There is no such thing as absolute value in this world. You can only estimate what a thing is worth to you.


6.1 Introduction

Measurement for measurement’s sake is a waste of time and money. It is not unusual for people to measure things simply because somebody—some edict, or some policy—stipulates that things should be measured. Yes—measurement certainly has a role to play in making software development successful. But, unless this role is thought through, measurement can degenerate into a meaningless exercise, as the cartoons in Figure 6–1 suggest. The purpose of this chapter is to help you set up a role for measurement that makes sense for your environment.

In the world of measurement, the term meaningless has a number of nuances that we briefly explore. Our purpose for this brief exploration is to arrive at what we believe is a fundamental measurement principle. Let us consider the following:
It is meaningless to try to measure lengths down to the nearest sixteenth of an inch with a ruler that contains only quarter-inch marks. This nuance underlies the dialog in the first cartoon in Figure 6–1. In this cartoon, the figures “53%” and “56%” are essentially the same, lacking any other information about the accuracy of the statistical measurements referenced.

It is meaningless to try to measure things in a vacuum. There are many ways to set measurement context. In this book, we take the stance that measurements should be taken for the purpose of answering specific questions. In the second cartoon in Figure 6–1, the intended audience of the measurement presentation may have a question that he wants answered, but he is evidently hoping that the statistics he is seeing are not the answer he may be looking for.

It is meaningless to express measurements in language the intended audience cannot understand. We use the term foreign language to express the idea that a means of communicating is not part of someone’s or some entity’s vocabulary. It clearly makes no sense for someone to listen to a presentation in, say, Spanish, if that person never studied Spanish. Likewise, it makes no sense to communicate measurements in a language that may be foreign to an intended audience. For example, if the intended audience is conversant with the language of statistics, it is certainly appropriate to use statistics to express measurements (with the caveat cited above). If, on the other hand, statistics is a foreign language for the intended audience, using terms such as “mean,” “mode,” and “standard deviation” will be meaningless (unless, of course, the presentation includes a tutorial on statistical concepts).

The preceding discussion leads us to the following fundamental measurement principle that underlies much of the discussion in this chapter:
Thus, measurement, like many of the other techniques in this book, is an exercise in effective communication.

We present measurement techniques that enable you to measure software products and software systems development processes in everyday terms familiar—and therefore meaningful—to your organization. We believe that understanding how to define, collect, use, and communicate measurement is a significant contributor to successful software projects. Furthermore, we believe that successful software systems development is a continual improvement exercise. Measurement is a means for effecting this improvement.

Figure 6–2 shows our conceptual framework for measuring software products and the process used to develop these products. We focus on two concepts—product integrity and process integrity. Product integrity can be defined in terms of product attributes and attribute value scales. For each product attribute, an attribute value scale is defined in everyday terms familiar to your organization. Similarly, process integrity can be defined in terms of process components, component activities, and activity value scales. The revolving arrows represent the interaction between product and process measurements. Understanding such interactions helps refine your measurement activities. As explained in this chapter, measuring product integrity and process integrity enables you to measure the “goodness” of the products and the “goodness” of the software systems development process used to develop the products.

To implement the conceptual framework shown in Figure 6–2, we use a generalized measurement technique that we developed called Object Measurement®. This technique can be used to quantify almost any object. In the most general case, we measure an object through its characteristics. For products, we called these characteristics attributes; for processes, we called these characteristics components and activities. Of course, there is nothing magic about these labels; you can call the product and process entities that you want to measure anything you want. We use these labels to illustrate product and process measurement. The translation of these labels to your labels should be straightforward.

Measurements need to be expressed in everyday terms that are familiar to the organization; otherwise, they may, at best, be of little value.
Thus, as Figure 6–3 indicates, we describe how to make measurement of product “goodness” and process “goodness” happen in your organization by using the general measurement technique OM. Through worked-out examples, we describe how to apply the technique. More important, these worked-out examples illustrate down to the “how-to-do-it” level one of the fundamental points about measurement made at the outset of this chapter—that measurements should be taken for the purpose of answering specific questions.

We wish to make an additional key point about the measurement approach in this chapter. We show how OM makes it easy to integrate a measurement program with the organization’s way of doing software systems development business. That is, we offer you guidance on how to blend measurement activity with the software systems development process framework that we introduced in Chapter 3. An apt analogy here is that of the various gauges on an automobile dashboard. These gauges help you determine such things as

Figure 6–2 This figure shows our conceptual framework for product and process measurement. Product measurement involves identifying product attributes ($a_t$) and corresponding value scales of interest to the organization. Process measurement involves an additional layer of decomposition. Processes are decomposed into components ($x_t$) and component activities ($x_{tj}$). Value scales are defined for each activity in terms that are meaningful to the organization.
In this chapter, we offer you guidance on how to measure product “goodness” and process “goodness” using a general measurement technique called **Object Measurement**. (The **Object Measurement** logo shown in this figure is a registered trademark owned by Scott E. Donaldson and Stanley G. Siegel.)

How far you have gone and what may not be operating properly. So, too, do measurements that are integrated with your business process help you determine such things as how far you have come in improving your business way and what may need to be fixed.

In addition to using **OM** to quantify software product “goodness” and software process “goodness,” we have used this measurement technique to quantify abstract entities such as strategic information management. In Appendix A, we indicate how **OM** can be used to quantify strategic information management and why quantification of this object is of interest.

Even though **OM** can measure almost anything, we need to stress that the technique is **not** a measurement “silver bullet.” If you have a measurement program that is already helping you effect continual process improvement, you may find that **OM** can complement your measurement program and make it even more robust. If you are new to the software measurement game, you may find that **OM** can help you overcome the blank-page syndrome in firing up a measurement program that makes sense for your organization.
Chapter 6 • Measurement

This chapter also includes a section on other process-related measurements in addition to product and process integrity. The purpose of that section is to illustrate how more conventional types of measurements can be used either in conjunction with measurements obtained from applying OM or instead of measurements obtained from applying this technique.

In the remainder of this section, we set context for the subject of software process and product measurement. We first discuss whether software process improvement may even be applicable to your organization. We then briefly review some measurement fundamentals.

The primary purpose of measurement is to bring about product and process improvement so that the customer is satisfied with the seller’s products. This purpose assumes that the product and process are worth improving. When an organization seeks to achieve orders of magnitude improvement, Business Process Reengineering (BPR) technology is often considered. Typically, a business seeks to restructure its processes when it is losing money, or worse, threatened with going out of business.

As illustrated in Figure 6–4, in many businesses, the software systems development process is part of a much larger business process. It is often not clear whether the overarching business process may prevent meaningful software systems development process improvement. If such is the case, then no amount of tinkering with the software systems development process will be useful until the larger business process is first improved—or, in the extreme, reengineered.

The Catch-22, then, is the following:

How do you know whether the software systems development process needs to be improved if you don’t know whether the larger business process is the real impediment to software systems development success in your business?

At the risk of oversimplification, this question translates into the following process improvement/reengineering analogy:

A process that uses a hammer to drive screws cannot generally be improved by redesigning the hammer; the process needs to be reengineered by replacing the hammer with a screwdriver.

Certain techniques grounded in common sense should bring about software process improvement. If it turns out that applying these techniques does not bring about improvement, then the problems lie elsewhere in the business. In such instances, BPR may need to be invoked in a context far larger than your software systems development process. Thus, if by applying the techniques

we present in this chapter you do not realize process improvement, you may need to look upward within your business to get at the real source of the problem standing in the way of successfully producing software systems. For example, your software systems development process may indeed be completing the development of software systems on time and within budget. However, these systems may not get into the hands of the customers until...
much later because of convoluted business processes associated with miles of paper pushing. Clearly, in such circumstances, no amount of tinkering with the software systems development process is going to solve the overarching business process problem of on-time delivery of the systems to the customers.

One additional observation is in order here regarding BPR versus software development process improvement. Improvement begins with a definition of the software systems development process. This definition provides the overall context for more detailed processes. From the BPR perspective, if a business has no defined and documented software systems development process, then the definition of such a process and its implementation constitute a form of BPR. Putting the software systems development process in place is the first step in bringing order to a presumably ad hoc or chaotic situation. Once some order has been established, it then makes sense to begin thinking about improving what has been put in place.

This chapter deals with the concept of measurement as it applies to software products and the process used to develop these products. Frequently, when the software engineering literature addresses measurement, it uses the term “metric.” IEEE Standard 610.12-1990 defines metric as follows:

*A quantitative measure of the degree to which a system, component, or process possesses a given attribute.*

In truth, the term “metric” is used in various ways. For example, Baumert and McWhinney, in the Software Engineering Institute Technical Report CMU/SEI-92-TR-25, “Software Measures and the Capability Maturity Model” (September 1992), offer the following definitions for the related terms “measure,” “measurement,” and “metric” (p. B-2):

**Measure n.**—A standard or unit of measurement; the extent, dimensions, capacity, etc. of anything, especially as determined by a standard; an act or process of measuring; a result of measurement. v. To ascertain the quantity, mass, extent, or degree of something in terms of a standard unit or fixed amount, usually by means of an instrument or process; to compute the size of something from dimensional measurements; to estimate the extent, strength, worth, or character of something; to take measurements.

**Measurement**—The act or process of measuring something. Also a result, such as a figure expressing the extent or value that is obtained by measuring.

**Metric**—In this document, metric is used as a synonym for measure.

To improve a software product or a process that produces the product, measurement is needed. Figure 6-5 presents our concept of product and process metrics. As stated in the figure caption, we use *metric* to mean “(1) a standard or unit of measurement, or formula used to quantify something, and/or
The term “metric” is used in a variety of ways in the software engineering literature. We use metric to mean “(1) a standard or unit of measurement, or formula used to quantify something and/or (2) the values that the standard or formula may assume.”

(2) the values that the standard or formula may assume.” For example, in the nonsoftware world, “foot” is a standard of measurement used to quantify the length of something. The formula “area = length × width” is used to quantify the region that a rectangle of a specified length and width occupies. The number calculated when an actual length and width are substituted into the formula is also a metric.

Also note that the value scales are generally different for product and process, but both scales range from a minimum value to a maximum value. In this chapter, we look at the analogues to length, width, and area for software development processes and resultant products. The challenge is to establish units of measurements (or, equivalently, value scales) and a relatively painless way to make measurements based on these value scales. In addition, your measurements need to have benchmarks. As shown in Figure 6–6, everyday measurements, such as a person’s weight, the time to run a certain distance, and the number of calories a person needs to consume daily, have meaning only when they can be related to certain standards, or benchmarks, for those measurements.

For example, running a mile in less than four minutes is considered to be “fast” even for the most highly trained runners. The 4-minute-mile benchmark has been established through many measurements made over many years during athletic events. This cumulative measurement experience gives meaning to the number “4-minute-mile” for people who are familiar with track and field events. Likewise, if process and product measurements are to be meaningful, benchmarks need to be established. Here, meaningful measure-
Chapter 6 • Measurement

Figure 6–6 To be meaningful, measurements must have benchmarks. Benchmarks need to be established for software products and software development process measurements, and the relationship between the product and process measurements.
ments means “the measurements can be used to determine whether and where the product or process needs to be improved.” For example, to determine that a project’s development process is “good,” it is necessary to determine (1) whether “good” products are being produced and (2) whether the project’s process “conforms” to the organization’s development process, as defined in the organization’s application development process environment (ADPE).

As we explain in this chapter, if a project is not conforming to the organization’s process but is producing “good” products, then the organization may need to (1) reconsider the development process definition, (2) work with the project to conform to the defined, organizational process so that consistent practices across projects can be achieved, or (3) reconsider the organizational questions being answered by the metrics. Figure 6–7 illustrates the point that measurement is tied directly to questions that are important to the organization.

Customers want products that do what they are supposed to do. Customers also want to have the products delivered on time and within budget. As shown in Figure 6–7, software systems development measurement should address fundamental questions such as the following:

♦ Am I producing “good” products? The name of the game is to produce “good” products that satisfy the customer. Whether or not you are in business to make a profit, your customer needs to be satisfied with your products. The measurement challenge is to determine what a “good” product means. Consequently, as shown in the upper panel of the figure, the “goodness” values are established from the customer’s viewpoint.

♦ Is my process consistently producing “good” products within budget? Whether or not you are in business to make a profit, your process needs to consistently produce “good” products. If you are in a profit-making situation, then your process should enable you to make your profit. If you are not in a profit-making situation, then your process should enable you to meet your budget. In either situation, the measurement challenge is to determine what a “good” process means to your organization. Consequently, as shown in the lower panel of the figure, the “goodness” values are established from the seller’s viewpoint.

In this context, software process improvement becomes an exercise in evaluating product “goodness” and process “goodness.” As Figure 6–7 illustrates, a product is “good” if it does what it is supposed to do and is delivered on time and within budget—so that the customer is satisfied. If the product is not “good,” then the product, the process that produced the product, or both need improvement. A process is “good” if it consistently yields good products such that the seller can make a profit or stay within a budget. If the process is not “good,” then the process needs improvement. Measurement needs to (1) be expressed in everyday terms so that the results make sense to the organization and (2) integrate seller and customer perspectives.
The plan for the rest of this chapter is the following:

♦ In Section 6.2—Measurement Key Ideas, we present the key ideas that you can expect to extract from this chapter.

♦ In Section 6.3—Product Integrity, we show you how to quantify software product attributes to help determine whether a customer is satisfied with seller results. Our intent is to explain to you one approach for assessing customer satisfaction in terms of an index that assigns a value to the integrity (i.e., completeness) of each product that comes out of the software systems development process.
In Section 6.4—Process Integrity, we show you how to measure the activities that make up your software systems development process to determine the correlation between these activities and producing products with integrity. This correlation provides insight into the extent to which these activities are, or are not, contributing to “good” products. Those activities not contributing are candidates for modification or elimination. These modifications and/or eliminations define what “process improvement” means. The discussion in this section is tied to the software systems development process described in Chapter 3. The purpose of this tie is to show you in specific terms how to measure the software development process in terms of its process components and activities. However, the measurement approach is general and can be applied to your development process.

In Section 6.5—Capability Maturity Model for Software (CMM), we describe how the product integrity and process integrity concepts can be applied to the Software Engineering Institute’s (SEI) widely known framework for improving software development, the CMM for Software.

In Section 6.6—Other Process-Related Measurements, we give you ideas for defining process-related metrics, other than the product and process integrity indexes. Our objective is not to be comprehensive, but rather to be suggestive of supplementary ways that you can attack software measurement.

In Section 6.7—Measurement Summary, we summarize the key points developed in the chapter. We include an annotated outline of an ADPE guideline to help you define an organizational approach for product and process measurement. As explained in the chapter, our approach to measurement is general in that it is independent of development technologies and tools.

In Appendix A, we indicate (1) how Object Measurement can be used to quantify strategic information management and (2) why quantification of this object is of interest.

