Foundations for a Comprehensive Approach to Software Quality Assessment

2

The objectives of this chapter are threefold:

1. to motivate the necessity of establishing a software quality assessment program based on fundamental software engineering concepts,
2. to present the OPA Framework with Software Quality Indicators (SQIs) as a sound comprehensive approach to quality measurement, and
3. to relate process and product measures to the goals of prediction and assessment.

2.1 Establishing a *Quality* Focus

To determine the rainfall for any particular day one could periodically measure humidity and temperature for the selected day and then, based on established physical laws, compute how much rain one would have expected to have fallen. Similarly, one might advance conjectures about the quality of a product based on influencing measures like conformance to schedule, productivity, and cost estimates. In both cases, the substitute measures are "somewhat related" to the stated measurement goal, but lack that definitive connection which directly relates the measurement process to the final objective. For example, a more accurate process to measure rainfall is to place a measurement device outside to directly collect and document the amount of rain that actually falls. Moreover substitute measures can often lead to false conclusions. Schedule slippage, for example, can and often does adversely impact product quality. Can one draw the reasonable conclusion then that if a project *is* on schedule, quality is present in the product? The obvious answer is, "No."

In concert with the above observations, we offer the following guidance as preliminary steps to establishing a software quality assessment program:

1. identify *software quality* as the major goal of the underlying measurement process, and
2. define measures that
   - are objective,
   - *directly* related to software quality, and
   - reflect inherent characteristics of the *software engineering* process.

Emphasizing quality measures that reflect characteristics of the software engineering process is of particular importance because it focuses attention on the domain from which such measures are extracted. More specifically from a software engineering perspective, software quality is not about efficiency, scheduling, cost or even functionality. To repeat an earlier assertion, these are *systems* engineering objectives that place *constraints* on the software engineering process, and subsequently, on the achievement of software quality goals. For example, to achieve mandated timing requirements in a real-time decision support system the software engineer might employ the use of global variables for inter-module communication. Although necessary to meet timing constraints, the use of global variables for inter-module communications is detrimental to maintainability. Similar examples can be cited for schedule, cost and functionality. Recognition of the differences between systems and software engineering goals is crucial, and the achievement of specific software engineering objectives is often constrained by the "givens" established at the higher systems engineering level. In turn, such recognition enables one to focus attention on the identification and definition of measures derived from the trends and artifacts of the software engineering process which more accurately reflect product quality.

The remainder of Chapter 2 expands on the guidance provided above by identifying and describing a framework that characterizes the software engineering process, while serving as the focusing agent for measuring software quality. Software quality indicators (SQIs) are also presented. SQIs play an integral role in the definition of quality measures reflecting the presence (or absence) of desirable product attributes. Additionally, we discuss the impact an established (or proposed) process model can have on the identification and definition of quality measures, and finally, distinguish between the measurement goals of assessment and prediction.

2.2 The Objectives/Principles/Attributes (OPA) Framework

The rationale of the Objectives/Principles/Attributes (OPA) Framework (Arthur and Nance, 1990) is briefly described in Chapter 1; more detail is given here. As illustrated in *Fig. 2.1*, the framework enunciates definitive linkages among project-level objectives, software engineering principles, and desirable product attributes, advancing the following rationale for software development:

- a set of *objectives* can be defined that correspond to project-level goals and objectives,
- achieving those objectives requires adherence to certain *principles* that characterize the process by which the product is developed, and
- adherence to a process governed by those principles should result in a product that possesses *attributes* considered to be desirable and beneficial.

Underlying this rationale is a natural set of relations, depicted in *Fig. 2.2*, that link individual objectives to one or more principles, and each principle to one or more attributes. For example, to achieve maintainability one might employ the principle of information hiding in the development process. In turn, employing information hiding will result in a product that exhibits a well-defined interface.

The OPA Framework differs from other structurally similar frameworks, e.g. McCall's Factor/Criteria/Metric (McCall et al., 1977) and Basili's Goal/Question/Metric (Basili and Rombach, 1988), in that all OPA measures are linked to project-level objectives through software engineering *principles* that guide the software development process. Analogically, principles function like a fulcrum, providing the supporting capability reflected in the software attributes to lift the product in attaining the designated objectives. More specifically, principles

- provide the foundational definition of the desired or proper process for developing software, and
- enable one to reason about and identify those activities that contribute to or adversely impact the software development process.

