficient of Concordance W that measures the extent of agreement among k raters ($k \geq 2$). This can be used to assess the extent of agreement of k human judges ordering a set of entities with respect to an attribute.
2) The Kendall Correlation between several judges and a criterion ranking T_C. This can be used to see how well a set of rankings produced by humans judges compares with the rank order produced by a measure.
3) The Kappa statistic K which can be used to assess the agreement among k human judges classifying a set of entities against attribute categories.
4) A generalisation of K to assess agreement with a criterion

(similar to T_C) [8].

Measures of association for nominal attributes vary from 0 (no association) to 1 (perfect association). Measures of association for ordinal attributes vary from -1 (perfect inverse association) to 1 (perfect association) with 0 indicating no association.

When we consider consistency among subjects, the measures of association need not be perfect but should not be significantly different from 1. If the correlation is not different from 0, we can conclude that the attribute is *not* exhibited by the entity. If the correlation is not significantly different from 1, we conclude that the attribute is exhibited by the entity. If the correlation is significantly greater than 0 but significantly less than 1, we can assume only that the attribute is not well understood.

If the attribute is valid, we can investigate the validity of the measurement unit. In this case we use the rank ordering (or classification) given by the proposed measurement unit with a particular measurement protocol as a criterion ranking. If the association measure between subjects and criterion ranking is not significantly different from 1, we can conclude that the measurement unit and measurement protocol lead to valid measures. If the association measure is significantly different from 1, we do not know whether our unit or our protocol (or both) were invalid.

We acknowledge that this form of validation is not very stringent. This form of validation can confirm only ordinal relationships even if our measure were really interval or ratio scale. Also it would identify some unacceptable measurements as "valid". For example, suppose you measure height using a protocol that measures a standing person from top of head to hip and then adds distance from the person's waist to the floor. If you measured a set of adult males, you would find a very strong correlation between the order obtained by ordering them on visual assessment of height and the measured value of height. However, the measurement protocol is not valid because it permits double counting, and the mapping would breakdown if you used the protocol on mixed set of adult males and adult females. Thus, this form of corroboration, although necessary, is not sufficient to confirm that a measure is valid.

C.2. *Indirect Measures*

Indirect measures are based on attribute relationship models. If we have derived a measure from a proven underlying model, we can justify our use of the measure by reference to that model. If our model is not proven, we can use standard statistical techniques such as regression to corroborate that an indirect measure is *identical* to a direct measure.

C.3. *Example of Empirical Validation Procedures*

Halstead [9] proposed the following indirect measure of the total number of operators and operands (N):

$$\tilde{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

where n_1 is the number of distinct operators and n_2 is the number of distinct operands.

Clearly, this model can be corroborated by obtaining a random sample of programs, and measuring n_1 and n_2 together with N_1 (the total number of operators) and N_2 (the total number of operands). By definition $N = N_1 + N_2$, and thus for each program we can compare a direct measure N with the indirect measure $\tilde{N}$.

Halstead suggested that this relationship would apply exactly only for pure programs that have no redundant operators and operands, so we should not expect $N = \tilde{N}$ for every program. However, we could use linear regression to investigate the relationship between the two values by fitting a linear model of the type:

$$N = a + b\tilde{N}$$

If Halstead's model were valid, we would expect our equation to have an intercept (a) that is not significantly different from 0 and slope (b) that is not significantly different from 1.

It is not sufficient to confirm that the *correlation* between N and $\tilde{N}$ is not significantly different from 1, because correlation does not imply an identity relationship. A correlation not significantly different from 1 could occur when the slope was different from 1 and the intercept was non-zero.

As with direct measures, when indirect measures are not derived from proven models, empirical corroboration is necessary but not sufficient to validate an indirect measure.

C.4. *Validating Alternative Units*

Some researchers offer empirical evidence that measures made using a new unit correlate strongly with measures made using the original one. However, such evidence *is not* appropriate for validating alternative units. Measurement theory suggests that if the new unit preserves the representation condition it is a valid measure of the original attribute. However, the correlation may be due to an association between the attributes rather than equivalence of the units.