**6.2 Measurement Key Ideas**

Figure 6–8 lists the key ideas that you can expect to extract from this chapter. To introduce you to this chapter, we briefly explain these key ideas. Their full intent will become apparent as you go through this chapter.

1. **Measurements need to be expressed in everyday terms that are familiar to the organization; otherwise, they may, at best, be of little use.**

Simply stated, if the people doing the day-to-day software systems development work do not understand the measurements, the collected measurement data may be counterproductive to your improvement activities. This chapter offers you an approach for defining meaningful measurements for your organization.
Measurement Key Ideas

1. Measurements need to be expressed in everyday terms that are familiar to the organization; otherwise, they may, at best, be of little use.

2. Keep the measurement process simple—otherwise, it will die quickly. Simple means “easy-to-collect data and easy-to-interpret information resulting from these data.”

3. Establish benchmarks to give meaning to measurements. Without context, process measurement is a waste of time.

4. Measure product integrity by (1) selecting product attributes to measure, (2) defining value scales for each product attribute, (3) recording observed attribute values, and (4) combining the recorded attribute values into a single number called a product integrity index.

5. Measure process integrity by (1) selecting the software development process components to measure, (2) selecting component activities to measure, (3) defining value scales for each component activity, (4) recording observed activity values, and (5) combining recorded activity values into a single number called a process integrity index.

6. Customer satisfaction is the ultimate measure of software systems development process value. If the process fails to yield products satisfying the customer, the process needs repair.

7. Measure customer satisfaction by incorporating customer feedback on delivered products into the process.

8. Measurements should be a part of the software systems development process.

9. Document in an ADPE element the measurement process and the items to be measured.

Figure 6–8 Successful software systems development is a continual improvement exercise. Measurement is a means for effecting this improvement. Here are key measurement concepts that are explained in this chapter.
3. Establish benchmarks to give meaning to measurements. Without context, process measurement is a waste of time.

This chapter offers you ideas for establishing a framework for interpreting the measurements you make and collect. Many of us at one time or another have been concerned about our weight. It is easy to measure our weight. However, the resultant measurement is generally of little value if, for example, our objective is to gain or lose weight. We need weight benchmarks to know whether we are underweight, okay, or overweight. Similarly, we need process benchmarks that can tell whether the process that we have measured is underweight, okay, or overweight with respect to, say, the integrity of delivered products that the process yields. This chapter offers you ideas for constructing such benchmarks.

4. Measure product integrity by (1) selecting product attributes to measure, (2) defining value scales for each product attribute, (3) recording observed attribute values, and (4) combining the recorded attribute values into a single number called a product integrity index.

Many of our conventional measures, such as the “foot,” have their origin in objects that most people could recognize. A challenge in the software process measurement game is to find analogues to such easily recognized units of measure. This chapter offers you ideas for such analogues. This chapter also offers you ideas for converting the multidimensional product integrity concept into a one-dimensional index. These ideas will, at the same time, give you insight into how you can measure individual product integrity attributes or combinations of these attributes—whatever attributes you may choose to quantify product “goodness.”

5. Measure process integrity by (1) selecting the software development process components to measure, (2) selecting component activities to measure, (3) defining value scales for each component activity, (4) recording observed activity values, and (5) combining recorded activity values into a single number called a process integrity index.

This chapter offers you ideas for converting the multidimensional process integrity concept into a one-dimensional index, and ideas about how this index is related to the product integrity index. We explain how process integrity is a generalization of the product integrity concept.

6. Customer satisfaction is the ultimate measure of software systems development process value. If the process fails to yield products satisfying the customer, the process needs repair.

This chapter offers you ideas for measuring customer satisfaction and linking this measure to process activities. Through this linkage, we offer you ideas for modifying process activities to increase customer satisfaction.
7. Measure customer satisfaction by incorporating customer feedback on delivered products into the process.

How can you get insight into what the customer thinks your software systems development process is delivering? This chapter offers you ideas for integrating within the process customer feedback on delivered products. We offer suggestions on how to measure this feedback in terms that can be linked to process activities.

8. Measurements should be a part of the software systems development process.

Defining, collecting, using, and communicating measurement data should be integrated into the development process and used, in part, to improve the organization’s products and processes. However, measurement for measurement’s sake is a waste of time and resources. This chapter presents ideas on how to establish measurements that can be integrated into your organization’s software development activities.

9. Document in an ADPE element the measurement process and the items to be measured.

A measurement process is an organized way of effecting software systems development process improvement. This chapter gives you ideas for documenting the measurement process, thereby helping you organize your approach to software systems development process improvement.

6.3 Product Integrity

Like other organizations, you want your organization to stay in business. It is axiomatic that “staying in business” is strongly tied to customer satisfaction, which can be expressed in many ways. The purpose of this section is to explain how the product integrity concept can be used to quantify “customer satisfaction.”

Our approach in this section is the following:

- We use an example set of attributes to define an example product integrity index.
- We use the example index to generate values for several different products to show you how to do product “goodness” measurement using the index.
- We then give you a general formula for computing the index.
- The worked-out examples and the general formula enable you to apply straightforwardly our product integrity measurement approach to your environment.
Figure 6–9 depicts the example set of attributes. We have chosen the five attributes shown because they are often of interest to management and product developers. These attributes are defined more specifically as follows:

**at₁** Fulfills specified customer needs (i.e., does what it is supposed to do as recorded and agreed to by the customer and the seller).

**at₂** Can be easily and completely traced through its life cycle (i.e., is “maintainable”—it can be easily updated to (1) incorporate new things, (2) revise existing things, and (3) get rid of things no longer deemed needed by the customer).

**at₃** Meets specified performance criteria (e.g., How many? How often? How long?; these criteria are sometimes considered special cases of customer needs—the first product integrity attribute).

**at₄** Meets cost expectations (i.e., costs what the customer and the seller agreed that it should cost as expressed in a project plan or updates to the plan).

---

**Figure 6–9** Here is an example of a way to define product integrity in terms of attributes that are often of interest to both management and product developers.
Meets delivery expectations (i.e., is delivered in accordance with schedules agreed to by the customer and the seller in a project plan or updates to the plan).

Product integrity is thus a multidimensional concept that associates attributes with a product. To use product integrity to quantify customer satisfaction, we need a convenient way to quantify something with multiple dimensions (here, something with five dimensions). The discussion that follows offers an approach that can be used to quantify any multidimensional entity. This discussion also makes it evident how any subset of the five product attributes we discuss, or any set of attributes you want to use, can be used to measure customer satisfaction. The following treatment thus provides a general approach to using product integrity as a basis for measuring customer satisfaction. Through experimentation with this general approach, you can define a preferred approach to apply in your environment.

The mathematical and scientific disciplines often handle multidimensional quantities with entities known as “vectors.” The scientific discipline of physics, for example, uses vectors to describe many quantities (displacement, velocity, acceleration, force, momentum—to name a few). To illustrate from this list, the change of position of a particle is called a “displacement.” When we go to work in the morning, we displace ourselves from our home to our place of work. We can represent this displacement as an arrow on a map drawn from the place on the map that is our home to a place on the map where our office is. This arrow represents the (straight-line) distance from our home to our office and the direction of this distance with respect to, say, some reference frame, such as that used to define the four compass points. Figure 6–10 shows the concept of displacement in one, two, three, and \( n \) dimensions.

Figure 6–10 also shows how the length of the vector is calculated to determine the magnitude of the displacement. For example, we represent displacements in three-dimensional space by specifying a triple of numbers \((x_1, x_2, x_3)\), which defines the displacement of a point with respect to three mutually perpendicular axes. These axes establish a scale of values in this space.

We use this notion of displacement in space to derive the idea of a product integrity index. The space of interest is product attribute space. That is, the axes in this space, which we also refer to as product integrity space, are product attributes. Figure 6–11 illustrates a three-dimensional product integrity space, where the attribute axes are the quantities \( a_{t_1}, a_{t_4}, \) and \( a_{t_5} \) defined earlier.

By extension, then, if we want to quantify product integrity as it is defined by the example set of five attributes introduced earlier, we can think of product integrity as an entity in five-dimensional space. One axis in this space shows how much a product “fulfills customer needs”; a second axis shows how the evolution of the product “can be easily and completely traced through its life cycle”; and so forth for the other product integrity attributes (unfortunately,
Chapter 6 • Measurement

How long is a line?

Since we live in a three-dimensional world, we cannot draw the five-dimensional extension to Figure 6–11.

To understand how we can use these vector-related ideas for quantifying the concept of product integrity as a means for measuring customer satisfaction, consider the following five-dimensional vector:

---

**Figure 6–10** The idea for a product integrity index derives from the concept of the length of a line in space. The figure shows how the length of a line can be portrayed in spaces of various dimensions as the magnitude of a vector representing a displacement. The tail of the vector represents the starting point, and the head of the vector represents the destination point. The length of the vector represents the distance between the starting point and the destination point. Similarly, the product integrity index is simply the length of a line in product attribute space.

---

since we live in a three-dimensional world, we cannot draw the five-dimensional extension to Figure 6–11).

To understand how we can use these vector-related ideas for quantifying the concept of product integrity as a means for measuring customer satisfaction, consider the following five-dimensional vector:
A Three-Dimensional Product Integrity Vector

Product Goodness

$PI = \text{Product Integrity Vector}$

- $a_1$: fulfills customer needs
- $a_2$: meets delivery expectations
- $a_3$: meets cost expectations
- $a_4$: Traceable
- $a_5$: PerfCrit
- $a_6$: WithinBudget
- $a_7$: OnTime

Figure 6-11 Product integrity is a multidimensional concept associating a number of attributes with a product. A vector is one way simply to represent a multidimensional concept. The figure shows a three-dimensional product attribute space made up of three members from the example set of five attributes introduced earlier. A vector in this space is the product integrity vector. Its length is what we will use to measure product "goodness." Our approach to measuring product "goodness" is thus an exercise in measuring the length of the product integrity vector.

$$PI = \frac{\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}}{N} = \frac{\begin{bmatrix} \text{CustNeeds} \\ \text{Traceable} \\ \text{PerfCrit} \\ \text{WithinBudget} \\ \text{OnTime} \end{bmatrix}}{N}$$

(6.3-1)
In Equation 6.3–1, $\mathbf{PI}$ is a vector in five-dimensional product integrity space whose components, $a_t$, are the example product integrity attributes defined earlier. The quantity $N$ is a normalization factor that establishes a “product goodness scale.” As we subsequently explain, we choose $N$ so that the length of $\mathbf{PI}$ is restricted to the range from zero to one.

Now, to turn Equation 6.3–1 into a measurement tool, we consider the following questions:

♦ How can we convert a five-dimensional quantity into a single quantity to simplify measurement interpretation?
♦ What scales do we establish for the attributes?
♦ What relative weights do we assign to the attributes?
♦ How can we assign a scale of values for the single quantity?

Clearly, there are many sensible ways to address these questions.

The first question deals with simplifying measurement. As Equation 6.3–1 indicates, multidimensional expressions of product integrity are possible. However, for simplicity, we have chosen to restrict ourselves to a one-dimensional quantity to express product integrity quantitatively. Recalling Figure 6–11, that quantity is the length of the product integrity vector $\mathbf{PI}$ (i.e., it is the five-dimensional extension to the three-dimensional case shown in the figure).

As Figure 6–11 indicates, each product attribute dimension contributes to the “length” of the vector $\mathbf{PI}$. To convert the five-dimensional quantity in Equation 6.3–1 into a single quantity (to represent “quality” or “completeness”), we calculate the “length” of the vector. We call the length of $\mathbf{PI}$ the Product Integrity Index, or $\mathbf{Plindex}$. As subsequently explained, this product integrity vector length, $\mathbf{Plindex}$, is simply the square root of the sum of the weighted ($w_i$) squares of the attributes $a_t$ divided by the normalization factor $N$.

The second question deals with attribute scales. Many people find it useful and convenient to quantify things in terms of percentages. Thus, a convenient range for an attribute scale goes from zero to one. Again, for simplicity, we take the approach of limiting the attribute scales to the range zero to one.

The third question deals with relative weights for product attributes. If we assign the same scale to each attribute (namely, zero to one), we are weighting each attribute equally. For simplicity, we will take this approach. However, you may wish to emphasize one attribute more than the others. For example, if you wanted to give “meets delivery expectations” double the importance of any of the other attributes, you could set its scale to run from zero to two and set the scales of the other attributes to run from zero to one. Equivalently, you

\footnote{Note that we are mapping our attribute values to dimensionless scales. This mapping allows us to combine the attribute values into a single quantity.}
can keep all the scales the same and give prominence to selected attributes through the use of weighting factors \((w_i)\). We show you how to introduce such weighting factors.

The fourth question deals with establishing a value scale for the length of PI. We select a scale for the magnitude of this vector by choosing a value for the normalization factor \(N\). Arguing as we did before, we simply select a scale that ranges from zero to one. For equally weighted attributes, the value of \(N\) then becomes the square root of the sum of the squares of the maximum values that the attributes \(a_{t_i}\) can take on. For the case in which a product has five attributes each with a maximum value of one, the value of \(N\) thus becomes the square root of 5. We also show you how to compute \(N\) if the attributes are not equally weighted.

On the basis of the preceding discussion of one way to address the four questions previously introduced, we can now define a product integrity index, \(PI_{\text{index}}\), that ranges from zero to one as follows:

\[
PI_{\text{index}} = \sqrt[\frac{\sum_{i=1}^{n} w_i^2 a_{t_i}^2}{\sum_{i=1}^{n} w_i^2 (\text{maximum}[a_{t_i}])^2}}
\]

where \(a_{t_i}\) = product integrity attribute
\(n\) = number of product integrity attributes
\(w_i\) = weighting factor for attribute \(a_{t_i}\)
maximum \([a_{t_i}]\) = maximum value of \(a_{t_i}\).

Figure 6–12 presents three examples of how Equation 6.3–2 can be used. Example 1 represents our software product that is characterized by five attributes. Example 2 represents the case in which the attribute, \(a_{t_1}\)—fulfilling customer requirements, is considered twice as important as the other attributes. Example 3 represents the case in which attributes \(a_{t_2}\) and \(a_{t_3}\) are suppressed.

The product integrity index, \(PI_{\text{index}}\), is normalized to one (i.e., restricted to the range of zero to one). If you want to remove this normalization, then remove the denominator.

To illustrate how Equation 6.3–2 works, we need to define value scales for each of our example software product attributes \(a_{t_i}\). There is a multiplicity of ways such assignments can be made. Figure 6–13 shows one way to set up value scales for these attributes.

This example is explained below, and it provides insight into ways that you can make such assignments that are relevant to your organization.
For $a_{t_1}$ (fulfills specified customer needs), we set up a three-value scale based on an acceptance of deliverable form as follows:

- $a_{t_1} = 1$ if the customer returns the form indicating “accepted as delivered.”
- $a_{t_1} = 0.5$ if the customer returns the form indicating “accepted with minor changes.”
- $a_{t_1} = 0$ if the customer returns the form indicating “changes to be negotiated.”

