

COMPARISON OF APPROACHES FOR SOFTWARE MEASUREMENT

Ledis Chirinos, Francisca Losavio

Centro de Ingeniería de Software y Sistemas (ISYS), Laboratorio LaTecS
Facultad de Ciencias, Universidad Central de Venezuela.
Apdo. 47567, Los Chaguaramos 1041-A
Caracas - Venezuela
{lchirinos, flosavio}@cantv.net

Abstract

This paper addresses the problem of constructing software measures to obtain repeatable and comparable values. Software measurement is a key component in assuring and evaluating software quality. Hence, theoretically correct and understandable measures are required. A survey on several approaches modeling software measures and software measurement process is presented. These approaches have been selected because they consider that a measure must be fully defined in order to avoid inconsistency. The main goal of this paper is to summarize and compare the approaches discussed: Kitchenham's framework for software measurement validation, SQUID (Software Quality In the Development process) approach, Kitchenham's modeling of software measurement data, and the ISO/IEC 15939: software measurement process framework. The comparison focuses on the definition of the elements involved in software measurement. Several criteria or point of comparison are established.

Keywords: software measurement, software quality, metrics, measures, software engineering

1. Introduction

Software measurement plays a critical role in project management, process understanding, and process and product improvement. With measurement, developers can evaluate current situations and products, predict future characteristics and behavior, and control the development and maintenance process. *Measurement* is defined as the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules [2]. An *entity* is an object (such as a person or a room) or an event (such as a journey or the testing phase of a software project) in the real world [2] and an *attribute* is defined as a measurable physical or abstract property of an entity [6]. The process of assigning numbers or symbols to entities with respect to an attribute is an encoded way of expressing information about the attribute, and these encoded descriptions of the attribute allow the differentiation between entities. It is important to notice that a measure of an attribute always should describes the entity in a manner that corresponds to generally accepted views about the attribute. Hence, it is extremely important that the measurement process should accomplish the task of defining a measure to identify and to define explicitly all the elements involved to obtain the values, in order to ensure consistency, repeatability, comparison and the minimization of measurement errors. Most of the data processing problems (collection, validation, storage, and analysis) arise from poor definitions of software measures, as well as the missing contextual information related to the measurement goals [4]. If software measurement definitions are incomplete and/or poorly documented, it is easy for different data collectors to collect invalid or incomparable measures. For example, it is widely accepted that the conditions and context during which the software measurement takes place have a profound impact on the comparability of data values.

This paper deals with the problem related to the *construction of software measures* in order to obtain repeatable and comparable values. By construction, we mean that all the elements involved in the definition of the measure must be structured and specified to avoid inconsistency. In this sense, the main goal of this work is to summarize, discuss and compare several approaches to model data elements for software measurement that can be used to define measures for software.

The area of software measurement is also known as software metrics. It is important to point out the confusing situation with the usage of the terms *measure* and *metric*. In the literature the terms metric and measure are used as synonyms, meaning that the term metric is not used within the strict mathematical definition, since mathematically this term assumes a space, analyzing a distance. Although both measure and metric are considered as functions, a metric could be viewed as a particular case of a measure. According to ISO 9126 [5], which is a standard to specify a quality model for software products, *metric* is the defined measurement method and the measurement scale. *Measure* is defined in [7] as a rule for assigning a quantitative or categorical value from a defined scale to one or more attributes. A *rule* may be a

measurement method, function, or model. The term “*measures*” is used to refer collectively to *base measures*, *derived measures* and *indicators*. A software engineer collects values of measures (bases and derived measures) in order to obtain indicators. An *indicator* is an estimate or evaluation of specified attributes derived from a model with respect to defined information needs [7]. A *model* is an algorithm or calculation combining one or more base and/or derived measures with associated decision criteria. In this sense, an indicator provides insight into the software process, a software project, or the product itself, which enables the project manager or software engineers to adjust the project or improve the process. Well-defined indicators come from valid measures, thus it is fundamental to characterize consistent and unambiguous measures. Several approaches that are directly related with the definition of a measure will be discussed in this paper.

Besides this introduction and the conclusion, this paper is structured as follows: four sections describing respectively the approaches to be compared, Kitchenham’s framework for software measurement validation, SQUID approach, Kitchenham’s method for modeling software measurement data, and the ISO/IEC 15939 software measurement process framework. The last section will be devoted to the comparison analysis and the criteria used for comparing. The main results are summarized in a table.