For example, we can confirm a relationship between units by showing that distance measured in metres is 3.281 times the same distance measured in feet. However, even if we found a very strong correlation lines of code and person hours for some projects, we would not regard lines of code as a unit of effort. The problem is that an empirical study of the correlation between two different values would not be able to distinguish the difference between a conversion between units (such as metres to feet), and a relationship between associated attributes (such as size and effort).

D. Validating Empirical Relationships Between Attributes

To determine if one measure can be used to predict the value of another measure, we need to establish whether or not there is a relationship between the two attributes being measured. Theoretical relationships resulting from entity and attribute models have already been discussed, but in software (and many other domains) practitioners and researchers are also interested in empirical relationships based on observation. For example, in medicine researchers have observed a relationship between smoking and lung cancer. In software, many researchers have identified relationships between structural complexity measures and fault rates.

Schneidewind has suggested that corroboration of relationships between attributes should be the fundamental method validating software metrics [16]. He states "if metrics are to be of

greatest utility, the validation should be performed in terms of the quality function (quality assessment, control and prediction) that the metrics are to support". We disagree with Schneidewind, because validating a predictive or usage model is different from validating a measure. However, we agree that basic statistical techniques such as correlation analysis can be used to investigate relationships between attributes. A full discussion of Schneidewind's approach is outside the scope of this paper. However, his work raises several issues that need to be addressed to ensure that empirical validation exercises are themselves valid:

1) We need to be aware of whether we are attempting to corroborate an association or a causal relationship. For example, there is a significant correlation between the number of storks in Scandinavia and the human population level. However, we do not believe that killing storks will reduce human population levels, because the relationship is associative not causal. Observation of a statistically significant correlation is not sufficient to corroborate a causal relationship. If we want to confirm causality, we must control the attribute we believe causes the relationship (the independent variable, x, in a regression equation) and confirm that the expected change is observed in the other attribute (the dependent variable y in a regression equation). However, in many cases, it is difficult to control software attributes.
2) We need to consider whether our method of investigating a relationship is likely to change the relationship itself. For example, to validate that a cost model produces accurate predictions, we would like to compare predicted effort and duration with actual effort and duration. However, if the cost model was used to set project budgets and schedules, we run the risk of getting results that are unrepresentative of the true accuracy of the model, because the project managers would actively manage their projects in order to achieve those budgets and schedules. This manager bias would force the results to fit the model. A similar problem occurs in medicine, where it is known that doctors can influence the patient's response to medical treatments. As a result, drug trials are performed as "double-blind" experiments where neither patient nor doctor knows who is receiving the placebo and who is receiving the new drug.
3) We need to be aware of the difference between goodness of fit and predictive capability. No matter how well an empirical model fits the historic data used to generate it, we cannot assume that it will be similarly accurate when used for predictive purposes. We need to use "cross-validation" to assess predictive capability. At its simplest, cross-validation uses a subset of data to generate a model; the remaining data are then used to test the accuracy of the model. This procedure assumes that the data on which the model will be used have exactly the same properties as the data that were used to generate and validate the model. A similar technique is used in AI applications that learn by example; there, a dataset of example cases are split into a "learning" dataset and a "test" dataset.

V. Conclusions

Proper validation of measures must be the scientific basis for the discipline of software measurement. We have identified the key issues involved in validating measurements, starting with a measurement structure model and then considering the other models involved in measurement. We have reviewed previous work in the field and identified a number of shortcomings. Our discussion of function points has made it clear that measurement validation is required for pragmatic as well as theoretical reasons. We invite software measurement practitioners and researchers to review and assess critically all work in this field including ours, so that we as a community can agree on the meaning and methods of validation.