If we wanted to provide more insight into the percentage of requirements fulfilled, we could count such requirements appearing in the product and compare them against some ground truth showing what this number of requirements should be (“shall” in the language of requirements analysis). For example, suppose the product were a requirements specification, and suppose CCB minutes indicated that 40 requirements should be addressed.

---

Example 1—Equal weighting factors

$n = 5$, all $w_i = 1$, all maximum $a_{t_i} = 1$

$$P_{index} = \sqrt[5]{\frac{\sum_{i=1}^{5} a_{t_i}}{5}}$$

Example 2—One attribute is twice as important as any other attribute

$n = 5$, all $w_i = 1$ except for $w_1 = 2$ (fulfilling customer requirements twice as important as any other attribute), all maximum $a_{t_i} = 1$

$$P_{index} = \sqrt[8]{\frac{4a_{t_1} + \sum_{i=1}^{3} a_{t_i}}{8}}$$

Example 3—One or more attributes are suppressed

$n = 5$, all $w_i = 1$ except for $w_2 = w_3 = 0$ (i.e., the traceability and performance requirements attributes are excluded), all maximum $a_{t_i} = 1$

$$P_{index} = \sqrt[3]{\frac{\sum_{i=1,3,5} a_{t_i}}{3}}$$

Figure 6–12 This figure illustrates three ways in which the general formula for the product integrity index, $P_{index}$, can be used.

---

We explained in Chapter 3 that, as part of our software systems development process, we use an acceptance of deliverable form to obtain, in part, customer feedback. For this example, we have assigned discrete values for the three possible customer evaluations.
### Example Product Integrity Attribute Value Scales

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fulfils specified customer needs (as recorded and agreed to)</td>
<td>1.0: Accepted as delivered; 0.5: Accepted with minor changes; 0.0: Changes to be negotiated</td>
</tr>
<tr>
<td>Can be easily and completely traced through its life cycle (maintainable)</td>
<td>1.0: Detailed written records throughout the life cycle; 0.5: Written records for some portion of life cycle; 0.0: Only customer’s statement of work exists</td>
</tr>
<tr>
<td>Meets cost expectations (within costs or updated costs)</td>
<td>1.0: Delivered for less than cost estimate; 0.9: Delivered for cost estimate; 0.0: Delivered for more than the cost estimate</td>
</tr>
<tr>
<td>Meets delivery expectations (within schedule, as planned or updated)</td>
<td>1.0: Delivered ahead of schedule; 0.9: Delivered within 10% of schedule; 0.0: Delivered late</td>
</tr>
</tbody>
</table>

#### Figure 6-13
This figure illustrates value scales for each of the five example product integrity attributes \( (a_t) \) discussed. You will want to set up attributes and value scales that make sense for your organization.
For \( at_2 \) (can be easily and completely traced through its life cycle), the situation can become complicated. Depending on the product, traceability may involve more than the product itself. For example, if the product is computer code, then traceability involves the existence of predecessor products such as design and requirements specifications. If the product is a requirements specification, then traceability typically involves documents that a customer may supply, such as congressional legislation or corporate policies. More generally, traceability involves such things as product decisions recorded at CCB meetings, internal project meetings, and recorded seller and customer management conversations and e-mail between these two parties. To keep things simple, we set up a crude three-value scale based on the existence of records showing how the product evolved, as follows:

- \( at_2 = 0 \) if nothing other than a customer-prepared statement of work (SOW) exists calling for the development of the product.
- \( at_2 = 0.5 \) if written records exist for some part of the project’s life cycle showing how the product contents are what they are.
- \( at_2 = 1 \) if detailed written records exist throughout the life of the project showing how the product contents are what they are.

For \( at_3 \) (meets specified performance criteria), we simply set \( at_3 = at_1 \), since performance criteria are often lumped with customer needs (if such is not the case in your environment, then you can follow the suggestions previously offered for the attribute \( at_1 \)). That is, we use the following scale:

- \( at_3 = 1 \) if the customer returns the form indicating “accepted as delivered.”
- \( at_3 = 0.5 \) if the customer returns the form indicating “accepted with minor changes.”
- \( at_3 = 0 \) if the customer returns the form indicating “changes to be negotiated.”

For \( at_4 \) (meets cost expectations), we set up a three-value scale as follows:

- \( at_4 = 1 \) if the product was delivered for less than the cost specified in the project plan or as modified in CCB minutes.
- \( at_4 = 0.9 \) if the product was delivered for the cost specified in the project plan or as modified in CCB minutes.
- \( at_4 = 0 \) if the product was delivered for more than the cost specified in the project plan or as modified in CCB minutes.

Clearly, this scale places a slight premium on delivering for less than planned cost. The scale also ranks a deliverable delivered for $1 more than planned cost the same as a deliverable delivered for $3,000 more than planned cost. Again, in your environment, you may not wish to place a premium on delivery below cost—but the preceding example gives the
idea for how you can establish such premiums (this remark applies also to the attribute $at_5$ [meets delivery expectations] in the following).

* For $at_5$ (delivered on time), we set up a scale as follows:
  * $at_5 = 1$ if the product was delivered before the delivery date specified in the project plan or before the delivery date as modified in CCB minutes.
  * $at_5 = 0.9$ if the product was delivered with no more than a 10 percent schedule slippage. Here, “percent slippage” is calculated by taking the length of time allocated in the project plan for preparing the product or as modified in CCB minutes and dividing that time into the slippage time and multiplying by 100. For example, if the product was scheduled to be delivered 10 weeks after project start, but it was actually delivered 11 weeks after project start, then $at_5 = 0.9$ because the slippage was $(1/10) \times 100 = 10$ percent.
  * $at_5 = (1 - X)$, where $X$ is the fraction of schedule slippage as just calculated. For example, if the product was scheduled to be delivered 10 weeks after project start, but it was actually delivered 13 weeks after project start, then $at_5 = (1 - [3/10]) = 0.7$. For all schedule slippages greater than or equal to the original length of time to produce the deliverable, $at_5 = 0$ (for example, if a deliverable was to be developed over a 10-week period, any delays greater than or equal to 10 weeks result in $at_5 = 0$).

This scale places a slight premium on delivering early. Also, it favors on-time product delivery while allowing for some planning leeway.

We now illustrate how to calculate $Plindex$ in Equation 6.3–2 using the preceding scales for the following example products: (1) a requirements specification, (2) a new release of a legacy software system, (3) an updated user’s manual, and (4) a new project plan.

**Example 1—$Plindex$ Calculation for a Requirements Specification**

The product is a requirements specification. After delivery, the customer sent back the acceptance of deliverable form showing “accepted with minor changes.” Thus, $at_3 = at_1 = 0.5$. The product was delivered on time so $at_5 = 0.9$. The project plan called for 300 hours to be expended on the task to produce the document, but only 275 hours were expended. Thus, $at_4 = 1$. Written records consisting of CCB minutes showing decisions underlying the document’s content exist for some part of the project. Thus, $at_2 = 0.5$. The product integrity index, $Plindex$, for this requirements specification is therefore the following:

$$Plindex = \sqrt{0.5^2 + 0.5^2 + 0.5^2 + 1^2 + 0.9^2} = 0.72.$$  

Figure 6–14 shows how this requirements specification example can be graphically presented.
### Product Integrity Index

\[
\text{Plindex} = \sqrt{\frac{\sum_{i=1}^{5} a_{t_i}^2}{5}} = \sqrt{\frac{0.5^2 + 0.5^2 + 0.5^2 + 1^2 + 0.9^2}{5}} = 0.72
\]

**Quantified Product Integrity Attributes**

- Fulfills specified customer needs \(a_{t_1}\)
- Accepted with minor changes \(a_{t_1} = 0.5\)
- Delivered within 10% of schedule \(a_{t_1} = 0.9\)
- Can be easily & completely traced through its life cycle \(a_{t_2} = 0.5\)
- Written records for some portion of life cycle \(a_{t_2} = 0.5\)
- Meets delivery expectations \(a_{t_2} = 0.5\)
- Delivered for less than cost estimate \(a_{t_4} = 1\)
- Meets specified performance criteria \(a_{t_3} = a_{t_1}\)
- Accepted with minor changes \(a_{t_3} = 0.5\)

**Figure 6–14** This figure illustrates one way to display the results of quantifying the integrity of a software product (e.g., a requirements specification). For the attribute values shown, \(\text{Plindex} = 0.72\).
The top panel displays how PIindex is calculated given the recorded data, and summarizes the product integrity attributes. The bottom panel provides additional detail into the attributes, their scales, and the recorded data. The bottom panel displays the observed attribute values on a Kiviat-like diagram. This diagram gives the next level of insight into the nature of the product being measured. In particular, it shows the recorded value of each product integrity attribute plotted on the scale for that attribute.6

Example 2—Plindex Calculation for a New Release of a Legacy Software System

The product is a new release of a legacy software system. After delivery (which was preceded by acceptance testing), the customer sent back the acceptance of deliverable form showing “accepted as delivered.” Thus, \( a_{t1} = at_5 = 1 \). The product was supposed to be delivered 20 weeks after project start, but was delivered 5 weeks late so \( at_5 = (1 - [5/20]) = 0.75 \). The project plan called for 3000 hours to be expended on the task to produce the system upgrade, but only 2900 hours were expended. Thus, \( at_4 = 1 \). No requirements or design specifications exist. However, each code module has a header containing key information about the module’s contents and a version number. In addition, written records consisting of CCB minutes showing decisions underlying the code module changes exist throughout the acceptance testing cycle. Thus, \( at_2 = 0.5 \). The product integrity index, \( \text{Plindex} \), for this new release of the legacy system is therefore the following:

\[
\text{Plindex} = \sqrt{1^2 + 0.5^2 + 1^2 + 1^2 + 0.75^2} \div \sqrt{5} = 0.87.
\]

6It is important to note that, in general, when you are dealing with unequally weighted attributes (which is not the case with the example shown in Figure 6–14), this situation can affect the way you display your measurements. When you have unequally weighted attributes (i.e., all \( w_i \) are not equal), there are several possible ways of using the Kiviat-like diagram to display what is going on. Some of these ways are the following:

- You can plot unequally weighted observed values. In this case, the length of a value scale in your display ranges from the weight \( (w_i) \) times the minimum attribute value (minimum \( [a_{ti}] \)) to the weight \( (w_i) \) times the maximum attribute value (maximum \( [a_{ti}] \)). For example, suppose that attribute \( a_{t1} \) in Figure 6–14 had twice the weight as any of the other attributes (i.e., \( w_1 = 2 \) and all the other \( w_i = 1 \)). In this case, the length of the \( a_{t1} \) value scale would run from a minimum value of zero (i.e., \( [w_1] \) times \( \text{minimum} [a_{t1}] = 2 \times 0 \)) to a maximum value of two (i.e., \( [w_1] \) times \( \text{maximum} [a_{t1}] = 2 \times 1 \)), while the other attribute value scales would run from zero (i.e., \( [w_i] \) times \( \text{minimum} [a_{ti}] = 1 \times 0 \)) to one (i.e., \( [w_i] \) times \( \text{maximum} [a_{ti}] = 1 \times 1 \)), for \( i = 2, 3, 4, 5 \).

- You can plot equally weighted observed values. Each scale in this case would run from zero to one (including the scale for \( a_{t1} \)). To show that \( a_{t1} \) has twice the weight as any of the other attributes, you could annotate the Kiviat-like diagram with a statement to this effect.

- You can plot unequally weighted and equally weighted observed values.

The bottom line here is to set up the display (Kiviat-like or otherwise) in a way that makes sense for your organization.
Chapter 6 • Measurement

Example 3—**Plindex Calculation for an Updated User’s Manual**

The product is an update to a user’s manual for a new release of a software system that the seller maintains. The customer was uncertain about many things that the manual should contain and constantly wanted to change its contents (even up to the last minute). After delivery, the customer sent back the acceptance of deliverable form showing “changes to be negotiated.” Thus, $a_{t_3} = a_{t_1} = 0$. Because of the customer uncertainty and the many changes to the document, the manual, which was supposed to be delivered 10 weeks after project start, was delivered 5 weeks late, so $a_{t_5} = (1 - [5/10]) = 0.50$. The project plan called for 300 hours to be expended on the task to produce the user’s manual, but, because of the numerous changes and schedule slippages, 360 hours were expended. Thus, $a_{t_4} = 0$. Written records consisting of CCB minutes showing the change track record of the document exist. These records also indicate that the customer was alerted to potential schedule slippages and cost overruns because of the document’s unsteady state. Thus, $a_{t_2} = 0.5$. The product integrity index, $Plindex$, for this user’s manual update is therefore the following:

$$Plindex = \frac{\sqrt{0^2 + 0.5^2 + 0^2 + 0^2 + 0.5^2}}{\sqrt{5}} = 0.32.$$  

Example 4—**Plindex Calculation for a New Project Plan**

The product is a new project plan for the development of a software system. The process engineering group, which is responsible for project planning, estimated that the plan, with one revision, would cost $3000 and would take 20 working days to deliver to the customer. The plan was actually delivered 24 working days after start, so that $a_{t_5} = (1 - [4/20]) = 0.80$. The actual cost to produce the plan was $2700, so that $a_{t_4} = 1$. Besides the SOW, the customer supplied needed reference material that was (1) referenced in the plan, (2) used to construct a current system concept, (3) used to construct a system concept after plan accomplishment, and (4) used to construct the technical approach. Thus, $a_{t_2} = 1$. The plan resulted in a contract, which implies that the plan fully responded to the customer requirements stipulated in the SOW. In addition, the contract implies customer acceptance of the project plan and is therefore equivalent to the customer sending back the acceptance of deliverable form showing “accepted as delivered.” Thus, $a_{t_3} = a_{t_1} = 1$. The product integrity index, $Plindex$, for this project plan is therefore the following:

$$Plindex = \frac{\sqrt{1^2 + 1^2 + 1^2 + 1^2 + 0.8^2}}{\sqrt{5}} = 0.96.$$  

Figure 6–15 summarizes the $Plindex$ calculations for the preceding examples.
Figure 6–15 This figure illustrates Plindex for four software products. Plindex was calculated after the customer received each product and returned the acceptance of deliverable form.
As we stated at the outset of this chapter, it is easy to measure our weight. However, the resultant measurement is generally of little value if, for example, our objective is to gain or lose weight. We need weight benchmarks to know whether we are underweight, okay, or overweight. Similarly, we need benchmarks for Plindex. For example, we can use the product integrity index to establish norms for “product quality” or “completeness.” As you gain experience with this index, you can establish goals for various types of products, projects, and seller periods of performance. For example, you can establish goals such as the following:

- Each release of a legacy system for which little or no documentation exists shall have a product integrity index not less than 0.75.
- Each deliverable for each project whose ultimate objective is to produce a new software system shall have a product integrity index not less than 0.85.