How does one determine if, and to what extent, a product possesses desirable attributes? The answer lies in the observation of product properties, i.e.

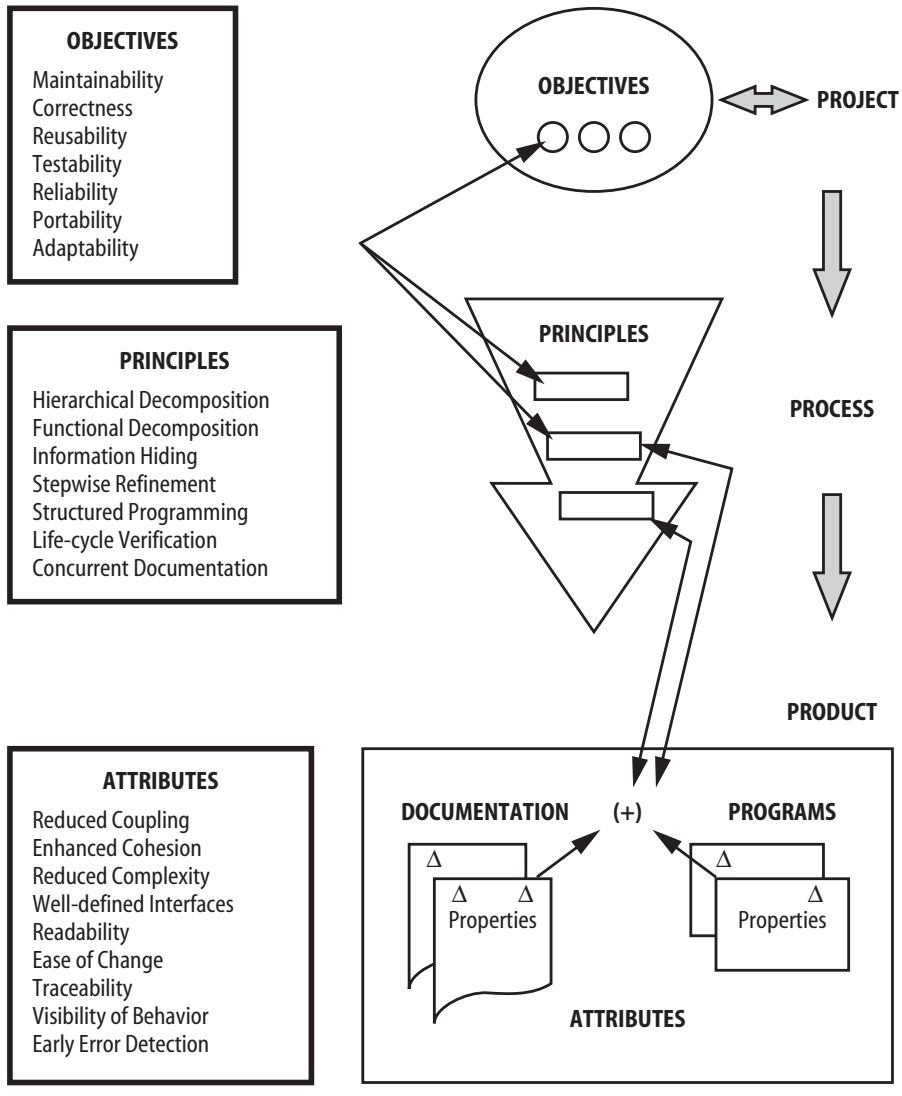

Figure 2.1 *Illustration of the relationship among objectives, principles and attributes in the software*

observable characteristics of the product. For example, the use of global variables indicates that a module interface is not well-defined (Dunsmore and Gannon, 1980, p. 149). More specifically, the number of global variables used relative to preferable forms of communications, e.g. parameter passing, indicates the *extent* to which the interface is ill-defined.