2. Kitchenham’s framework for software measurement validation

A framework for validating software measurement is proposed in [3]. It is based on the identification of the elements of the measurement and their properties, identifying how those elements are defined to construct a measure and define appropriate theoretical and empirical methods of validating those properties and definition models. The model, shown in Figure 1 is used to identify theoretical criteria supporting the measurement validation. A *measure* maps an empirical attribute to the formal, mathematical world. The *attributes* are the properties that characterize an entity. The *entities* are the objects observed in the real world. The model shows that for a given attribute, there is a relationship of interest in the real or empirical world, which should be captured formally in the mathematical world.

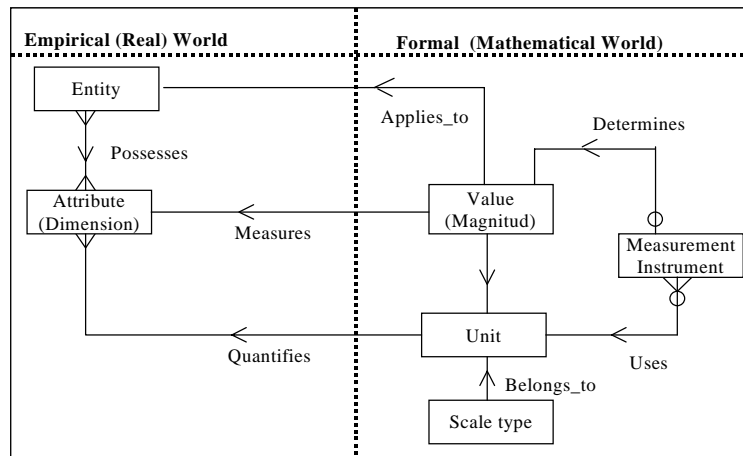


Figure 1. Structural model of measurement expressed in E-R notation [3]

A *measurement unit* determines how an attribute is measured. The model shows also that an attribute may be measured in one or more units. It is also clear that 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. When measurement units are considered, it leads to the necessity to understand the different *measurement scale types* implied by the particular unit. A unit’s scale type determines the admissible transformation applied to a particular unit. There is a one-to-one relationship between unit and scale type; however, a proof of this relationship is not offered. Rather it is confirmed by examples that the scale type is inherent in the unit not the attribute. The use of units is extended to allow the definition of the scale points for scale measures and the categories used for nominal scale measures. Thus, in the context of nominal and ordinal scale measures where the measures are mappings to arbitrary labels, a unit is suggested to ensure that such measures are used consistently. An attribute is measured by applying a specific measurement unit to a particular entity and attribute to obtain a value. Therefore, a *measured value* is interpreted when the entity where it applies, the measured attribute and the used unit, are known. The values are often numerical, but they do not have to be. For example, a module can be labeled “inspected” or

“not inspected”. However, for convenience, it often maps to numbers. Thus, the set {not inspected, inspected} can be mapped to the set {0,1} or equivalently to the set {1001,100}. It is important to point out that these values represent nominal scale measures and they are arbitrary labels; so they cannot be summed or averaged. The measures should be defined over a *set of permissible values*, which may be finite or infinite, bounded or unbounded, discrete or continuous. This model also shows that a *measurement instrument* may optionally be used to obtain the measured value of an attribute. Therefore, there may be many different measurement instruments available for a particular unit. For example, the height can be measured by using either a tape measure or variations in air pressure. Measurement instruments are also used to classify entities. For example, a genetic test can be used as a measurement instrument to determine the sex of an athlete.

The model presented in Figure 1 is appropriate for direct measures but cannot cope with indirect measures. In this case, according to [3] an *equation* must be defined to represent an indirect measure, which is formalized as a mathematical equation. The equation defining an indirect measure acts as a form of measurement instrument. The attribute(s) used in an equation may relate to an entity or entities different from the entity whose attribute will be measured indirectly. The equation can be based on an empirically observed association between attributes, (e.g. when program size is used in an equation to predict project effort) or it is based on a compound measurement unit (e.g. lines of code per hour is used as unit for measuring productivity). In this case, the equation is derived from the unit of the new attribute. 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. The indirect measures should be measured in all reasonable or expected situations; they should not exhibit unexpected discontinuities.

Other important elements involved in the measurement are *measurement protocols* and *entity population models*. They are not reflected in this model (Figure 1).