The examples discussed deal with calculating Plindex after a product is delivered to the customer. However, Plindex can also be used to quantify a product’s integrity during its development, as well as after its delivery. As shown in Figure 6–16, to apply Plindex during product development, (1) think of the product development process as building a sequence of interim products leading up to the delivered product (e.g., outline, annotated outline, rough draft, etc.), and (2) measure the integrity of each of these interim products in a way similar to the way that the integrity of the delivered product is measured.

Assessing the integrity of these interim products can help the project manager and the product development staff appropriately focus efforts to increase the integrity of the to-be-delivered product. The figure shows how each of the product integrity attributes can be interpreted for interim products. These interpretations are based on the interpretations given to these attributes for the to-be-delivered product. You can set up a similar correspondence for whatever interpretations you choose to give to your product integrity attributes.

To aid in tracking the evolution of a product, it may be useful to plot the interim Plindexes. Figure 6–17 illustrates this idea for a requirements specification. Plindex and each of the product attributes (at) are plotted. Such juxtaposed plots can help the project manager to ensure that a product is evolving as planned. These plots can also give the customer quantitative visibility into the evolution of the deliverable.

As shown in the top panel of Figure 6–17, there are six reporting periods before the requirements specification is scheduled for delivery to the customer. During the first reporting period, the requirements specification Plindex was reported to be near 0.9. Looking down the period-1 column, you can see the following:
Chapter 6  •  Measurement

Quantified Product Integrity Attributes for Interim Products

- Fulfills specified customer needs (as recorded and agreed to) \((at_i)\)
  - 1.0: Product needs no changes—proceed \((at_i = 1.0)\)
  - 0.5: Product needs minor changes—make them, then proceed \((at_i = 0.5)\)
  - 0.0: Product needs to be reworked—give me new draft before proceeding \((at_i = 0.0)\)

- Can be easily and completely traced through its life cycle (maintainable) \((at_i)\)
  - 1.0: Detailed written records during product’s development \((at_i = 1.0)\)
  - 0.5: Written records for some portion of product’s development \((at_i = 0.5)\)
  - 0.0: Only customer’s statement of work exists \((at_i = 0.0)\)

- Meets specified performance criteria (How many? How often? How long?) \((at_i)\)
  - 1.0: See \(at_i\) \((at_i = 1.0)\)
  - 0.5: See \(at_i\) \((at_i = 0.5)\)
  - 0.0: See \(at_i\) \((at_i = 0.0)\)

- Meets cost expectations (within costs or updated costs) \((at_i)\)
  - 1.0: Produced for less than the cost estimate \((at_i = 1.0)\)
  - 0.9: Produced at estimated cost \((at_i = 0.9)\)
  - 0.0: Produced for more than the cost estimate \((at_i = 0.0)\)

- Meets delivery expectations (within schedule, as planned or updated) \((at_i)\)
  - 1.0: Produced ahead of schedule \((at_i = 1.0)\)
  - 0.9: Produced within 10% of schedule \((at_i = 0.9)\)
  - Produced late \(\left(\frac{at_i = 1 - \left[\frac{\text{# days/weeks late}}{\text{# scheduled days/weeks}}\right]}{at_i = 0.0}\right)\)
  - 0.0: Produced late, ≥ twice the number of scheduled days/weeks \((at_i = 0.0)\)

**Figure 6–16** The product integrity index, \(Plindex\), can be used to quantify a product’s integrity during its development, as well as after its delivery to the customer.
Figure 6–17  This figure illustrates how the product integrity index concept can be used to track the integrity of a product as it evolves from the start of its development to the time it is delivered to the customer.
Chapter 6 • Measurement

The specified customer’s needs were met with an outline that was approved.

Only the customer’s SOW existed at the start of the project, which is what you would expect.

Performance criteria are set equal to customer’s needs.

Cost expectations were exceeded.

Delivery expectations were met.

The project seems to be going well, so the project manager told the team to skip the next reporting period and report progress in period 3. To the project manager’s surprise, when $PI_{index}$ was reported in period 3, the value had fallen to 0.5. Looking down the period-3 column, you can see the following:

The specified customer’s needs were not met with an annotated outline that was not approved.

The approved outline and CCB minutes existed, but other hallway and telephone conversations with the customer were not reflected in the CCB minutes.

Performance criteria are set equal to customer’s needs.

Cost expectations were exceeded.

Delivery expectations were not met.

When $PI_{index}$ dropped, it acted as an indicator that the project was not progressing as desired. Upon inspecting the attribute values, the project manager was able to gain some insight into the situation. The project manager then assembled the appropriate team members for a meeting to discuss the particulars and make decisions about what to do next. The project manager decided to have the team rework the annotated outline and discuss the results with the customer. The interaction with the customer was to be documented so that “what” the customer was saying would (1) not be forgotten and (2) could be incorporated into the outline. The project manager also decided to bring in a more senior person who had specific experience that could help the team work the annotated outline. Finally, the project manager, in concert with the team, decided to submit the reworked outline ahead of schedule, so that if there were any last minute issues, they could be addressed before formal delivery to the customer. As can be seen in the period-4 results, the decisions made resulted in an increase in $PI_{index}$. The story goes on, but the point is that the $PI_{index}$ value, the attributes, the attribute value scales, and the display of the collected information help to focus attention on those areas of interest to
the customer and the seller. Such focus helps to reduce project risk and increase success.

Such juxtaposed plots can also help the project manager’s bosses gain visibility into project progress. Such insight is particularly useful when these bosses have responsibility for multiple projects. By periodically reviewing such plots, managers (and others) help to drive out what really matters to the customer and the organization. Once a project or organization settles on what is important (as reflected in the value scales), then the product integrity index can help the project or organization follow a structured approach to improving the products it develops. We illustrate this fundamental idea as follows for an organization that consists of a number of projects and which produces, say, tens of deliverables a month:

♦ By looking at monthly reports of interim or final \( P_{\text{index}} \) values, the head of the organization (the program manager) can quickly see which of the tens of deliverables (1) to be shipped out or (2) shipped out may be falling short. Those deliverables with values near 1 probably do not need the program manager’s attention. By definition, those deliverables embody what the organization thinks is important. It is only the deliverables with values far away from 1 that need attention. However, how far is “far away” will be determined by the organization as it becomes more comfortable working with \( P_{\text{index}} \). For those deliverables needing attention, the program manager can use Kiviat-like diagrams such as those shown in Figure 6–15 to see quickly why those products fell short. Thus, using \( P_{\text{index}} \) to track product development can help management at all levels further pinpoint product shortfalls—before they become big headaches.

♦ Over time, as the database of \( P_{\text{index}} \) values grows, the program manager and others in the organization can observe trends in the “goodness” of products being delivered. Trend analysis can help pinpoint those areas where the organization can improve—and by how much. This insight can be used to decide on corrective actions. For example, if such trend analysis shows that most products with low \( P_{\text{index}} \) values are that way because they are delivered over budget, then the organization can take steps to improve its resource-estimating procedure with some confidence that this corrective action is the right thing to do. In addition, if the program manager finds that on-time delivery is averaging less than 0.50, this statistic may be a signal that project planning estimating techniques need improvement.

Regardless of which form of the \( P_{\text{index}} \) formula you decide on for your organization, compiling statistics based on \( P_{\text{index}} \) can help you gain insight into areas for product development process improvement.

We summarize our product measurement discussion in Figure 6–18, which lists five steps to follow when setting up and collecting product
1. Decide on the questions that you want and/or need to address (e.g., am I producing “good” products?).
2. Select the products from your software systems development process that you want to measure (e.g., requirements specification).
3. Identify the product attributes that you want to measure (e.g., for a requirements specification, you might identify an attribute as “at 4—meets cost expectations”).
4. For each identified attribute (e.g., at 4), define a value scale in everyday terms that are familiar to the organization (e.g., delivered for more than the cost estimate = 0.0, delivered for cost estimate = 0.9, and delivered for less than cost estimate = 1.0).
5. Using the formulas given in this chapter, calculate the product integrity index value. For simplicity, use the formulas that yield values between zero and one. Select weighting factors to reflect your perception of the relative importance of your product attributes.

Figure 6–18 This high-level procedure helps you through the product measurement steps based on the concepts and examples introduced so far in this chapter.
metrics. Our recommendation is to start simple and try a pilot measurement program.

After you have decided on what questions you want answered and what products you want to measure, you need to decide on the granularity of your product measurements. We recommend that you do not select too many product attributes at first. You do not want the measurement program to take on a life of its own. You need to collect data and determine what attributes are worth measuring. As the number of product attributes increases, each attribute’s contribution to the measurement is reduced accordingly, but you do not want too few, else you may not gain significant insight into the answers to your questions. The key point here is that the steps in Figure 6–18 provide (1) a structured way of quickly focusing in on product weaknesses and (2) a structured way of taking corrective action to correct these weaknesses.

We want to make one final point in this section. How do you know when you have improved the software systems development process? As shown in Figure 6–19, one way is to measure the average value of the Plindex for products over a period of time. On the basis of the analysis of the Plindex attributes, you can make adjustments to your software systems development process. Then, you can measure the average value of Plindex for products developed using the changed process. If the average value has increased, then the process has probably been improved. This approach is an indirect method of measuring process improvement. In the next section, we discuss how you can directly measure your software systems development process for purposes of finding ways to improve it.

There are at least two different ways that this average might be computed. One way is to simply take the average of the values of Plindex for all the products that you are including in a given period of time. A second method is to look at the average as the length of a vector whose components are the average values of the corresponding attributes that went into computing Plindex for each of the products being included in the period of time. We illustrate these two methods by considering a very simple example consisting of two products. We assume for simplicity that only two product attributes are included in the definition of product “goodness.” Also, to keep the computations simple, we will not normalize the index values to the range zero to one. Let the values of the attributes for the first product be, respectively, 0.3 and 0.4. Plindex for this product would then be the square root of 0.09 + 0.16, or 0.5. Let the values of the attributes for the second product be, respectively, 0.4 and 0.0 so that Plindex for this product would be 0.4. Using the first way of computing averages, the average value of Plindex would be (0.5 + 0.4)/2 = 0.45. Using the second way of computing averages, the average value of Plindex would be as follows:

\[
\sqrt{\frac{(0.3 + 0.4)^2}{2}} + \sqrt{\frac{(0.4 + 0.0)^2}{2}} = 0.403.
\]
Figure 6–19  *Plindex* can be used indirectly to measure improvements in the organizational software development process.
6.4 Process Integrity

The purpose of this section is to give you guidance for defining a set of process metrics that can be used in conjunction with the product integrity index discussed in the preceding section. Process integrity metrics can provide you with input for improving your software systems development process.

As previously stated, one dictionary definition of process is the following:

*A series of actions or operations leading to an end.*

Chapter 3 described how to define our example software systems development process. There, we showed you how to define the actions (activities) and their relationships whose purpose is to produce software products (i.e., documents, computer software systems, and/or databases). Here, we show you how to measure that software systems development process in terms of its process components and component activities. Before we measure our example process, we need to define process integrity.

Process integrity is defined as follows:

*A process, when performed as part of software product development, has integrity if the process components and associated component activities are performed as part of product development in accordance with ADPE element content.*

As shown in Figure 6–20, by analogy to the product integrity index, \( P_{index} \), we define a process integrity index, \( P_{onlindex} \). The figure shows that process integrity is more complicated than product integrity. Simply stated, \( P_{index} \) is two layers deep, and \( P_{onlindex} \) is three layers deep. At the first level, the process is decomposed into components. At the second level, each process component is decomposed into activities. At the third level, value scales are defined for each activity.

The form of the process integrity equations is similar to the form of the product integrity equation (i.e., Equation 6.3–2). At the first level, the process component \( (x_t) \) replaces the product attribute \( (a_t) \) in the numerator. “Maximum value of \( x_t \)” replaces “maximum value of \( a_t \)” in the denominator. At the second level, the process component \( (x_{t_i}) \) is analogously defined in terms of process component activities \( (x_{t_i}) \).

To explain \( P_{onlindex} \) further, we discuss how to measure the software systems development process described in Chapter 3. As shown in Figure 6–21, this process consists of the following four process components \( (x_t) \): 

\[ x_t \]

\( x_{t_1} \) Seller Project Planning—The Seller Process Engineering Group is responsible for planning the work to be accomplished based on a customer’s statement of work (SOW).
Chapter 6 • Measurement

Process Integrity Index

\[ \text{ProcIndex} = \frac{\sqrt{\sum_{i=1}^{N} w_i x_i^2}}{\sqrt{\sum_{i=1}^{N} w_i^2 (\text{maximum } x_i^2)}} \]

\( N \) = number of organizational process components
\( w_i \) = weighting factor for process component \( x_i \)
\( x_i \) = the \( i \)th process component
\( \text{maximum } x_i \) = maximum value of \( x_i \)

\[ x_i = \frac{\sqrt{\sum_{j=1}^{N_i} w_{ij} x_{ij}^2}}{\sqrt{\sum_{j=1}^{N_i} w_{ij}^2 (\text{maximum } x_{ij}^2)}} \]

\( N_i \) = number of activities making up process component \( x_i \)
\( w_{ij} \) = weighting factor for activity \( x_{ij} \) of process component \( x_i \)
\( x_{ij} \) = the \( j \)th activity of the \( i \)th process component
\( \text{maximum } x_{ij} \) = maximum value of \( x_{ij} \) for each \( j \)

Figure 6-20 This figure presents the general formula for the process integrity index, \( \text{ProcIndex} \), that is normalized to one.

\( x_{f2} \) Seller Development Team—This team is responsible for accomplishing the work specified in the project plan.

\( x_{f3} \) Customer/Seller Development Team—This team is responsible for coordinating project activities with one another.

\( x_{f4} \) Seller Senior Management—This management level is responsible for reviewing and approving project products for delivery to the customer.

There are seven process integrity measurement steps. To set the stage for explaining these steps, Figure 6-22 depicts how we define \( \text{ProcIndex} \) for our example software systems development process. Now we will walk you through the seven process integrity measurement steps.
Figure 6–21 The software systems development process can be measured by assessing specific process components. In this example, four process components are shown.
Figure 6-22  The left-hand side of this figure represents our process measurement framework that is used to decompose a process into its components and activities. Activity value scales are defined in terms meaningful to the organization. The right-hand side of this figure represents how our example organizational software systems development process maps to our framework.
Process Integrity Measurement Step 1

The first process measurement step is to decide on the questions we want and/or need to address. Here, we are addressing the following question:

Is the organizational software systems development process producing “good” products?

Process Integrity Measurement Step 2

The second process measurement step is to select the process components from the organizational software systems development process that we want to measure. We have selected the following four process components as previously described: (xt1) Seller Project Planning, (xt2) Seller Development Team, (xt3) Customer/Seller Development Team, and (xt4) Seller Senior Management. These four process components are the first layer of the process metric calculation. Figure 6–23 illustrates how the process integrity index, ProclIndex, is calculated by using these four process components.