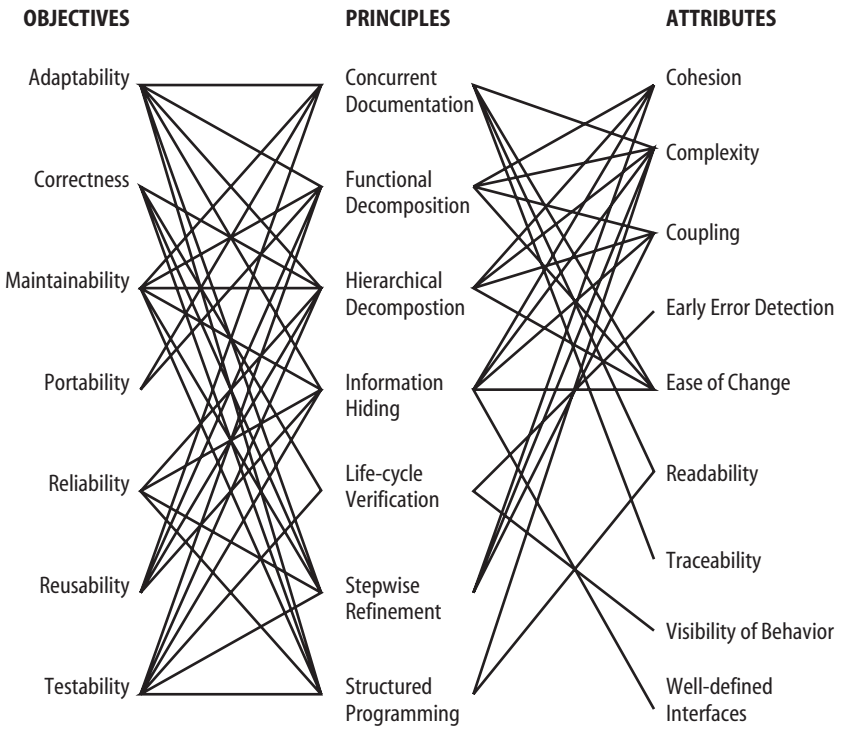

Figure 2.2 *Linkages among the objectives, principles and attributes*

Implementing an effective quality measurement program mandates a systematic approach that reflects the best current software engineering practices. We recommend an approach that embraces the OPA Framework as a basis. Through its attribute/property pairs and linkages relating attributes to principles and principles to objectives, the OPA Framework supports a well-defined, *systematic* approach to *examining* product and process quality. The OPA Framework *definitively* links the achievement of software engineering objectives to the use of specific principles, and the use of such principles to the realization of desirable attributes in the product. Subsequently, by observing product properties to determine the extent to which desirable attributes are present in the product, one can determine the extent to which particular principles are governing the development process and, in turn, the extent to which stated software engineering objectives are achieved.

Moreover, guided by an OPA characterization of the software (both the artifacts and the development or sustainment process), one can analyze and examine relationships in the interpretation of quality measures. For example,

if one observes a value indicating a low degree of achievement for a software engineering objective (not consistent with expectations), then contributing principles are examined (based on the defined linkages among objectives and principles) for anomalous values. Similarly, the linkages among principles and attributes point to candidate attributes to be examined to identify the contributing source(s). Finally the attribute/property relations enable the identification of the most prominent process or product characteristic(s) influencing the original objective value. The identification of an anomalous value for an attribute/property pair indicates the misuse (or omission) of a critical software engineering principle. The points where this principle is most utilized in the process become the prime candidates for attention. With appropriate reporting one can also determine if the offending product component(s) are isolated or the problem is widespread.

2.3 Software Quality Indicators

The OPA Framework and its enunciated rationale binds measurement and measurement interpretation to a realistic characterization of how software is actually produced. Below, we describe the concept of Software Quality Indicators (SQIs) that reflect an OPA perspective and provide a sound basis on which quality measures are defined.

2.3.1 Establishing a Basis for Measuring the Unmeasurable

"Software quality factors," "software quality metrics" and "software quality indicators" – are all terms used in the conviction that the quality of the software product should be measurable, at least in a relative sense. In a paper by Kearney et. al., (1986) the authors issue a rather compelling criticism of the inadequate basis for measuring software complexity and of the shortcomings of experimental research intended to support complexity metrics. We share the opinions of Kearney and his colleagues, and propose the use of statistical indicators as the basis for scalar determination of product and process characteristics. The motivation for using statistical indicators of software quality stems from the qualified successes in applying them to unmeasurable economic and social concepts. This motivation, as well as extension of the applicable theory to the derivation of software quality indicators, is described below.