Conclusion on the Kitchenham’s software measurement framework

This model helps to identify a number of theoretical criteria that need to be satisfied by a valid measure. The following conclusions are derived:

- The model addresses the problems related with the identification of the elements involved in the measurement, their relationships and properties in order to construct reliable measures.
- It does not offer well-defined guidelines on how defining and storing appropriate data sets allowing the researcher and practitioner to make valid analysis.
- It does not define how to store data that involve several projects.
- The model is based on direct measurement, indirect measurement (multi-dimensional attribute and empirically observed associated among attributes).
- The values can be actual, target and estimates.
- It does not provide a thorough structure for defining the measurement protocols in order to get repeatable measures.

3. SQUID (Software Quality In the Development process) approach.

SQUID [1] is a method for software quality assurance and control, which includes a toolset to support application of the method. SQUID combines the process and product approaches to software quality. The method involves a number of quality activities to support the quality management during software development. It is based on a unifying model, which combines by measurement a product view, a data view, and a quality view of software.

The *product view* represents the software development model, that is, the SQUID approach requires a development model to be defined prior the use of the method. In the development model three types of entities (evaluation objects) are identified, such as are:

- Deliverables types: model documents and other software project tangible objects
- Development activity types: process models (specification, design, coding, testing, etc.)
- Event types: events models (start of project, delivery of specification, etc.)

The entity types are used to assemble a development model. During the development these are instantiated with real entities, i.e. the actual documents, source code, etc., which are produced during the development process. Appropriate relations among them must be provided for the purpose of data analysis.

The *quality view model* specifies how software quality characteristics are refined into sub-characteristics and attributes (measurable properties). The SQUID method is general and does not prescribe any particular quality model. The *data view model* specifies how the data elements are distinguished as actual values, target values, and estimate values. The SQUID view of measures is shown in Figure 2. It is consistent with the recommendations of [3]. SQUID uses two types of measure: *internal measure* relating to software and its development process and *external measures* relating to operational behavior and the support process. The term *attribute* is used to refer to the measurable properties of software products or projects, for example; product size, project duration, etc. the term attribute applies to internal and external software properties. *Internal attributes* are associated with the project object type or entity type in a development model. *External attributes* are associated with a product portion. SQUID uses the product portion concept to define the part of a product that has a distinct set of quality requirements.

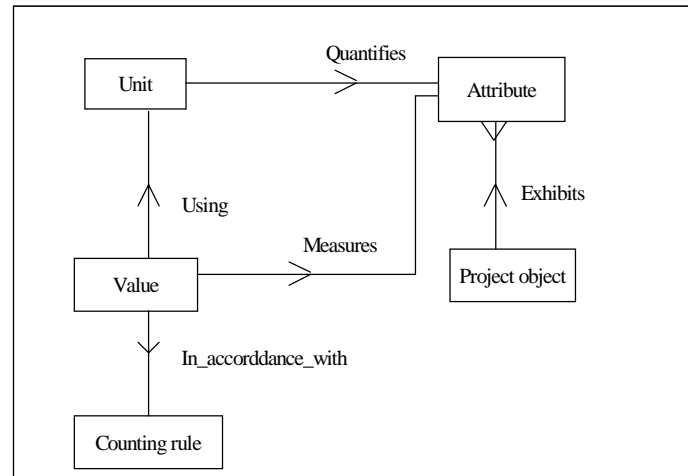


Figure 2. The SQUID view of measurement [1]

The term *units* refers to the units used to measure quantifiable attributes, for example lines of code is a product code length unit. The same attribute–unit pair can be related to different project object types (e.g. size in lines of code is associated both with module and final system). In addition, a particular project object type may have a particular attribute measured in two different ways (e.g. size can be measured on the final system either in terms of lines of code or in terms of the number of modules). The attribute *values* (i.e. measurement) are values corresponding to the attributes of a particular software object in the agreed scale. In order to ensure that measured values are repeatable and comparable, a counting rule must be defined. *Counting rule* defines the conditions under which a measure is obtained. The counting rule identifies any information that is needed to ensure that measures are collected in a repeatable and comparable manner. The values as mentioned above can also be targets; based on project constraints or requirements, or estimates besides actual values. The counting rule is similar to the measurement protocol defined in [3].

SQUID combines the development model, quality model and data model by means of software measurement (Figure 3). Each internal attribute belonging to a specified quality model must be associated to an appropriate project object type belonging to a specified development model and associating a unit, scale and counting rules to each attribute. This information is used to interpret the values. That is, for each value stored in the database, it is specified which unit and scale are used and how the value is obtained, i.e. the counting rule

The process of attaching attributes to entity types or project object types corresponds to define measures. In other words, the definition of measures is the glue that links the three SQUID models together.