Process Integrity Measurement Step 3

Before we calculate each process component, we need to identify the process component activities that we want to measure. Each process component (xti) needs to be defined in terms of specific activities (xtij). For example, Seller Project Planning (xt1), is defined by its six process activities:

xt11 Seller reviews SOW, communicates with customer, and assembles project planning team.
xt12 Seller performs risk assessment.
xt13 Seller Project Planning Team develops task-derived resource estimates.
xt14 Seller Business Manager calculates task-derived dollar estimates.
xt15 Seller Business Manager calculates risk-derived dollar estimates.
xt16 Seller Management reconciles task-derived dollar estimates with risk-derived dollar estimates.

These six activities represent the second layer of process metric calculation for a process component. Figure 6–24 gives a complete list of process activities for each of the four process components. Measuring activities can help to identify those activities that are, or are not, contributing to customer satisfaction. Those activities that are not contributing directly to customer satisfac-
Chapter 6 • Measurement

Organizational Software Systems Development Process Measurement

Figure 6-23 This figure illustrates how the process integrity index, \( Proclindex \), is calculated by using four process components—\((xt_1)\) Seller Project Planning (which includes risk assessment), \((xt_2)\) Seller Development Team (which includes peer reviews), \((xt_3)\) Customer/Seller Development Team (which includes CCB activity), and \((xt_4)\) Seller Senior Management (which includes review and approval activities).
Organizational Software Systems Development Process Components and Associated Activities

| (xt₁) — Seller plans work based on customer’s SOW | (xt₂) — Seller Development Team accomplishes work specified in the project plan | (xt₃) — Customer/Seller Development Team members coordinate project activities with one another | (xt₄) — Seller Senior Management reviews and approves project products for delivery to customer |
| (xt₅) — Seller reviews SOW, communicates with customer and assembles project planning team | (xt₆) — Seller Project Manager communicates with customer and evolves software products | (xt₇) — Customer Project Manager provides technical guidance to Seller Project Manager | (xt₈) — Seller Senior Management reviews project products to determine if products conform to ADPE |
| (xt₉) — Seller performs risk assessment | (xt₁₀) — Seller Lead Developer establishes project files | (xt₁₁) — Seller and Customer Project Managers hold project CCBs | (xt₁₂) — Seller Senior Management approves products for delivery to customer |

Figure 6–24 Example activities for our organizational software systems development process.

...tion may be candidates for modification or elimination. We categorize such modifications as process improvement activities.

Figure 6–25 illustrates how the process integrity index, Prociindex, is further defined and calculated by using the process activities for each of the four components. As shown in the figure, Seller Project Planning and Seller Development Team each consists of six activities; Customer/Seller Development Team and Seller Senior Management each consists of two activities.

**Process Integrity Measurement Step 4**

Before measurement data can be collected, the third layer of the process metric calculation needs to be performed. Specifically, the fourth process measurement step is to define an activity value scale for each process activity. Figures 6–26, 6–27, and 6–28 present example activity value scales for each of the component activities. The activity value scales are expressed in everyday terms.

Some of the activity value scales shown have only two values, while others have three or four values. The values on the scale help to influence the
To compute Proclindex, each process component is decomposed into specific activities.
**Figure 6–26** Example activity value scales for the Seller Project Planning component of the organizational process.
Figure 6–27  Example activity value scales for the Seller Development Team component of our organizational process.
Figure 6–28 Example activity value scales for the Customer/Seller Development Team and Seller Senior Management components of our organizational process.
direction the organization wants to go. For example, in Figure 6–26, activity (xt_{12})—Seller performs risk assessment—has two values (i.e., 0.0 and 1.0). In this example, the organization (or perhaps the buyer) places importance on risk assessments, and it is expected that this activity is to be done. If risk assessment does not happen, a value of 0.0 is assigned. As will be shown, when this value is plotted on a Kiviat-like diagram, it is readily apparent that this activity has not been done.

When defining value scales, it is necessary to understand what specific item(s) or action(s) trigger the activity that is to be measured. Such items or actions are called measurement triggers. For example, for “(xt_{12})—Seller performs risk assessment,” value scales can be defined as “Seller did not perform risk assessment on customer’s SOW = 0.0,” and “Seller performed risk assessment on customer’s SOW = 1.0.” You can use measurement triggers to help assign values to process activities.

Activity value scales may provide an opportunity to show more gradual improvement in performing a specific activity. For example, in Figure 6–27, activity (xt_{23})—Seller Lead Developer conducts peer reviews—has four values (i.e., 0.0, 0.25, 0.75, and 1.0). In this example, the organization may have some projects that routinely use peer reviews, and they have found them to be useful. Therefore, the organization wants to change its development culture such that all projects have peer reviews. However, it is recognized that it may take some time to get everyone up to speed on how to conduct and to document peer reviews. In some instances, some developers may be reluctant to have their work reviewed, or some customers may not want to pay for such reviews. Changing the culture may take time; therefore, the scale can accommodate incremental improvements, rather than an all-or-nothing approach.

Activity value scales can be established to measure customer, as well as seller, activities. For example, in Figure 6–28, activity (xt_{31})—Customer Project Manager provides technical guidance to Seller Project Manager—attempts to measure customer communication. In an effort to be responsive to customer needs, potential confusion (with respect to what needs to be done next, or what the “real” requirements are, etc.) can occur if clear communication channels are not established. This value scale is set up to reward customer and seller communication, but it emphasizes that the customer and seller management should be talking with one another. The customer manager tells the seller manager what is needed, and the seller manager supervises and directs the seller development team members. Another example is activity (xt_{32})—Seller and Customer Project Managers hold project CCBs. This value scale is set up to measure if agreed-upon communication channels and decision making are being followed. The seller has established an ADPE element for making decisions at CCBs, and the customer has agreed to this mechanism. This value scale is set up to reward holding project CCBs in accordance with the ADPE guidance. If the customer and seller do hold CCBs in accordance with the guidance but are not happy with how the CCBs are working, then the ADPE guidance can be changed. There is no one way to hold the
Process Integrity Measurement Step 5

Figure 6–29 illustrates the four process components that we are interested in measuring, and the sixteen corresponding component activities and scales. The fifth process measurement step is to observe activity accomplishment and to choose a corresponding scale value reflecting that accomplishment. For example, for “\(x_{12}\)—Seller performed risk assessment,” we observe that the customer’s SOW was assessed for risk according to our organization’s risk assessment procedure. Thus, the observed activity accomplishment is “Seller performed risk assessment on customer’s SOW,” and consequently the corresponding scale value reflecting that accomplishment is 1.0.

Remember, the measurement trigger can be different for each activity. Thus, to measure an entire process, a number of triggers are generally needed.

Process Integrity Measurement Step 6

The sixth process measurement step is to use the formulas given in this chapter to calculate the process component value based on the activity values. For simplicity, we use the formulas that yield values between zero and one. We also select weighting factors that reflect our perception of the relative importance of our process component activities. For our example, we set all weighting factors to one.

Electronic spreadsheets can be established to capture the measurement data, calculate the metrics, and display the results. Figure 6–30 illustrates an example of what the results may look like.

Process Integrity Measurement Step 7

Once all the activity values have been assigned, the seventh process measurement step is to combine the process component values into a process integrity index. In this example, the results are input into the \(\text{Proclindex}\) equation. As shown in the top panel of Figure 6–30, \(\text{Proclindex}\) is equal to 0.59. By examining the details in the lower panel of the figure and referring to the corresponding value scales, the following observations are made:

- **Seller Project Planning (\(x_{1}\)) = 0.85.** The seller reviewed the SOW, communicated with the customer to discuss any questions, and assembled a project planning team. A risk assessment was performed, and the planning team used its expert judgment to develop task-derived resource estimates.
Figure 6-29  Proclindex is defined and calculated in terms of process components, component activities, and activity value scales.
Figure 6-30 This figure illustrates one way to display the results of quantifying a software development process. On the basis of the example measures, the process integrity index, Proclindex, equals 0.59.
The seller business management calculated the task-derived and risk-derived dollar estimates so that the management team could compare the top-down risk estimate with the bottom-up task estimate. The seller managers got together to discuss the estimates, and they could not agree. The senior manager made a decision.

♦ **Seller Development Team** ($x_{t2}$) = 0.56. The cost and schedule were tracked on an ad hoc basis. The person who was assigned the Lead Developer position established project files but did not do it according to the schedule. The Lead Developer held peer reviews but they were not documented. The independent product assurance support was tracked on a periodic basis. The project documentation was only partially edited. Because of several concurrent projects, the seller management did not mentor the seller project management.

♦ **Customer/Seller Development Team** ($x_{t3}$) = 0.50. The customer management communicated with the seller developers, and there was some confusion regarding requirements. This confusion was fostered by the fact that the CCBs that were held did not document decisions regarding the requirements.

♦ **Seller Senior Management** ($x_{t4}$) = 0.35. The seller management was overloaded with work and did not take the time to review the work before approving it for delivery to the customer.

At this point, we need to decide on whether our question—Is the organizational software systems development process producing “good” products?—has been answered. We suggest meeting with the appropriate people to examine the observations and discuss how to address the corresponding results. This measurement process helps to focus on what activities the organization needs to address. We would also suggest looking at the corresponding product integrity results.

We summarize our process measurement discussion in Figure 6–31, which lists seven steps to follow when setting up and collecting process metrics. Our recommendation is to start simple and try a pilot measurement program.

After you decide on what questions you want answered, the process components, and the process component activities you want to measure, you need to decide on the granularity of your process measurements. We recommend that you do not select too many process component activities at first. You need to collect data and determine what activities are worth measuring. If you have many activities, each activity’s contribution to the measurement is reduced accordingly (unless weighting factors are used). However, you do not want to measure too few activities, else you may not be gaining insight into whether or not your process is consistently producing products that satisfy your customer.

As a result of reviewing the preceding measurement observations, decision makers can focus their attention (and potentially, resources) on those activi-
Figure 6–31 This high-level procedure helps you through the process measurement steps based on the concepts and examples introduced in this chapter.

Chapter 6 • Measurement
ties that may need improvement or on questions that need to be answered. The decision might be to take more measurements and review them carefully. Perhaps, the software development process needs to be more closely followed; maybe the process needs to be changed, or maybe the management is overcommitted. Regardless, these measurements are expressed in everyday terms to be used consistently to achieve customer satisfaction in terms of value scales that make sense for your organization.

6.5 Capability Maturity Model (CMM) for Software

The purpose of this section is to show how the process integrity concept and formulas discussed in the preceding sections can be applied to a widely known framework for improving software systems development—the Capability Maturity Model (CMM) for Software developed by the Software Engineering Institute (SEI). We assume that you are familiar with the CMM, and you can skip this section without loss of continuity. However, to set context and to link with the previous discussion of process measurement, we present a brief summary of the model. A complete description of Version 1.1 of the model can be found in the following publications (see the bibliography at the end of this book for a brief description of each of these documents):


The CMM for Software (hereafter referred to as the CMM) summary given in the following paragraphs is adapted from the first two documents in the list.

The CMM is a five-level road map for improving the software process. The CMM is a guide (not a cookbook) for evolving toward a culture of software engineering excellence. It is a model for organizational improvement. The CMM provides a framework for improving software engineering practice. The CMM provides guidelines for not only improving process management but also for introducing technology into an organization. Furthermore, the

---

8See the bibliography for a brief description of the SEI mission.
CMM is an underlying structure for consistent software process improvement efforts. An organization can perform these exercises on itself to assess its capability to produce good software products consistently. Customers can perform corollary evaluation exercises on prospective software development vendors to help assess the risk of doing business with those vendors. Figure 6–32 depicts the five maturity levels—(1) Initial, (2) Repeatable, (3) Defined, (4) Managed, and (5) Optimizing.

As indicated in the figure, Level 1 organizations produce software by some amorphous process that is only known to a few individuals or heros. During the course of the project, the project leader ends up saying something like the following:

We only have a few weeks before delivery. Kiss your spouses, boy friends, girl friends, dogs, cats, whatever, goodbye for the next three weeks. By the way, that includes nights and weekends, as well. That's what it is going to take to get it done.

With luck, the work somehow gets done. However, even in the best of circumstances, it is difficult to account for everything that is needed for successful software development to take place. Therefore, the SEI defines each maturity level as a layer in the foundation for continuous process improvement. A maturity level is a well-defined evolutionary plateau on the path toward becoming a mature software organization. Associated with each maturity level (except Level 1) is a “software process capability” that describes the range of expected results from following a process.

As indicated in Figure 6–32, the software process is essentially ad hoc and generally undisciplined for a Level 1 organization. A Level 1 organization’s process capability is unpredictable because the software process is constantly changed as the work progresses. Level 1 performance depends on the individual capabilities of the staff and managers and varies with their innate skills, knowledge, and motivations. Level 2 organizations focus on project management. The process capability of an organization has been elevated by establishing a disciplined process under sound management control. In contrast to a Level 1 organization, at Level 2 a repeatable process exists for software projects. At Level 3, the focus shifts to establishing organizationwide processes for management and engineering activities. Level 3 processes evolve from the processes and success while achieving Level 2. At Level 2, one or two projects may have repeatable processes, but at Level 3 all projects use the processes. At Level 4, the measurements that have been put in place at Level 2 and Level 3 are used to understand and control software processes

*We have shown the levels to be parallel because we believe that most organizations operate at multiple levels, at the same time. In contrast, the SEI literature presents the levels in a staircase-like fashion to indicate that an organization needs to establish itself at one level before moving up to the next level.
Chapter 6 • Measurement

<table>
<thead>
<tr>
<th>Software Process Maturity Level</th>
<th>Lessons Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optimizing</strong></td>
<td>Reality is that organizations operate at multiple levels at the same time.</td>
</tr>
<tr>
<td></td>
<td>Continuous process improvement enabled by <strong>quantitative feedback</strong> from process and from testing innovative ideas and technologies.</td>
</tr>
<tr>
<td><strong>Managed</strong></td>
<td>There are few Level 5 organizations.</td>
</tr>
<tr>
<td></td>
<td>Detailed measurements of software process and product quality collected.</td>
</tr>
<tr>
<td></td>
<td>Both software process and products <strong>quantitatively understood</strong> and controlled using detailed measures.</td>
</tr>
<tr>
<td><strong>Defined</strong></td>
<td>Measurement contributes to product and process “quality.”</td>
</tr>
<tr>
<td></td>
<td>Software process for both management and engineering activities is <strong>documented, standardized</strong>, and integrated into organizationwide process.</td>
</tr>
<tr>
<td></td>
<td><strong>All</strong> projects use process.</td>
</tr>
<tr>
<td><strong>Repeatable</strong></td>
<td>Peer reviews help reduce errors.</td>
</tr>
<tr>
<td></td>
<td>Basic project <strong>management</strong> processes established to track cost, schedule, functionality.</td>
</tr>
<tr>
<td></td>
<td>Necessary process discipline in place to <strong>repeat</strong> earlier successes with similar applications.</td>
</tr>
<tr>
<td><strong>Initial (ad hoc)</strong></td>
<td>Project planning is key to Level 2 success.</td>
</tr>
<tr>
<td></td>
<td>Software product (usually) produced by some <strong>amorphous</strong> process.</td>
</tr>
<tr>
<td></td>
<td>Performance driven by competence and <strong>heroics</strong> of people doing work.</td>
</tr>
<tr>
<td><strong>Managed</strong></td>
<td>There are few Level 5 organizations.</td>
</tr>
<tr>
<td><strong>Defined</strong></td>
<td>Measurement contributes to product and process “quality.”</td>
</tr>
<tr>
<td><strong>Repeatable</strong></td>
<td>Peer reviews help reduce errors.</td>
</tr>
<tr>
<td><strong>Initial (ad hoc)</strong></td>
<td>Project planning is key to Level 2 success.</td>
</tr>
<tr>
<td><strong>Optimizing</strong></td>
<td>Reality is that organizations operate at multiple levels at the same time.</td>
</tr>
<tr>
<td></td>
<td>Continuous process improvement enabled by <strong>quantitative feedback</strong> from process and from testing innovative ideas and technologies.</td>
</tr>
</tbody>
</table>

Figure 6–32 The Software Engineering Institute’s Capability Maturity Model for Software is a five-level road map for improving an organization’s software systems development process. Each maturity level is a well-defined evolutionary plateau on the path toward becoming a “mature” software organization.