We are not suggesting that software measurement should stop while researchers try to decide how to validate measures, because it is clear that many direct measures at least conform to the Representation Condition. Such measures include "lines of code" as a measure of program length, cyclomatic number as a structure measure identifying the number linearly independent paths through a program, effort in person hours as a measure of the human resources needed for a task, and many others. The main problems appear to rest with indirect measures. To avoid major problems, we suggest that practitioners adhere to the following rules:

- Use measures that you understand in the context of your own goals and situation.
- If you are concerned about a multi-dimensional attribute such as complexity or quality, use different measures for different aspects of the attribute.
- Do not try to combine different aspects of a multi-dimensional attribute into a single measurement unless you have a model or theory to support you.
- If you measure different aspects of a multi-dimensional attribute and want to predict some other attribute, use stepwise multivariate linear regression to "combine" the attributes into a single predictive model. Using a stepwise algorithm ensures that only aspects that contribute to the prediction are included in the model.
- Use "function points" in accordance with the suggestions in the example in Section III.D.
- If you use a predictive model that is based on analysing empirical data, not on an underlying theory, treat the model with caution; it is unlikely to represent a truth of nature.

We also caution researchers to avoid justifying measures on the basis of either disputed properties or empirical relationships with other attributes.

We have identified a number of issues that need to be considered when measures are defined and used. However, our framework and suggestions do not constitute a definitive theory of validation. Additional thought must be given to the following issues:

1) If our structure model is correct, it would be useful to be able to prove that it is sufficient as well as necessary. In particular, we want to prove that the relationship between unit and scale type is one-to-one.

2) Various researchers have proposed "basic" properties that valid software measurements should possess. We have suggested four such properties, but we want to confirm that they are necessary and sufficient.
3) We have recommended the use of vector attributes as well as scalar attributes, but we have not provided definitive rules by which the dimensions of an attribute can be confirmed. Indeed, we do not know whether it is possible to identify such rules except within the context of a theoretical model.
4) In Section III, we identified five different "models" involved in measurement but we cannot prove that, with the structure model in Section II, they represent the complete set of models needed for valid measurements. For example, other researchers may dispute the distinctions we have made among models; in particular, some researchers may regard an instrument model as a special case of a unit definition model.

A final problem with indirect measures is one raised by Melton et al. [13]. Measurement models are based on abstract entities. We need to have more abstract definitions of software entities before we can derive theories to justify the relationships among their attributes. Thus, software measurement creators and users should remember that measurement is ultimately concerned with software engineering understanding, control and improvement, not just with the abstract properties of measures.

ACKNOWLEDGMENTS

We would like to thank David Law for reviewing an earlier version of this manuscript and suggesting valuable improvements to our approach.