The process of defining metrics based on these models is called *configuration* in the SQUID terminology. However, in the SQUID context, only a quality model is used by a project, but a quality model can be used by many similar projects.

Conclusion on the SQUID approach

Our study was based on the measurement model used to support the application of the quality activities. The following conclusions are derived:

- It extends the Kitchenham's framework [3], incorporating quality modeling, quality specification and quality control issues.

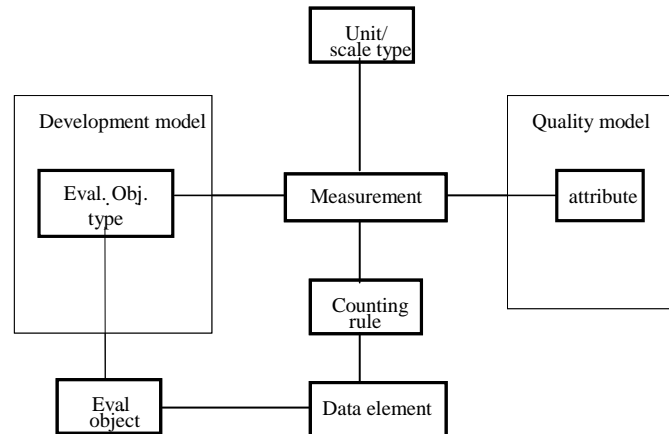


Figure 3. Linking a development model and a quality model [1]

- It supports data collation and storage.
- The measures provide the link between quality requirements and development process.
- It provides a framework that facilitates the consistent data collection over a number of development projects.
- It allows analysis of both the quality performance of individual projects and the process maturity of the company.
- It does not provide a thorough structure for defining the counting rule in order to get repeatable measures.

4. Kitchenham's method for modeling of software measurement data

In [4], a method for specifying models of software data sets in order to capture the definitions and relationships among software measures is proposed. One of the goals of the method is to provide a standard structure for defining software measures and a means of modeling complex data sets in order to be fully documented and exchanged with confidence. Complex data sets include values related to entities at different levels of granularity, past values of the same measure, values related to actual, estimates and targets, and to different scale types. Thus, the standard structure may contribute both to the definition of a company measurement program and to the exchange of data sets among researchers. The conceptual model is expressed in terms of the E-R model (Figure 4). It supports the following requirements: 1) keep each individual data set as a separate entity; 2) represent (model) the different structure of different data sets; 3) maintain the definitions (metadata) of the measure used in each data set and 4) allow the data collection at different levels of granularity (e.g. project data versus module or object data). The conceptual model is intended to allow the structure of software data sets to be accurately modeled and software measures to be fully defined. It also provides the basic logical data model for automating data modeling and storage. The metadata to be considered for a software data definition standard should include:

1. The definition of any entity being measured and the identification of the level of granularity of the entity.
2. The definition of each of the attributes measured on each entity.
3. The definition of each unit associated with each measurable attribute (textual or string attributes do not have a unit)
4. The definition of the counting rules associated with each attribute-unit pair in the context of the entity being measured (all attributes types need counting rules, even non-measurable ones).
5. Specification of the scale type associated with each attribute-unit pair.
6. For nominal and restricted ordinal scale, the definition of each category/scale point.

The metadata to be maintained about individual values include:

1. The type of value that is required: actual, estimate, and/or target.
2. The legal range of values for each value type.

3. Whether or not the individual data values can be overwritten.
4. Whether or not individual values for absolute, ratio, interval, and unrestricted ordinal scale measures map to the integers or the real.
5. The permitted range of values for each value type given the specific data collection context.

The conceptual model (Figure 4) incorporates a view of a measure similar to that proposed in [3], but corrects the problem of confusing an entity occurrence and an entity type. A software data set is defined in terms of a development model comprising a set of software entity types each of which have a set of associated measures (i.e. attribute–unit pair). The entities and measures define the structure of the data set. The real software entities observed in the project are instances of the entity type. The instances exhibit measured values, so these are linked to the entity instances. The conceptual model shows also three components or domains; the first two domains (Generic domain, Development model (DM) domain) define the metadata necessary to facilitate more complete and accurate reporting of software data. I.e. the *generic domain* defines *attributes* (Generic attribute), *units* (Generic unit), and *scale type* (generic scale range) independent of other considerations. Thus, common measurement concepts are available for use in many different development models. A generic modeling domain is concerned not only with the similarities between the measurement concepts used in different development models, but also with similarities among entities (DM element), it might include a Generic DM Element Entity. The DM domain provides the link between *measures* (DM element Measure type) and *entities* (DM element) of interest in order to provide a context for measurement. The characteristics of a development method may be captured by recording, as entity types of interest, the processes defined for the method, the products used or generated by those processes, and the resources consumed by those processes. Different DM may also exist