438
and products quantitatively. At Level 5, continuous process improvement is enabled by quantitative process feedback and technology insertion.

Each maturity level consists of “key process areas (KPAs)” that are defined by “key practices.”10 Key process areas identify the issues that must be addressed to achieve a maturity level. KPAs are a cluster of related activities that, when performed collectively, achieve a set of goals considered important for enhancing process capability. KPAs are defined to reside at a single maturity level.11 For example, as shown in Figure 6–33, associated with Level 2 are six KPAs—(1) Requirements Management, (2) Software Project Planning, (3) Software Project Tracking and Oversight, (4) Software Subcontract Management, (5) Software Quality Assurance, and (6) Software Configuration Management.

Associated with each of these six KPAs is a set of goals. For example, associated with the Software Project Planning KPA are three goals, one of which is the following: “Software estimates are documented for use in planning and tracking the software project.” Goals are associated with key practices which are the policies, procedures, and activities that contribute most to the effective institutionalization and implementation of a goal (and therefore a KPA). A key practice can be associated with more than one goal.

Key practices are grouped into five common features—(1) Commitment to Perform, (2) Ability to Perform, (3) Activities Performed, (4) Measurement and Analysis, and (5) Verifying Implementation. The key practices that make up the common features represent the “what needs to be done” or, simply stated, the requirements. Although many practices contribute to success in developing effective software, the key practices were identified because of their effectiveness in improving an organization’s capability in a particular key process area. Implementation of the key practices is the “how” part of institutionalization of KPAs.

With the preceding as background, we present the following simplified example of how process integrity can be computed for each of the SEI maturity levels or KPAs. This example is presented using the seven process measurement steps:

---

10 Regarding the use of the word “key” here and elsewhere in the model, the description of the CMM includes the following statements:

The adjective “key” implies that there are process areas (and processes) that are not key to achieving a maturity level. The CMM does not describe all the process areas in detail that are involved with developing and maintaining software. Certain process areas have been identified as key determiners of process capability; these are the ones described in the CMM.

Although other issues affect process performance, the key process areas were identified because of their effectiveness in improving an organization’s software process capability. They may be considered the requirements for achieving a maturity level.

11 This constraint may be removed in future versions of the model. The measurement approach subsequently discussed in this section is not tied to this constraint.
Figure 6-33  Each maturity level consists of "key process areas (KPAs)." Each KPA is characterized, in part, by "goals" and "key practices."
Process Measurement Step 1

Decide on what questions you want and/or need to address. Is my project performing activities associated with the Level 2 KPAs?

Process Measurement Step 2

Select the process components from your software systems development process that you want to measure. As shown in Figure 6–34, we select the Level 2 KPAs as the process components to be measured.

Process Measurement Step 3

Identify the process component activities that you want to measure. For each Level 2 KPA, we identify the Activities Performed common feature as the process component activities to be measured. For simplicity of explanation,

---

Figure 6–34  A repeatable software process that has integrity is one that has the following six process components shown above—\( \{x_1, x_2, x_3, x_4, x_5, x_6\} \).
Figure 6–35 shows only the detail for the Requirements Management KPA. The process component activities to be measured for Requirements Management correspond to the activities labeled RM.AC.1, RM.AC.2, and RM.AC.3. Other process component activities would be measured for the remaining Level 2 KPAs.

**Process Measurement Step 4**

For each identified activity, define a value scale in everyday terms that are familiar to the organization. Figure 6–36 shows our activity value scale definitions for the Requirements Management activities. The \( xt_{11} \) activity value scale is continuous and is based on a percentage of the requirements reviewed by the engineering group.\(^{13}\) The \( xt_{12} \) activity value scale has three discrete values designed to encourage review and incorporation of changes to agreed-upon requirements before they are incorporated into the project. The \( xt_{13} \) activity value scale reflects either “yes” or “no.” This activity value scale was designed to stress whether or not the activity is being performed.

Similarly, to compute integrity values for the other five KPAs, value scales for each of the activities associated with each KPA would be defined. The resulting Level 2 process “goodness” scale is shown in Figure 6–37.

**Process Measurement Step 5**

For each identified activity, observe activity accomplishment and choose a corresponding scale value reflecting that accomplishment. Observations are made, values are assigned, and values are recorded using a software spreadsheet program.

**Process Measurement Step 6**

Using the formulas given in this chapter, calculate the process component value based on the activity values. For simplicity, use the formulas that yield values between zero and one. Select weighting factors to reflect your perception of the relative importance of your process components.

---

\(^{12}\)In Figure 6–35, the term “allocated requirements” is used in the CMM to denote those system requirements that are set apart (i.e., allocated) for implementation through software code.

\(^{13}\)In the language of our book, “engineering group” is an organizational element encompassing development disciplines.
Chapter 6  •  Measurement

Figure 6–35  The Requirements Management process component (i.e., key process area) can be measured using the three activities labeled RM.AC.1, RM.AC.2, and RM.AC.3.
Chapter 6 • Measurement

As shown in Figure 6–38, the generalized process integrity index formula is used to establish the formulas necessary to measure the activities associated with the Level 2 KPAs. As indicated in the figure, the formulas are normalized so that the calculated values will fall between zero and one, and the weighting factors are set equal to one.

**Process Measurement Step 7**

Using the formulas given in this chapter, combine the process component values into a process integrity index value. For simplicity, use the formulas that yield values between zero and one. Select weighting factors to reflect your perception of the relative importance to your organization.

A formula for rolling the KPA integrity values up into a Level 2 integrity index is shown in Figure 6–38. For the Requirements Management KPA (\(xt_1\)), there are 3 activities; hence, the square root of 3 in the denominator. For the Software Project Planning KPA (\(xt_2\)), there are 15 activities; hence, the square root of 15 in the denominator for the formula. For the Software Project Tracking and Oversight KPA (\(xt_3\)), there are 13 activities; for the Software Subcontract Management KPA (\(xt_4\)), there are 13 activities; for the Software Quality
Chapter 6  •  Measurement

SEI Level 2
Process Integrity
Index

Max. Value 1.0

Repeatable

Software Process

Min. Value 0.0

The key process area (KPA) activities are not performed.

Figure 6–37  The Level 2 process “goodness” scale ranges from a minimum value of 0.0 (i.e., activities not being performed in any KPA) to a maximum value of 1.0 (i.e., all activities being performed in each KPA).

Assurance KPA ($x^5$), there are 8 activities; and for the Software Configuration Management KPA ($x^6$), there are 10 activities. Thus, in terms of determining an organization’s compliance with Level 2 KPAs and underlying practices, the process integrity vector resides in a space of 62 dimensions ($3 + 15 + 13 + 13 + 8 + 10$). A similar approach can be used to compute process integrity indices for the other CMM levels.
Process Integrity Index

\[ \text{ProcIndex} = \sqrt{\frac{\sum_{i=1}^{N} w_i^2 x_i^2}{\sum_{i=1}^{N} w_i^2 \text{(maximum } [x_i])^2}} \]

\( N \) = number of organizational process components
\( w_i \) = weighting factor for process component \( x_i \)
\( x_i \) = the \( i \)th process component
\( \text{maximum } [x_i] \) = maximum value of \( x_i \)

\[ x_j = \sqrt{\frac{\sum_{i=1}^{N_j} w_{ij}^2 x_{ij}^2}{\sum_{i=1}^{N_j} w_{ij}^2 \text{(maximum } [x_{ij}])^2}} \]

\( N_j \) = number of activities making up process component \( x_i \)
\( w_{ij} \) = weighting factor for activity \( x_{ij} \) of process component \( x_i \)
\( x_{ij} \) = the \( j \)th activity of the \( i \)th process component
\( \text{maximum } [x_{ij}] \) = maximum value of \( x_{ij} \) for each \( j \)

SEI Level 2—Repeatable ProcIndex

\( N = 6, \) all \( w_i = 1, \) all maximum \( [x_i] = 1 \)
\( N_1 = 3, N_2 = 15, N_3 = N_4 = 13, N_5 = 8, N_6 = 10, \) all \( w_{ij} = 1, \) all maximum \( [x_{ij}] = 1 \)

\( x_{1i} = \) Requirements Management, \( x_{2i} = \) Software Project Planning, \( x_{3i} = \) Software Project Tracking and Oversight, \( x_{4i} = \) Software Subcontract Management, \( x_{5i} = \) Software Quality Assurance, and \( x_{6i} = \) Software Configuration Management

\[ \text{ProcIndex} = \sqrt{\frac{\sum_{i=1}^{6} x_{i}^2}{6}} = \sqrt{\frac{x_{1i}^2 + x_{2i}^2 + x_{3i}^2 + x_{4i}^2 + x_{5i}^2 + x_{6i}^2}{6}} \]

\[ x_{1i} = \sqrt{\frac{\sum_{j=1}^{3} x_{1ij}^2}{3}} \]
\[ x_{2i} = \sqrt{\frac{\sum_{j=1}^{15} x_{2ij}^2}{15}} \]
\[ x_{3i} = \sqrt{\frac{\sum_{j=1}^{13} x_{3ij}^2}{13}} \]
\[ x_{4i} = \sqrt{\frac{\sum_{j=1}^{13} x_{4ij}^2}{13}} \]
\[ x_{5i} = \sqrt{\frac{\sum_{j=1}^{8} x_{5ij}^2}{8}} \]
\[ x_{6i} = \sqrt{\frac{\sum_{j=1}^{10} x_{6ij}^2}{10}} \]

Figure 6–38 The process integrity index for CMM Level 2 can be defined using the activities for each of the six Key Process Areas. For example, there are three activities for Requirements Management (i.e., \( x_{1i} \)), fifteen activities for Software Project Planning (i.e., \( x_{2i} \)), etc.
6.6 Other Process-Related Measurements

In addition to product integrity and process integrity measurements, it may be useful to establish other process-related measurements tied to one or more components of the software systems development process. Again, the question is, “What attributes of the software systems development process are of interest to measure?” In part, the answer is tied to determining which activities contribute to “staying in business,” which is strongly tied to “customer satisfaction.” “Customer satisfaction” can be expressed in many ways. In this section, we show you how to effect process improvement, using an approach other than product and process integrity indexes. Our approach consists of the following two steps:

♦ The application of metrics to the software systems development process activities to provide insight into the extent to which these activities are, or are not, contributing to customer satisfaction (as expressed in terms of the five product integrity attributes\(^4\)).
♦ Those activities that are not contributing to customer satisfaction will be modified (or eliminated) until they do. These modifications are what “process improvement” means.

We now explain how to apply these two steps to derive a set of process metrics. The context for this discussion is our example organizational software systems development process. This process is sufficiently general so that you will be able to adapt it to your own environment.

The discussion that follows assumes that the organizational process is used to govern a number of projects unfolding, more or less, in parallel. We measure things on individual projects and then average these things over one or more projects. From these averages, we derive findings about the underlying software systems development process to effect its improvement.

In order to perform actual measurements of software systems development processes, the preceding considerations need to be tempered by practical considerations. Measurement involves collecting data and putting the data into a meaningful form for process improvement purposes. These tasks cannot be onerous because they will get in the way of software systems development work—and measurement will not be performed. Thus, as we stressed in preceding sections, the metrics must be simple to collect and analyze. The price for this simplicity is that the metrics are limited regarding the insight they provide into process workings. For the near term, your approach should be to collect some simple metrics to see if they help highlight activities that should

\(^4\)Remember, in Section 6.3, we explained the concept of product integrity in terms of the following product attributes: \((at_1)\) fulfills customer needs, \((at_2)\) can be easily and completely traced through its life cycle, \((at_3)\) meets specified performance criteria, \((at_4)\) meets cost expectations, and \((at_5)\) meets delivery expectations.
be changed to effect process improvement. Through this experience, you can then determine whether you need more sophisticated measurement techniques.

The simplicity criterion just mentioned means in a metrics context that we simply count the number of times specific software systems development activities are performed. To bring in customer satisfaction, we use the receipt of deliverable and acceptance of deliverable forms. The Acceptance of Deliverable form provides customer feedback regarding each product delivered according to the following three degrees of "customer satisfaction" (in descending order of this satisfaction):

♦ The product is accepted as delivered
♦ The product is accepted with minor changes needed
♦ The product requires changes to be negotiated.

Clearly, the data on the acceptance of deliverable form do not provide detailed insight into the extent to which the product fulfills specified customer needs (i.e., product integrity attribute $a_{t_1}$) or meets specified performance criteria (i.e., product integrity attribute $a_{t_3}$). In terms of overall process improvement, these detailed considerations are not pertinent. For instance, our example organizational software systems development process mandates that, before computer code is delivered to the customer, it must be acceptance tested. This acceptance testing activity does address the details of the product integrity attributes $a_{t_1}$ and $a_{t_3}$ for that computer code product. In fact, if acceptance testing does demonstrate the presence of these attributes, customer confirmation on the acceptance of deliverable form is a foregone conclusion. The point is that (1) counting these forms, (2) putting these counts into the three bins of degrees of customer satisfaction just listed, and then (3) relating these counts to the number of times certain activities are carried out, does provide gross insight into the effectiveness of these activities in the overall software systems development process.

To relate the preceding discussion to actual process measurement, we discuss some specific process-related metrics. We begin with a general process-related metric and then illustrate it with specific examples. This general metric and the associated examples are a starting point for defining a set of process-related metrics to provide some insight into the state of your software systems development process and to effect its improvement. We also consider other metrics to illustrate additional process-related measurement ideas.