Both economic and social indicators are based on the premise that *directly unmeasurable* qualitative conditions can be indirectly assessed by

measurable quantitative characteristics. The economic indicators of a "good or improving economy" are routinely discussed in business news. Social indicators like "safe streets" are often cited as contributing elements of policy decisions. Meier and Brudney provide an instructive definition for social indicators that serves as the foundation for our definition of software quality indicators (Meier and Brudney, 1981, pp. 95–96):

> An *indicator* is a variable that can be measured directly and is linked to a concept through an operational definition. An *operational definition* is a statement that tells the analyst how a concept will be measured.

Two important characteristics of social indicators are stressed by Carley (1981, p. 2):

- Social indicators are "surrogates" that do not stand by themselves – a social indicator must always be related back to the unmeasurable concept for which it serves as a proxy.
- Social indicators are concerned with information, which is conceptually quantifiable, and must avoid dealing with information, which cannot be expressed on some ordered scale.

The parallels which can be drawn between the concept of social indicators and that of software quality indicators are: (1) both attempt to measure the "directly unmeasurable" through the use of surrogate (or substitute) measures that are directly observable, and (2) an undeniable relationship must exist between the surrogate measure and the concept being measured.

2.3.2 Applying the Social Indicator Concept to Software Quality Measurement

The concept of software quality indicators is a natural extension of the use of statistical indicators in the social sciences. The need arises from the fact that certain characteristics cannot be measured directly and require surrogate measures in order to obtain quantitative assessment (Carmines and Zeller, 1979, pp. 9–11). An example in software is the measurement of cohesion, which cannot take a simple direct form; thus, the need exists to define an indicator that can reflect either desirable (high) or undesirable (low) cohesion in a software component. Multiple indicators can perform confirming and contrasting roles to permit a "hardening" of the softness typically associated with this indirect form of measurement.

Software quality indicators are embodied in the OPA Framework through attribute/property relationships. For example, an intangible attribute of the

development process, like early error detection, can be indirectly assessed through measurable properties, like the changing of requirements after the software specification review. For clarification purposes, we note that our use of the term "Software" in "Software Quality Indicators" is not intended to be restrictive, but applicable to both process and product quality indicators.

A ***Software Quality Indicator (SQI)*** is a variable whose value can be determined through direct analysis of product or process characteristics, and whose evidential relationship to one or more attributes is undeniable (Arthur and Nance, 1987, p. 25).

Crucial in this working definition is that

- the value is *directly* measurable through the analysis of the software development process or products of that process, e.g. programs and documentation, and
- SQIs are *always* attribute/property pairs denoting *undeniable* relationships, and indicative of the *presence* or *absence* of one or more attributes.

Consider, for example, an SQI based on code analysis: coupling through the use of structured data types (CP/SDT). The property in this SQI is the use of structured data types, and the attribute is coupling. One can argue that the use of a structured data type as a parameter argument has a detrimental impact on module coupling. That is, structured data types allow the consolidation of data items perceived to be related in a given context. When passed as a parameter, however, rarely does the calling module access every data item in the structure. Consequently, these extraneous items, from the perspective of the calling module, unnecessarily increase the coupling between the calling and called modules (Troy and Zweben, 1981, p. 115). A candidate measure for this coupling is the ratio of the number of structured data types used as parameters relative to the total number of parameters:

$$\text{Structured Data Types Passed as Parameters/Coupling} = \frac{\text{\# of SDTs in Parameter List}}{|\text{Parameter List}|}$$

where | Parameter List | is the number of parameters in the parameter list (the cardinality function).

Note that: (a) the value is directly measurable, (b) the SQI is an attribute/property pair, (c) the relationship described between the use of structured data types and coupling is undeniable (and intuitive), and (d) the stated SQI can indicate the presence (or absence) of coupling between two modules.

To summarize, we want to measure quality in terms of characteristics set forth in the OPA Framework, i.e., project-level objectives, process principles, and desirable product attributes. Product attributes, although still not directly measurable, are significantly less abstract than process principles and project objectives, and serve as the basis on which software quality indicators are defined. More specifically, we identify process and product properties that: (a) are directly measurable, and (b) undeniably reflect the presence (or absence) of specific process and product attributes. In turn, these measures are propagated along the linkages defined by the OPA Framework, yielding subsequent measures reflecting the proper use of process principles and the achievement of stated software engineering objectives.

Assuming that valid quality indicators can be formed from quantifiable characteristics of the process, code and documentation, then automatic or semi-automatic (human assisted) procedures can be developed to assess software quality (Nance and Arthur, 1994).