The measures defined in the DM Element Measure type (Figure 4) are derived from the Generic attribute and the Generic unit. The unit and the attribute can be redefined when the measure is created to identify DM specific issues. Similarly, the scale point definitions associated with a nominal or restricted ordinal unit in an occurrence of the DM Element Measure Type may also need to be redefined. The DM Specific Scale range (Figure 4) supports this requirement. A Measure Type can be used to support one or more *measure aspects* (i.e. Actual, estimate, or target), so a DM Measure Aspect entity is required for each measure. The DM measure Aspect includes information relevant to each aspect including: 1) counting rules used to ensure that collected values are comparable; 2) whether or not the values associated with the aspect should be overwritten when the values change. In short, the DM domain defines and records the metadata naming the types of entity that are to be the subject of analysis and the attributes of those entity types for which data are to be collected. The DM may be visualized as constituting a set of templates into which instances of the collected data must fit.

The last component (*Project domain*) represents the data values collected from real projects. It links data values (Recorded value) to actual instances of the entities (project object occurrence) that are defined in the development model domain, which are stored in a format that is derived from the metadata. Data may be collected for one or more project that conforms to the same DM. Instances where details of several projects are collected allow the larger granularity measures to be analyzed otherwise if data are only available for a single project, then only the smaller granularity measures would be applicable.

Conclusion on the Kitchenham's method

- It allows software metrics researchers to assemble trustworthy collections of software data from different sources in order to make valid analysis.
- It addresses the problems related to the collection, storage and analysis of the measures.
- It does not provide a thorough structure for defining the counting rule in order to get repeatable measures.

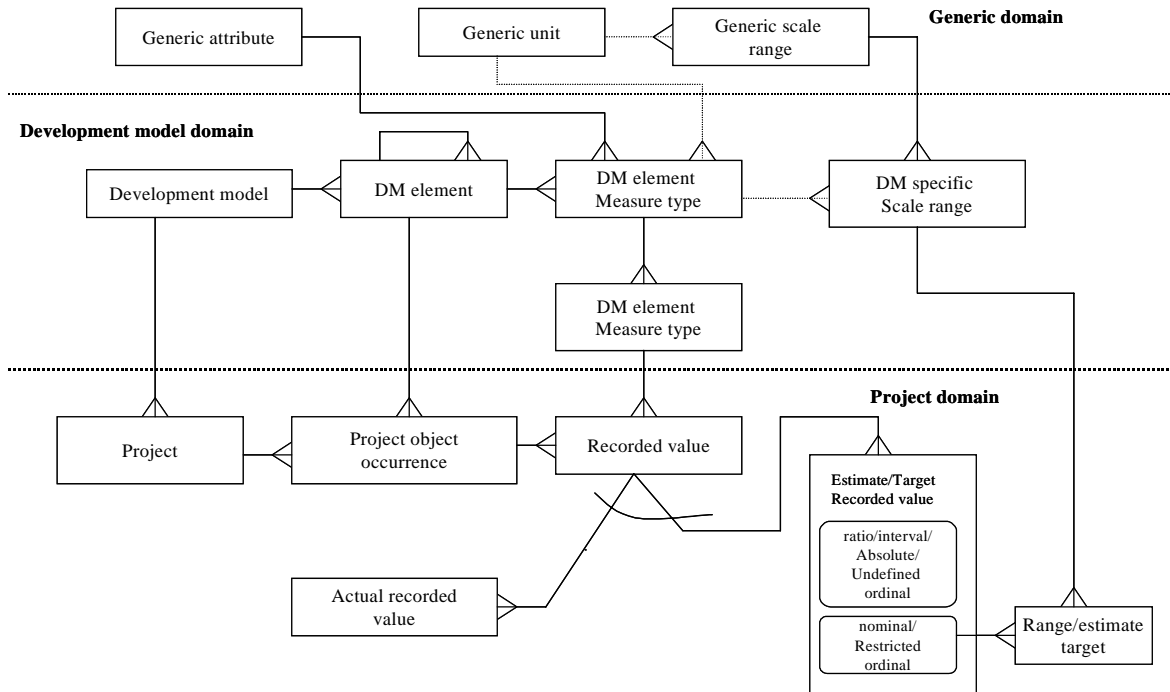


Figure 4. Conceptual model, expressed in E-R notation [4]