The general process-related metric is the following:

$$M_{1q} = \frac{\sum_{i=1}^{\#Del} (NProcActivity_{qi})}{\#Del}$$

(6.6–1)
Chapter 6 • Measurement

$M_{1q}$ is the average number of times it takes to perform activity $q$ in the organizational software systems development process in producing the $i^{th}$ deliverable before delivery. The quantity $N_{ProcActivity_{qi}}$ is the number of times the $q^{th}$ process activity is performed on the $i^{th}$ deliverable before delivery. The quantity $#Del$ is the number of deliverables to include in the average. This number may apply to a specific project or to a group of projects. For example, $#Del$ may be the number of deliverables produced on a project over a three-month period. As another example, the quantity $#Del$ may be the number of deliverables produced on all projects under the supervision of a particular seller senior manager. Examples of the quantity $N_{ProcActivity_{qi}}$ are the following:

♦ Number of CCB meetings where the $i^{th}$ deliverable was discussed
♦ Number of peer reviews to produce the $i^{th}$ deliverable
♦ Number of product assurance reviews to produce the $i^{th}$ deliverable
♦ Number of technical edits to produce the $i^{th}$ deliverable
♦ Number of management reviews of the $i^{th}$ deliverable

The metric $M_{1q}$ can indicate the following, depending on $#Del$ included in the sum:

♦ The extent to which the $q^{th}$ organizational software systems development process activity is being used to produce deliverables
♦ The average number of times activity $q$ is required to get a deliverable to the customer
♦ The trend in the average number of times activity $q$ is required to get a deliverable to the customer (this trend would be measured by collecting and reporting the metric, for example, every month for a given value of $#Del$)

To illustrate $M_{1q}$, let $N_{ProcActivity_{qi}} = N_{Peer_i}$, the number of peer reviews required to produce the $i^{th}$ deliverable. Then, we define the metric $MPeer$ related to the organizational software systems development process peer review activity as follows:

$$MPeer = \frac{\sum_{i=1}^{#Del} (N_{Peer_i})}{#Del}$$  \hspace{1cm} (6.6–2)

This metric is the average number of peer reviews required to produce deliverables for delivery. If, for example, the sum in Equation 6.6–2 is restricted to a single project, this metric indicates the following:

♦ The extent to which the peer reviews are being used on the project
♦ The average number of peer reviews required to get a deliverable to the customer for that project
If this metric were collected and reported, say, monthly, the trend in the average number of peer reviews required to get a deliverable to the customer for that project

If similar statistics were compiled for other projects, then we could determine for subsequent project planning purposes how many peer reviews to include in the cost and schedule of project plans. This information would serve to improve the project planning process called out in your organizational software systems development process because it would help to refine the costing and scheduling algorithms.

However, the metric in Equation 6.6–2 does not explicitly address product integrity attributes. The following metric, which is an adaptation of Equation 6.6–2, illustrates how connection to these attributes can be made:

$$M_{PeerACC} = \frac{\sum_{i=1}^{#DelAcc} (N_{Peer_i})}{#DelAcc}$$

(6.6–3)

This metric is the average number of peer reviews required to produce deliverables that are accepted by the customer (i.e., the customer returns the acceptance of deliverable form indicating “the product is accepted as delivered”); the quantity #DelAcc is the number of such deliverables. If, for example, the sum in Equation 6.6–3 is restricted to a single project, this metric indicates the following:

♦ The average number of peer reviews required to get a deliverable accepted by the customer for that project

♦ If this metric were collected and reported, say, monthly, the trend in the average number of peer reviews required to get a deliverable accepted by the customer for that project

If similar statistics were compiled for other projects, then we could see whether there is a correlation between the number of peer reviews and customer acceptance. Of course, other organizational software systems development process activities influence customer acceptance. It is thus admittedly an oversimplification to say that there is a single value for this average that should be applied across all projects to enhance the likelihood of customer acceptance of products. For example, certain projects might involve the development of complex products that by their nature would require more peer reviews than less complex products would.

But the preceding metric could provide some insight into the correlation between the peer review activity and customer satisfaction as expressed on the acceptance of deliverable form, as follows:
Suppose several projects have a track record of consistent product acceptance by the customer. Suppose also that the value of MPeerACC obtained by averaging over these projects is, say, 3.5 (i.e., three to four peer reviews are used to produce deliverables on these projects). Furthermore, suppose that (1) several other projects have a track record of consistent deliverables requiring “changes to be negotiated” and (2) the value of MPeerACC averaged over these projects is, say, 0.5 (i.e., one or no peer reviews are used to produce deliverables on these projects). Then, other organizational software systems development process activities being equal (admittedly, a big “if,” but this “if” could be examined by applying the instantiations of the metric M1q for these activities), it could be surmised that there is some correlation between the peer review activity and customer acceptance of products.

Of course, things generally turn out to be far more complicated than the simple situation just illustrated. To get a sense of such complications, we modify the situation so that it reads as follows:

Suppose several projects have a track record of consistent product acceptance by the customer. Suppose also that the value of MPeerACC averaged over these projects is, say, 3.5 (i.e., on average, three to four peer reviews are used to produce deliverables on these projects). Furthermore, suppose that (1) several other projects have a track record of consistent deliverables requiring “changes to be negotiated” and (2) the value of MPeerAcc averaged over these projects is, say, 4.5. Then, if other organizational software systems development process activities were being applied consistently across both sets of projects (which could be determined by applying the instantiations of the metric M1q to these activities), the organizational software systems development process peer review activity would need to be examined from perspectives such as the following: (1) Are there fundamental differences between the way peer reviews are being utilized on one set of projects versus the other set (e.g., are the peer reviews on the project set with MPeerACC = 4.5 less formal with no written record of what was accomplished)? (2) Are the two sets of projects fundamentally different in terms of the nature of their products so that it is not meaningful to say that, because it takes more peer reviews on these projects, the peer review process is less effective? For (1), the process improvement response might be to modify the way that set of projects performs its peer reviews so that it mirrors the way the other set of projects performs peer reviews. If this modification brings MPeerACC in line with the value of this metric for the other set of projects, then the effectiveness of the peer review activity would be demonstrated. For (2), the process improvement response might be to modify the organizational software systems development process to call out two approaches to peer reviews—one for projects that mirror the one project set and the other that mirror the other project set. The result of this organizational software systems development process modification would be process improvement.

It should be noted that an assumption underlying this analysis is that if there is a correlation between doing an activity $N$ times and customer product ac-
ceptance, then doing the activity much less than \( N \) or more than \( N \) is less desirable. This assumption helps simplify the analysis. It is not possible to make general statements about the validity of this assumption. You will need to see what makes sense for your organization. For example, you may need to specify the number of times a given activity is to be performed. Then, you can observe the effect on customer acceptance of products and make adjustments accordingly.

Metrics such as the one defined in Equation 6.6–3 can be extended to encompass more than one organizational software systems development process activity. This approach may be useful if it proves too difficult to correlate a specific activity with customer satisfaction. For example, it may be more useful to lump peer reviews and independent product assurance reviews into a single metric. This metric might provide insight into the correlation between detailed technical product reviews (which these activities are intended to address) and customer satisfaction. Extending the Equation 6.6–3 idea, such a metric might be the following:

\[
MTR_{Rev\; ACC} = \frac{\sum_{i=1}^{#\; DelAcc} (NPeer_i + NPA_i)}{#\; DelAcc}
\]  

(6.6–4)

This metric is the average number of peer reviews and independent product assurance reviews required to produce deliverables that are accepted by the customer (i.e., the customer returns the acceptance of deliverable form indicating “the product is accepted as delivered”). As was the case in Equation 6.6–3, \( NPeer_i \) is the number of peer reviews required to produce the \( i^{th} \) deliverable accepted by the customer. Similarly, \( NPA_i \) is the number of independent product assurance reviews required to produce the \( i^{th} \) deliverable accepted by the customer. If, for example, the sum in Equation 6.6–4 is restricted to a single project, this metric indicates the following:

- The average number of detailed product technical reviews (i.e., peer reviews and independent product assurance reviews) required to get a deliverable accepted by the customer for that project
- If this metric were collected and reported, say, monthly, the trend in the average number of detailed product technical reviews required to get a deliverable accepted by the customer for that project

If similar statistics were compiled for other projects, then we could see if there is a correlation between the number of detailed product technical reviews and customer acceptance.

In addition to activity-specific metrics such as those just discussed, there are metrics that can address product integrity attributes by simply counting the
number of deliverables over a specific period of time. For example, the following metric addresses the product integrity attribute of “at _5—meets delivery expectations”:

\[
\%\text{DelOnTime} = \left( \frac{\#\text{DelOnTime}}{\#\text{Del}} \right) \times 100
\]

This metric gives the percentage of deliverables delivered on time to the customer during a specific period for certain projects, where “on time” is according to delivery dates specified in project plans or CCB minutes. The quantity \( \#\text{Del} \) is the number of deliverables delivered during a specific period for specific projects. The quantity \( \#\text{DelOnTime} \) is the number of these deliverables delivered on time. For example, \( \#\text{Del} \) may be the number of deliverables delivered during a particular month for all projects active during that month. As another example, \( \#\text{Del} \) may be the number of deliverables on a specific project during the entire period the project was active. The preceding metric provides insight into how well the organization is meeting planned schedules. This insight, in turn, provides insight into the effectiveness of the project planning activity in scheduling deliverables and the effectiveness of the CCB activity in updating project plan schedules. Ideally, the organization should strive to have \( \%\text{DelOnTime} = 100 \) (all deliverables are delivered on time) for whatever \( \#\text{Del} \) encompasses. If \( \%\text{DelOnTime} \) encompasses all projects and if the value of this metric is significantly less than 100 (say, 50), then this statistic would be used to investigate whether (1) certain organizational process activities should be accomplished in less time and/or (2) the project planning activity needs to be revised to set more realistic schedules. This investigation, in turn, may precipitate changes to the activities in question—the end result being organization process improvement.

Regarding the project planning activity, this activity precedes actual organization product development work. Yet, there are product integrity issues regarding this activity. Sometimes the customer pays for the product resulting from this activity—namely, the project plan. If the cost of this activity has been a major customer concern in the past, one metric that can possibly help in this area is the following:

\[
\text{AvPPlan}\$ = \frac{\sum_{i=1}^{\#\text{Projects}} (P\text{Plan}\$_i)}{\#\text{Projects}}
\]

The metric \( \text{AvPPlan}\$ \) is the average cost to produce a project plan resulting in a project. \( P\text{Plan}\$_i \) is the cost to produce the \( i^{th} \) project plan resulting in a pro-
#Projects is the number of projects to include in the average. This average can be computed over any period. Thus, for example, #Projects could be the number of projects in a six-month period. By computing this average periodically (e.g., monthly), the trend in this average can be determined (e.g., the average cost to produce a project plan has declined at a rate of 10 percent per month for the last six months). Also, the project plans to include in the sum can be limited by defining #Projects appropriately. Thus, for example, the metric in Equation 6.6–6 can be used to compute the average cost to produce a project plan for a specific customer by limiting #Projects and PPlan$$_i$$ to customer “ABC Corporation” projects. This metric can also be used to define the average cost of project plans for various categories of projects. For example, by limiting #Projects and PPlan$$_i$$ to “O&M” projects, we can compute the average project planning cost for O&M work. The metric AvPPlan$$_i$$ can also help set customer expectations regarding the cost of the organization’s project planning process.

Embedded in the last metric is the number of iterations required to produce a project plan before it results in a project. The following metric can give visibility into these iterations and thereby provide additional insight into how to control project planning cost (and thereby increase the integrity of the project planning part of the organization’s way of doing business):

$$\text{Av#PPlan} = \frac{\sum_{i=1}^{#\text{Projects}} (P\text{Plan}_i)}{#\text{Projects}} \quad (6.6-7)$$

The metric Av#PPlan$$_i$$ is the average number of drafts required to produce a project plan resulting in a project. PPlan$$_i$$ is the number of drafts required to produce the $$i^{th}$$ project plan resulting in a project. #Projects is the number of projects to include in the average. This average can be computed over any period. Thus, for example, #Projects could be the number of projects in a six-month period. By computing this average periodically (e.g., monthly), the trend in this average can be determined (e.g., the average number of drafts required to produce a project plan has declined at a rate of 10 percent per month for the last six months). Also, the project plans to include in the sum can be limited by defining #Projects appropriately. Thus, for example, the metric in Equation 6.6–7 can be used to compute the average number of drafts required to produce a project plan for a specific customer, by limiting #Projects and PPlan$$_i$$ to customer “ABC Corporation” projects. This latter statistic could be used to determine whether there was a shortcoming in the project planning process or a difficulty with a particular customer (or some combination of these two considerations). The metric in Equation 6.6–7 can also be used to determine the number of drafts required to produce project plans for various categories of projects. For example, by limiting #Projects and PPlan$$_i$$,
to “O&M” projects, we can compute the average number of drafts required to produce project plans for O&M work. This statistic, in turn, can help refine the project planning activity through scheduling algorithms that depend on the type of work that the organization is being requested to do. For example, if $Av\#PPlan$ turns out to be 3.5 for O&M work and 1.5 for the development of new software systems, and other factors being equal (e.g., there are no customer dependencies), then we can inform customers that three to four meetings between the customer and the organization will probably be required to finalize an O&M project plan, while one to two meetings between the customer and the organization will probably be required to finalize a new-systems-development project plan.

One organizational software systems development process activity often given special attention is acceptance testing. Unless specifically dictated by the customer, software systems should not be delivered to the customer without acceptance testing. Furthermore, customer involvement in the acceptance testing CCBs is an effective way of assuring that the software system to be delivered does what it is supposed to do, that is, fulfills specified customer needs and/or meets specified performance criteria. The following two metrics can be used to assess respectively the extent to which (1) software systems are being acceptance tested before delivery and (2) the customer participates in acceptance testing CCB activity:

$$\frac{\#\text{SystemsAccTested}}{\#\text{SystemsDel}} \times 100$$  \hspace{1cm} (6.6–8)

$$\frac{\#\text{AccTestedSystemsDel}}{\#\text{SystemsAccTested}} \times 100$$  \hspace{1cm} (6.6–9)

In Equation 6.6–8, the metric $\%\text{SystemsAccTested}$ gives the percentage of software systems accepted tested during a specific period for certain projects. The quantity $\#\text{SystemsDel}$ is the number of software systems delivered during a specific period for specific projects. The quantity $\#\text{AccTestedSystemsDel}$ is the number of these systems acceptance tested. For example, $\#\text{SystemsDel}$ may be the number of software systems delivered during a particular month for all projects active during that month. As another example, $\#\text{SystemsDel}$ may be the number of systems delivered on a specific project during the entire period that the project was active. Ideally, the organization should strive to have $\%\text{SystemsAccTested} = 100$ (all software systems are acceptance tested before de-

---

15 As discussed in Chapter 5, acceptance testing is the system-level testing that the seller performs before delivering the system to the customer. We recommend that the customer be involved in the acceptance testing process. In this way, what the customer receives at delivery is what the customer expects.
livery) for whatever \#SystemsDel encompasses. If \%SystemsAccTested encompasses all projects and if the value of this metric is significantly less than 100 (say, 50), then this statistic would be used to investigate why acceptance testing is not being performed. The reasons uncovered may precipitate changes to one or more organizational process activities. If, for example, a reason uncovered was that senior customer management issued edicts—at the eleventh hour—that no acceptance testing be performed, then it may be necessary to clarify the organizational software systems development process to ensure seller senior management involvement with customer senior management prior to planned acceptance testing. The end result of such changes to the activities in question is organizational process improvement.