2.3.3 Measuring Characteristics of Process and Product

Because software evolution begins with requirements specification activities and continues *throughout* the life of the product (including attendant maintenance activities), SQIs must embrace both process and product measures, and ideally, must admit to at least semi-automatic computation. As illustrated in *Fig. 2.3*, we propose the use of SQIs throughout the product software life-cycle. Initially SQI measures must reflect process characteristics because little, if any, product is available. As development continues and products become more readily available, SQI measures should expand correspondingly to reflect product characteristics. Preliminary work in the SQI domain suggests that process, documentation and code indicators are needed (Arthur et al., 1991, p. 5).

Measuring Quality Through an Accumulation of Evidence

Software quality measurement should not be based on a single measure. If such a measure attempts to incorporate many aspects of quality, it becomes unwieldy and unintuitive (Gaffney and Cruickshank, 1980). If it focuses on a single product or process characteristic, e.g. McCabe's Cyclomatic Complexity Measure, then pertinent information is inappropriately constrained, providing only a limited view of product quality. The SQI determination, embedded within the OPA Framework, however, is predicated on the exploitation of multiple measures, each attesting to the presence or absence of particular

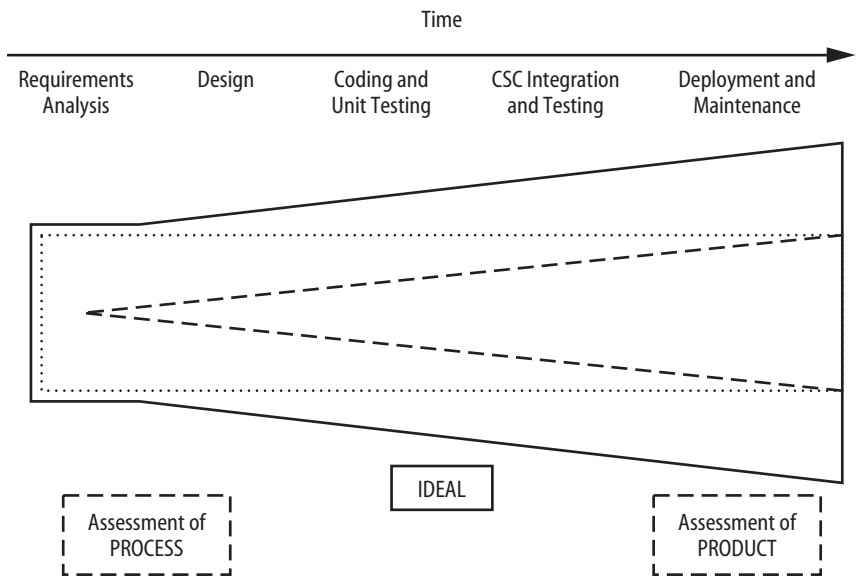

Figure 2.3 *Exploiting both process and product indicators*

attributes in the product. OPA embraces the philosophy that demonstrating that software possesses a desired attribute (or does not) is not a proof exercise; rather, it resembles an exercise in civil litigation in that evidence is gathered to support both contentions (the presence or absence) and weighed on the scales of comparative judgment (Nance et al., 1986; Nance and Arthur, 1994). As illustrated in *Fig. 2.4*, measures reflecting the absence of an attribute provide values in the -5 to zero range; measures attesting to the presence of a desirable span the range of zero to $+5$. Returning to an earlier example, if we consider the extent to which a product exhibits a well-defined interface, the use of global variables for inter-module communication has a detrimental impact. The use of parameterized calls, on the other hand, supports such a contention. Hence, for any given product attribute the aggregation of multiple confirming and contrasting measures yields one value in the range $(-5, +5)$ indicating the degree to which a desirable attribute is present or absent in the product. Note that values falling in the designated range $(-0.5, 0.5)$ might occur because evidence of both presence and absence is detected or because no evidence is available (which results in a zero).