5. ISO/IEC 15939 Software engineering – Software measurement process framework

ISO/IEC [7] defines a software measurement process applicable to all software-related engineering and management disciplines. The process is described through a model (Figure 5) that defines the activities of the measurement process that are needed to adequately specify what measurement information is required, how the measures and analysis results are to be applied, and how to determine if the analysis results are valid. An *activity* is a set of related tasks that contributes towards achieving the purpose and outcomes of the software measurement process. A *task* is a well-defined segment of work. Each activity is comprised of one or more normative tasks. This standard does not specify the details of how to perform the tasks included in the activities. According to Figure 5, the software measurement process consists of four activities. The activities are sequenced in an iterative cycle allowing for continuous feedback and improvement of the measurement process. The measurement process model is an adaptation of the Plan-Do-Check-Act cycle commonly used as the basis for quality improvement. Within activities, the tasks are also iterative. Two activities are considered to be the core of the measurement process: *plan the measurement process* and *perform the measurement process*; these activities mainly address the concerns the user of the **measurement process** who is represented by an individual or organization that uses the *information product*. An information product is one or more indicator and their associated interpretations that address information need (for example, a comparison of a measured defect rate to a planned defect rate along an assessment of whether or not the difference indicates a problem). The core measurement process is driven by the information needs of the organization. An *information need* is an insight necessary to manage objectives, goals, risk, and problems. For each information need, the core measurement process produces an information product that satisfies the information need. The information product is conveyed to the organization as a basis for decision-making. The link between measures and information needs is described as the measurement information model. It is closely related to the GQM approach [8]. The *measurement information model* is a structure linking information needs to the relevant entities and attributes of concern. *Entities* are the objects (e.g. process, product, project or resource) that are to be characterized by measuring its attributes. This model describes how the relevant attributes are quantified and converted to indicators that provide a basis for decision-making. The selection or definition of appropriate measures is based on a *measurable concept*. A measurable concept is an abstract relationship between attributes of entities and information needs. Measurement constructs (*indicators*) are defined that links these attributes to specified information need. Each construct may involve several types or levels of

measures (i.e. base measure, derived measure). The measures should be documented by their name, the unit of measurement, their formal definition, the method of data collection, and their link to the information needs. The *scale and measurement method* affect the choice of analysis techniques or models used to produce indicators. The procedures for data collection should specify how data are to be collected, as well as how and where they will be stored. The data analysis method(s), and format and method for reporting the information products should be specified. The other two activities (Figure 5), *establish and sustain measurement commitment* and *evaluate measurement*, provide a foundation for the core measurement process and a feedback to it. These two activities address the concerns of the **measurement process owner**. The *technical and management processes of an organizational unit or project* activities are not within the scope of this standard. A *Measurement experience base* must be included in the cycle. This artifact is intended to capture information products from past iterations of the cycle, previous evaluations of information products, and evaluations of previous iterations of the measurement process. This would include the measures that have been found which are useful in the organizational unit. No assumptions are made about the nature or technology of this measurement experience base, only that it must be a persistent storage.