REFERENCES

[1] A. Albrecht, "Measuring application development productivity," *Proc. IBM Applications Development Joint SHARE/GUIDE Symp.*, Monterey, Calif., pp. 83-92, 1979.
[2] B. Boehm, *Software Engineering Economics*. Englewood Cliffs, N.J.: Prentice-Hall, 1981, p. 57.
[3] S. Chidamber and C. Kemmerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, June 1994.
[4] N. Fenton, "Software measurement: A necessary scientific basis," *IEEE Transactions on Software Engineering*, vol. 20, no. 3, pp. 199-206, Mar. 1994.
[5] N. Fenton, *Software Metrics: A Rigorous Approach*. London: Chapman-Hall, 1991, pp. 28-37.
[6] N. Fenton and B. Kitchenham, "Validating software measures," *J. of Software Technology, Verification and Reliability*, vol. 1, no. 2, pp. 27-42, 1991.
[7] N. Fenton, R. Whitty, and A. Kaposi, "A generalised mathematical theory of structured programming," *Theoretical Computer Science*, vol. 36, pp. 145-171, 1985.
[8] J. Fleiss, "Measuring nominal scale agreement among many raters," *Psychological Bull.*, vol. 76, pp. 378-382, 1971.
[9] M. Halstead, *Elements of Software Science*. New York: Elsevier North Holland Inc., 1977, chapter 2, pp. 9-18.
[10] S. Henry and D. Kafura, "Software structure metrics based on information flow," *IEEE Transactions on Software Engineering*, vol. 7, no. 5, pp. 510-518, May 1981.
[11] B. Kitchenham, L. Pickard, and S. Linkman, "An evaluation of some design metrics," *Software Eng. J.*, vol. 5, no. 1, pp. 50-58, Jan. 1990.
[12] B. Kitchenham, "Using function points for software cost estimation," N. Fenton, R. Whitty, and Y. Iizuka, eds., *Software Quality Assurance and Measurement*. International Thomson Press, 1995, pp. 266-280.
[13] A. Melton, D. Gustafson, J. Bieman, and A. Baker, "A mathematical perspective for software measures research," *J. of Software Eng.*, vol. 5, no. 5, pp. 246-254, 1990.
[14] S. Pfleeger, "Experimental design and analysis in software engineering," *Annals of Software Eng.*, vol. 1, no. 1, pp. 219-253, Jan. 1995.
[15] K. Popper, *The Logic of Scientific Discovery*, 10th impression (revised). London: Hutchinson Co. Publishers Ltd., 1980, chapter 4, pp. 78-92.
[16] N. Schneidewind, "Methodology for validating software metrics," *IEEE Transactions on Software Engineering*, vol. 18, no. 5, pp. 410-422, May 1992.
[17] M. Shepperd, "Design metrics: An empirical analysis," *Software Eng. J.*, vol. 5, no. 1, pp. 3-10, 1990.
[18] S. Siegel and N. Castellan, Jr., *Nonparametric Statistics for the Behavioral Sciences*, 2nd edition. New York: McGraw-Hill Book Company, 1988, chapter 9, pp. 224-312.
[19] C. Symons, "Function point analysis: Difficulties and improvements," *IEEE Transactions on Software Engineering*, vol. 14, no. 1, pp. 3-11, Jan. 1988.
[20] E. Weyuker, "Evaluating software complexity measures," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1,357-1,365, Sept. 1988.
[21] H. Zuse, "Support of experimentation by measurement theory," H. Rombach, V. Basili, and R. Selby, eds., *Experimental Software Engineering Issues* (Lecture notes in computer science, vol. 706). New York: Springer-Verlag, 1993, pp. 137-140.

Barbara Kitchenham received a PhD from the University of Leeds. Currently she is a software engineering consultant at Britain's National Computing Centre. Her interests are software metrics and their application to project management, quality control, and evaluation of software technologies. She was a programmer for ICL's Operating System Division before becoming involved with a number of UK and European research projects on software quality, software cost estimation, and evaluation methodologies for software technology. She has written more than 30 papers on software metrics. She is an associate fellow of the Institute of Mathematics and Its Applications and a fellow of the Royal Statistical Society.

Shari Lawrence Pfleeger received a PhD in information technology from George Mason University. Dr. Pfleeger is president of Systems/Software, a consultancy that specializes in software engineering and technology transfer. Her clients have included major corporations, government agencies, and universities. Dr. Pfleeger has been a principal scientist at both the Contel Technology Center and Mitre. She spent three years as a visiting professorial research fellow at the Centre for Software Reliability, investigating how software engineering techniques affect software quality. She has written three software engineering texts and several dozen articles. A member of IEEE, IEEE Computer Society, and ACM, she is an adviser to *Spectrum* and associate editor-in-chief of *IEEE Software*.

Norman Fenton is a professor of computing science in the Centre for Software Reliability at City University, London. He was previously the director of the Centre for Systems and Software Engineering at South Bank University and a postdoctoral research fellow at Oxford University (Mathematical Institute). He has consulted widely to industry about metrics programs and has also led numerous collaborative projects. One such current project is developing a measurement-based framework for the assessment of software engineering standards and methods. His research interests are in software measurement and formal methods of software development. He has written three books on these subjects and published many papers. Prof. Fenton is editor of Chapman and Hall's Computer Science Research and Practice Series and is on the editorial board of the *Software Quality Journal*. He has chaired several international conferences on software metrics. Prof. Fenton is secretary of the (National) Centre for Software Reliability. He is a Chartered Engineer (Member of the IEE), Associate Fellow of the Institute of Mathematics and Its Applications, and a member of the IEEE Computer Society.