In effect, the SQI approach offers four substantial advantages over the single metric approach to software quality measurement: (1) multiple measures, (2) measures which confirm or refute the existence of a quality attribute, (3) a relative measurement scale reflecting consistency of judgment and (4) measures

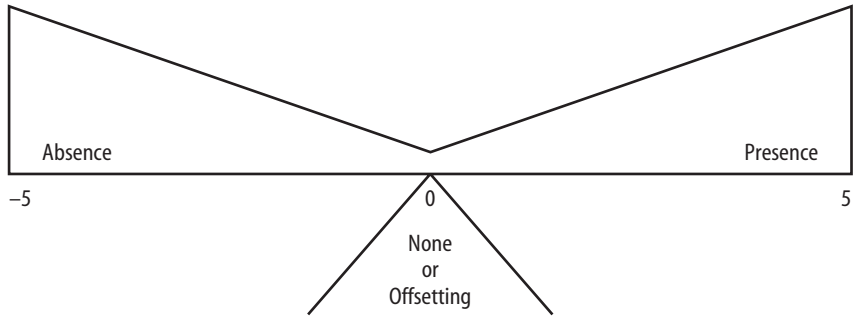

Figure 2.4 Measurement scale

that are simple and intuitive. Sections 3.3 and 3.4 outline systematic procedures for defining and interpreting software quality indicators.

2.4 Influences of the Process Model

Within an established development process, well-defined procedures and guidelines serve as the basis for structured activities supporting product development while emphasizing specific organizational goals. Among organizations such goals usually emphasize similar objectives, i.e., producing a quality product on time and within budget; their development processes, however, often vary in approach and magnitude. For example, one organization's process might employ the conventional waterfall approach, while another might follow an incremental approach guided by critical path analysis. Although the OPA Framework, with the SQI concept embedded, is defined independently of any particular software development methodology, its application must be tempered by the realities of the prevailing process model underlying the development effort. In effect, the process model and attendant activities prescribe artifacts and timing, i.e., the focus of measurement.

Consider, for example, an organization that employs an incremental approach to software development, and desires only to examine code for quality characteristics. One possible approach is to analyze each code unit when it is first placed under configuration management (CM). While such an approach meets its intended objectives, i.e., providing the software engineer and program manager with quality-related information, it constrains the measurement process to focus primarily on code assessment, and correspondingly, on those activities related to placing code under CM.

Clearly, a more inclusive picture of quality could be obtained if assessment includes an examination of the design document before coding begins and a tracking of software trouble reports (STRs) written against the code after it is placed under CM. Nonetheless, practical considerations, such as limited resources and implementation deadlines, often dictate sub-optimal quality assessment procedures. Similarly, particulars of the development process can, and do, define when and where measurement activities are feasible. In effect, tradeoffs must be made to balance the benefits of additional quality assessment (and prediction) with the organizational costs associated with producing and collecting such data.

Crucial to the above observations is that, in establishing a measurement program, one must balance needs with cost and practicality. To do so, one first examines the process model to determine where each necessary data element can be obtained, and then, based on organizational constraints and priorities and on the practicality of being able to collect the requisite data elements, one identifies those data collection points that yield the most "bang for the bucks." Once the appropriate "where, when and what" are determined, the OPA Framework offers an appealing approach to establishment of an effective measurement program. More specific discussion of the effects of the process model on the application of the OPA Framework is given in Chapter 8 (Section 8.1).

2.5 Establishing Measurement Goals: Assessment or Prediction

Within the framework of software quality measurement two complementary concepts exist: *quality assessment* and *quality prediction*. Quality assessment entails an examination of the product for characteristics deemed desirable and beneficial *after* the product is developed. Quality prediction, on the other hand, focuses on the examination of artifacts that enables one to infer, with confidence, the extent (or probability) that a product will possess desirable quality characteristics *before* development is completed.

In establishing a measurement program, an a priori determination of the purpose is necessary: assessment, prediction, or both. Such determination is crucial because process instrumentation can differ depending on the purpose. In particular, assessment requires an examination of the product, while prediction focuses on an examination of process artifacts. Product code and documentation are examples of the former; software development folders and process trends exemplify the latter. Our experience has shown that predictive measurement, while having the greatest potential for controlling

quality, is the more difficult and costly of the two to achieve. Predictive measurement requires process artifacts which are the hardest to identify and collect because: (1) they are non-standard and often amorphously defined, and (2) no two development processes are identical, making the direct application of procedures developed by others difficult, awkward and at best only partially effective. Recalling our admonition against trying to do too much (Section 1.2), we suggest that the start of a measurement program adopt assessment as the initial purpose, but with the understanding that *both* assessment and prediction form the ultimate goal.