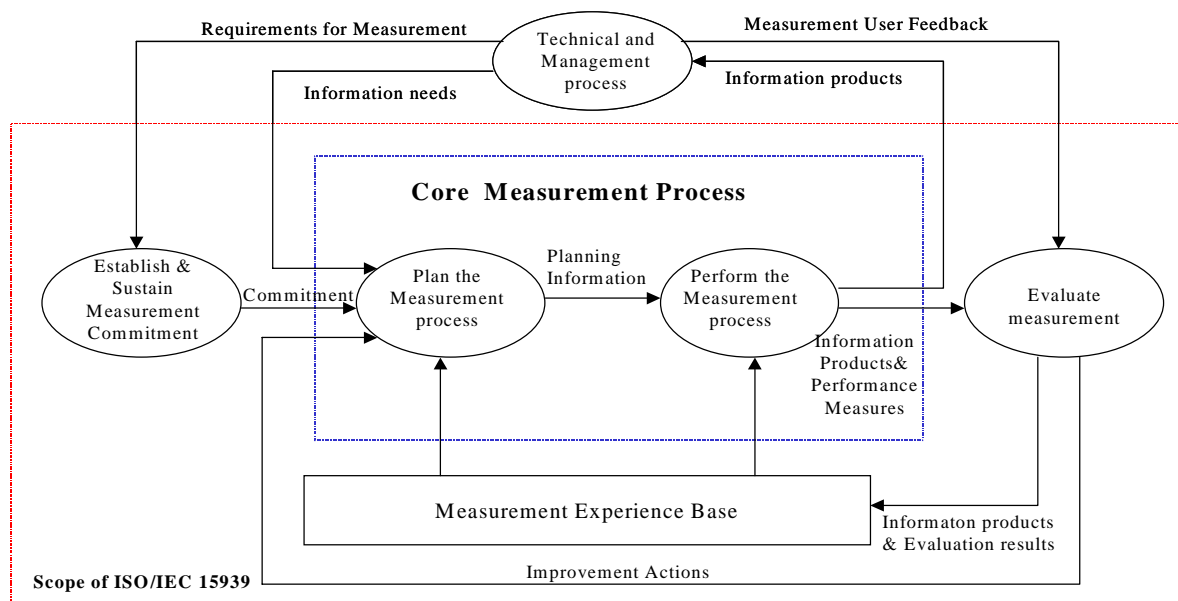


Figure 5. ISO/IEC 15939 Software Measurement Process Model [7]

Since the process model is cyclical, subsequent iterations may only update measurement products and practices. This standard does not imply that measurement products and practices need to be developed and implemented for each iteration of the process. The wording used in this standard adopts the convention that the measurement process is being implemented for the first time (i.e. the first iteration). During subsequent iterations, this wording should be interpreted as updating or changing documentation and current practices. The standard does not assume or prescribe an organizational model for measurement.

Conclusion on the ISO/IEC 15939 approach

- It describes the activities that must be performed when a measurement process is implemented.
- It indicates how the measures should be documented.
- It does not define a thorough data model of how the elements that define a measure are related and how to establish the data set to encourage the analysis of the collected data sets.

6. Comparison of the Measurement – Based approaches.

In the previous sections, four approaches involving software measurement have been described and discussed. On the basis of the above discussion, the following criteria for comparing the four approaches in Table 1 have been selected: 1. Purpose of the approach. 2. Measurement terminology used. (the definition of the term will be given, in case of being present): Entity type, Entity, Attribute, Unit, Scale, Scale type. 3. Measure definition (the definition suggested by each approach will be given). 4. Collection process of the

measure (the collection process for each approach will be given). 5. Types of measures (the types of measure considered for each approach will be given). 6. Types of measurement values (the types of measurement values considered by each approach will be given). 7. Software data sets definition (the mechanism used by each approach will be given). These criteria were chosen for their direct relation with measure definition. Other criteria related to the application of the approach could have been selected such as product/process, domain, development phase. We have not included them in order to abridge this presentation.

Criteria	Framework for Software Measurement Validation	SQUID Approach	Method for Modeling Software Measurement data	ISO/IEC 15939 Software Measurement Process Framework
1. Purpose	Describe the objects involved in software measurement and their relationships.	Describe the objects involved in the software measurement to support the application of the quality activities.	Describe data sets to model the measures definition	Collect, analyze and report data related to software product and process.
2. Measurement terminology				
2.1 Entity type	Not present	It is called <i>evaluation object type</i> linked to a development process.	A specific object in the real world linked to a development process.	Not present.
2.2 Entity	The objects observed in the real world.	It is called <i>evaluation object</i> . (Represent the instantiation of an evaluation object type).	Represent the instances of a given entity type	The objects observed in the real world.
2.3 Attribute	Properties that an entity possesses.	A measurable property of an object.	Properties to be measured on the entities.	Property or characteristic of an entity.
2.4 Unit	Determine how an attribute is measured	The way to quantify an attribute.	The way of measuring an attribute.	A particular quantity, defined and adopted by convention
2.5 Scale	Range of values for each value type.	Range of values for each value type.	Range of values for each value type.	Ordered set of values, continuous or discrete, or a set of categories to which the attribute is mapped.
2.6 Scale type	Defined for the class of admissible transformations implied on a specific unit.	Defined for the class of admissible transformations	Defined for the class of admissible transformations implied on a specific unit.	Defined on the nature of the relationship between values on the scale.
3. Measure definition	{Entity}x{attribute}x{unit/scale type}	{Attribute}x{Scale_type/Unit} x {Evaluation_Obj_type}	{Entity type} x {Scale_type/Unit} x {attribute}	{name, unit, formal definition, method of data collection, and their link to the information needs}
4. Collection process of a measure	Measurement protocols Measurement instrument	Counting rule	Counting rule	A procedure that should specify how data are to be collected, as well as how and where they will be stored.
5. Types of measures	Direct Indirect (empirically observed association between attributes and compound measurement unit)	Direct (internal measure) Indirect (internal, external measures and indicator)	Direct measure.	Base measure, derived measure and indicator.
6. Types of measurement values	Actual, target and estimates	{Target, estimate, actual} x {absent, single, multi}	{Target, estimate, actual} x {absent, single, multi}	Actual, estimate
7. Software data sets definition	Entity population models	Defined in terms of a development model comprising a set of software entity types	Defined in terms of a development model comprising a set of software entity types.	Not defined explicitly.

Table 1. Comparison of the software measurement-based approaches

The criteria listed above are presented as entries in Table 1, showing the comparison for each approach to software measurement studied in previous sections and summarizing the extent to which the four different software measurement-based approaches handle the elements involved in the software measurement. A

clear idea of the state of the art or progress in software measurement is observed, in the sense that the approaches studied improve the weakness of previous researches. A precise terminology was appreciated in the SQUID approach [1] and the Kitchenham's method [4], according to the criteria established in Table 1. However, the structure model defined in [4] is oriented towards the definition and collection of direct measurements. With respect to indirect measurement, they suggest that any automate data modeling facility should support the construction of appropriate data sets by concatenating data values from related entities. Kitchenham's framework [3] suggests that an indirect measure must be defined through of a mathematical equation or model. Table 1 shows also that a procedure (counting rule or measurement protocols) to specify how data are to be collected, must be defined. However, a structure for the counting rule is not defined. On the other hand, the SQUID approach [1] and the Kitchenham's method [4] clearly define how data collection maps to the software development, while Kitchenham's framework [3] is limited to define a measure. ISO/IEC 15939 [7] offers a more general view, it propose a set of activities that must be performed when a measurement process is required in a given organization. A data model for software measurement is not defined by ISO/IEC, but it specifies the elements that should be defined for documenting and collecting a measure. Moreover, the data models defined in [1] and [4] could be used in [5] in order to get a precise description of a measure.

7. Conclusion

Relations among the four approaches have been appreciated according to the criteria defined in Table 1. This study shows that each approach addresses some of the weakness of the previous researches in order to obtain well-defined measures and to ensure that they are reliable. They share the fact of considering that, in order to ensure that different data collectors can collect comparable measures, it is necessary to define all the elements that give rise to the recorded data values. However, some flaws and shortcomings have been identified, such as the lack of a precise definition of the counting rule. In spite of considering that the counting rule to assign value to the attribute is essential to ensure repeatable and comparable values, a formalization of its definition has not been provided until now. They only suggest identifying the role responsible for data collection, the stage in the development process when the data is to be collected, and any data extraction tools to be used. On the other hand, since software measurement is essentially a measurement, the principles and practices of software measurement should therefore be in accordance with those of the measurement theory [9], which is the basis of scientific measurement. However, a weakness of the use of these principles and practices was appreciated on the approaches studied. It can be appreciated that an important research issue is the formalization of the counting rule, in order to avoid the collection of invalid or incomparable measures. These results summarizing the observed weakness are actually being used to define a new data model for software measurement. This model is part of an ongoing research, which will be published in the near future. The new model should help to establish a standard terminology and formalism for defining software measures and to apply these in a consistent manner over an extended period of time, towards a more sound engineering practice of software.

References

- [1] Boegh J., Depanfilis S., Kitchenham B. and Pasquini A. "A method for Software Quality, Planning, Control and Evaluation", IEEE Software, 69-77, March/April 1999
- [2] Fenton N.E. and Pfleeger S. L.. "Software Metrics. A Rigorous & Practical approach". Second Edition 1997.
- [3] Kitchenham B., Pfleeger S. L., Fenton Norman. "Towards a framework for software measurement validation", IEEE Transactions on Software Engineering, December 1995.
- [4] Kitchenham B., Hughes Robert and Linkman S. "Modeling Software Measurement Data". IEEE Transactions of software engineering. Vol. 27, N° 9, September 2001.
- [5] ISO/IEC Draft 9126-1.-2. "Information Technology - Software Product Quality. Part 1: Quality Model", 1998.
- [6] ISO/IEC 14598-3. "Information Technology - Software Product Evaluation - Part 3: Process for Developers. Software Engineering", June 1998.
- [7] ISO/IEC FCD 15939 "Software Engineering-Software Measurement Process Framework", January 2001.
- [8] Solingen, R. van and Berghout, E., "The Goal/Question/Metric method: a practical handguide for quality improvement of software development". McGraw-Hill Publishers, 1999.
- [9] Zuse H. "A framework of software Measurement", Walter de Gruyter – New York 1998.