In Equation 6.6–9, the metric \%SystemsAccTested with Customer gives the percentage of the acceptance testing activity conducted with customer participation in acceptance testing CCBs, during a specific period for certain projects. The quantity \#SystemsAccTested with Customer is the number of software systems delivered during a specific period for specific projects where the customer participated in acceptance testing CCBs. The quantity \#SystemsAccTested is the number of systems acceptance tested during this period for the specific projects in question. Ideally, the organization should strive to have \%SystemsAccTested with Customer = 100 (all software systems are acceptance tested with customer involvement before delivery) for whatever \#SystemsAccTested encompasses. The presumption here is that, when test incident report (TIR) resolution involves the customer, our first and third product integrity attributes are satisfied by definition.

Another potentially useful approach to assessing process effectiveness at a gross level (i.e., independent of any particular organizational process activity) is to do simple counts on the contents of the acceptance of deliverable form. The following metric illustrates this approach for the case in which the contents of the form indicate “Accept as Delivered” (analogous metrics can be defined for the cases “Accept with Minor Changes” and “Requires Changes to be Negotiated”):

\[
\text{CustomerSatisfied} = \left[ \frac{\#\text{FormAcc} + \#\text{Unknown}}{\#\text{DelKnown} + \#\text{DelUnknown}} \right] \times 100
\]  

(6.6–10)

This metric offers a variety of interpretations depending on how the factors in the numerator and denominator are used. The scope of the metric depends on the scope of the factors in the denominator. The quantities in this equation are defined as follows:

- \#FormAcc is the number of deliverables for which the organization has received an acceptance of deliverable form indicating “Accept as Delivered.”
- \#Unknown is the number of deliverables for which the organization has not yet received an acceptance of deliverable form and for which an assumed value will be assigned.
#DelKnown is the number of deliverables for which the organization has received an acceptance of deliverable form.

#DelUnknown is the number of deliverables for which the organization has not yet received an acceptance of deliverable form.

CustomerSatisfied is the percentage of deliverables for which the customer has indicated “Accept as Delivered” and thus is a gross measure of positive customer perception of the organization.

The following example illustrates how this metric can be used:

Suppose we are interested in customer perception of the organization for a two-month period across all projects. Suppose further that during this period the organization delivered 100 deliverables and received 50 acceptance of deliverable forms, 25 of which indicated “Accept as Delivered.” In this case, #DelKnown = 50, #DelUnknown = 50, and #FormAcc = 25. Regarding the 50 deliverables for which the forms have not yet been received, suppose we conjecture that none of them will come back with “Accept as Delivered” (a highly undesirable case for these outstanding deliverables, the worst case being that all the forms come back indicating “Changes to Be Negotiated”). With this conjecture, #Unknown = 0, and the value of CustomerSatisfied becomes 25 percent. If, on the other hand, we conjecture that they all will come back with “Accept as Delivered,” #Unknown = 50, and the value of CustomerSatisfied becomes 75 percent. These values then have the global interpretation that, for the two-month period in question, the degree of customer satisfaction with organization products across all projects is no worse than 25 percent and no better than 75 percent. If, in fact, there is a good likelihood that indeed most of the outstanding acceptance of deliverable forms will come back with other than “Accept as Delivered” indicated (as can be determined by querying the responsible project managers), then the 25 percent figure would be more representative of the state of organization affairs regarding customer perceptions. In this case, a detailed look at other metrics would be called for to see why the approval rating is so low. Equation 6.6–10 can also be used to assess customer satisfaction solely on the basis of the known status of the deliverables. For this assessment, the quantities #UnKnown and #DelUnknown are both set to zero. Using the numbers just given, the value of CustomerSatisfaction for the two-month period in question would then become 50 percent. This value would have the following interpretation: “For the two-month period in question, half the deliverables for which the customer returned acceptance of deliverable forms were judged acceptable as delivered.”

The preceding discussion focused on metrics pertaining to organization process improvement in a product integrity context. Other quantities, although not directly related to process improvement, may offer insight into organization work profiles that could eventually be used to effect process improvement. Regarding these profiles, questions such as the following might be asked:
Chapter 6 • Measurement

♦ What is the average size project?
♦ What is the average project cost?
♦ How is the work distributed across the customer’s organization?

Regarding the first question, a metric such as the following, patterned after the metric in Equation 6.6–6, might be helpful:

\[
AvProjectPersSize = \frac{\sum_{i=1}^{#Projects} (ProjectPersSize_i)}{#Projects}
\] (6.6–11)

The metric \( AvProjectPersSize \) is the average number of organization employees working on an organization project. \( ProjectPersSize_i \) is the number of organization employees working on the \( i \)th project. \( #Projects \) is the number of projects to include in the average. This average can be computed over any period. Thus, for example, \( #Projects \) could be the maximum number of projects active in a six-month period. By computing this average periodically (e.g., monthly), the trend in this average can be determined (e.g., the average project size has declined at a rate of two people per month for the last six months). Such trends, when coupled to trends derived from other metrics such as \( CustomerSatisfied \) in Equation 6.6–10, may indicate general failure (or success) of the organization process. For example, declining project size coupled with increased customer satisfaction during the same period may indicate process success because the organization is able to do good work with fewer people. On the other hand, declining project size coupled with declining customer satisfaction for the same period may indicate process failure because the customer’s organization is taking its business elsewhere.

The project types to include in the Equation 6.6–11 sum can be limited by defining \( #Projects \) appropriately. Thus, for example, Equation 6.6–11 can be used to compute the project size for a specific customer by limiting \( #Projects \) and \( ProjectPersSize \), to ABC Corporation projects. This metric can also be used to define the average project size for various categories of projects. For example, by limiting \( #Projects \) and \( ProjectPersSize \), to “O&M” projects, we can compute the average project size for O&M work. Again, coupling these metrics to other metrics can provide insight into failure (or success) of the organization process in particular spheres. For example, by limiting \( #Projects \) and \( ProjectPersSize \), to “O&M” projects, and by limiting the inputs to the \( CustomerSatisfied \) metric in Equation 6.6–10 to O&M deliverables, we can gain insight into how well, or poorly, the process may be working for O&M work. For instance, a trend in O&M project size showing a decline and a customer satisfaction trend for these projects for the same period showing an increase may indicate process success for O&M work. That is, these two trends may indicate that the organization is able to do good O&M work with fewer people.
The following are some counting issues that need to be considered when using the metric in Equation 6.6–11:

♦ Should product assurance personnel be included in $ProjectPersSize_i$?
♦ Should support personnel (e.g., technical editors) be included in $ProjectPersSize_i$?

A global response to these issues is that values quoted for the metric should indicate what $ProjectPersSize_i$ includes.

The second question in the list—what is the average project cost?—can be addressed by a metric analogous to the one given in Equation 6.6–11, namely

$$AvProjectS = \frac{\sum_{i=1}^{#Projects} (ProjectS_i)}{#Projects} \quad (6.6–12)$$

The metric $AvProjectS$ is the average cost of an organization project. $ProjectS_i$ is the cost of the $i^{th}$ project. $#Projects$ is the number of projects to include in the average. This average can be computed over any period. Thus, for example, $#Projects$ could be the maximum number of projects active in a six-month period. By computing this average periodically (e.g., monthly), the trend in this average can be determined (e.g., the average project cost has declined at a rate of $10,000$ per month for the last six months). Such trends, when coupled to trends derived from other metrics such as $CustomerSatisfied$ in Equation 6.6–10, may indicate general failure (or success) of the organization process. For example, declining project cost coupled with increased customer satisfaction during the same period may indicate process success because the organization is able to do good work at reduced cost.

The project types to include in the Equation 6.6–12 sum can be limited by defining $#Projects$ appropriately. Thus, for example, Equation 6.6–12 can be used to compute the project cost for a specific customer by limiting $#Projects$ and $ProjectS_i$ to ABC Corporation projects. This metric can also be used to define the average project cost for various categories of projects. For example, by limiting $#Projects$ and $ProjectS_i$ to “O&M” projects, we can compute the average project cost for O&M work. Again, coupling these metrics to other metrics can provide insight into failure (or success) of the organization process in particular spheres. For example, by limiting $#Projects$ and $ProjectS_i$ to “O&M” projects and by limiting the inputs to the $CustomerSatisfied$ metric in Equation 6.6–10 to O&M deliverables, we can gain insight into how well, or poorly, the process may be working for O&M work. For instance, a trend in O&M project cost showing a decline and a customer satisfaction trend for these projects for the same period showing an increase may indicate process success for O&M...
work. That is, these two trends may indicate that the organization is able to do good O&M work at less cost.

The third question in the list—how is the work distributed across the customer’s organization?—can be addressed, for example, by using the metrics in Equations 6.6–11 and 6.6–12. As already discussed, these metrics can be used to compute the average size and cost of projects for Office X. If we perform these computations across all customer offices that the organization does business with, we generate a cost and manning profile of work across the customer’s organization. By observing trends in these profiles and by coupling these trends to the corresponding trends in the \textit{CustomerSatisfied} metric in Equation 6.6–10, we can gain insight into how well (or poorly) the organization is serving different client communities. We can use this insight to sharpen the organization’s client focus and thereby increase its business base. For example, if the trends in these metrics indicate that the organization is serving the ABC Corporation poorly, the organization can give added attention to staffing its projects with, for example, more experienced personnel than might otherwise be considered.

Table 6–1 summarizes this section by listing the metrics formulas and their definitions.

<table>
<thead>
<tr>
<th>Metric Formula</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_{1q} = \frac{\sum_{i=1}^{# Del} (NProcActivity_{q,i})}{# Del} )</td>
<td>The average number of times it takes to perform activity ( q ) in the organization software systems development process in producing the ( i )th deliverable before delivery</td>
</tr>
<tr>
<td>( MPeer = \frac{\sum_{i=1}^{# Del} (NPeer_{i})}{# Del} )</td>
<td>The average number of peer reviews required to produce deliverables for delivery</td>
</tr>
<tr>
<td>( MPeerACC = \frac{\sum_{i=1}^{# DelAcc} (NPeer_{i})}{# DelAcc} )</td>
<td>The average number of peer reviews required to produce deliverables that are accepted by the customer (i.e., the customer returns the acceptance of deliverable form indicating “the product is accepted as delivered”)</td>
</tr>
<tr>
<td>( MRevACC = \frac{\sum_{i=1}^{# DelAcc} (NPeer_{i} + NPA_{i})}{# DelAcc} )</td>
<td>The average number of peer reviews and independent product assurance reviews required to produce deliverables that are accepted by the customer (i.e., the customer returns the acceptance of deliverable form indicating “the product is accepted as delivered”)</td>
</tr>
</tbody>
</table>

(continued)
### Metric Formula Definition

#### %DelOnTime

\[
\%\text{DelOnTime} = \left( \frac{\# \text{DelOnTime}}{\# \text{Del}} \right) \times 100
\]

The percentage of deliverables delivered on time to the customer during a specific period for certain projects, where “on time” is according to delivery dates specified in project plans or CCB minutes.

#### AvPPlan$\$\$

\[
\text{AvPPlan} = \frac{\sum_{i=1}^{\# \text{Projects}} (\text{PPlan}_i)}{\# \text{Projects}}
\]

The average cost to produce a project plan resulting in a project.

#### Av# PPlan$\Delta$

\[
\text{Av# PPlan} = \frac{\sum_{i=1}^{\# \text{Projects}} (\text{PPlan}_i)}{\# \text{Projects}}
\]

The average number of drafts required to produce a project plan resulting in a project.

#### %SystemsAccTested

\[
\%\text{SystemsAccTested} = \left( \frac{\# \text{AccTestedSystemsDel}}{\# \text{SystemsDel}} \right) \times 100
\]

The percentage of software systems accepted tested during a specific period for certain projects.

#### %SysAccTestedwithCustomer

\[
\%\text{SysAccTestedwithCustomer} = \left( \frac{\# \text{AccTestedSystemswithCustomer}}{\# \text{SystemsAccTested}} \right) \times 100
\]

The percentage of the acceptance testing activity conducted with customer participation in acceptance testing CCBs, during a specific period for certain projects.

#### CustomerSatisfied

\[
\text{CustomerSatisfied} = \left( \frac{\# \text{FormAcc} + \# \text{Unknown}}{\# \text{DelKnown} + \# \text{DelUnknown}} \right) \times 100
\]

The customer perception of the seller organization.

#### AvProjectPersSize

\[
\text{AvProjectPersSize} = \frac{\sum_{i=1}^{\# \text{Projects}} (\text{ProjectPersSize}_i)}{\# \text{Projects}}
\]

The average number of seller organization employees working on a customer’s project.

#### AvProject$\$

\[
\text{AvProject} = \frac{\sum_{i=1}^{\# \text{Projects}} (\text{Project}_i)}{\# \text{Projects}}
\]

The average cost of a project.
6.7 Measurement Summary

Software systems development processes produce software products, such as requirements specifications and computer code. A product can be measured by assessing product attributes, and a process can be measured by assessing its process components and corresponding activities. As shown in Figure 6–39, product and process measurements can be used to help improve software systems development. However, numbers can be used to prove almost anything. As Mark Twain once said, “There are three kinds of lies: lies, damned lies, and statistics.”

When you set out to establish your product and process metrics program, it is important to think through ahead of time what the measurements are going to be used for. With this purpose in mind, a measurement program can be based on values that are significant to both the seller and the customer.

Figure 6–39  Measurements can be used to help improve software systems development processes and the resultant products.

A word of warning—be sensitive to the concern within your organization that people may view product and process measurements as measuring them. Asking such questions such as, “What is Sam’s productivity? Is he turning out as many lines of code as Roger or Sally?” is tricky business. In fact, we recommend that you avoid such direct questions. We suggest questions that probe the product or process may be more acceptable. Questions such as, “Can we consistently produce products that satisfy our customer? Do we have a development process that produces products that satisfy our customer and make a profit?” may be of more value. Measurements must answer questions that are important to the organization—otherwise, they are not worth collecting. As Figure 6–40 illustrates, there are multiple viewpoints

**Figure 6–40** The product integrity index or process integrity index can be implemented for organization and project perspectives.
when measuring products and process. These views apply to any organization that consists of more than one software project. That is, the organization has a process that each project adapts to its special needs to accomplish its tasks. The organization is aiming to improve its software systems development process and resultant products, while the project is aiming to improve task-level performance and corresponding products. The project measurements can be used as a feedback mechanism to improve the organizational process. Figure 6–41 summarizes the possible relationship between the product integrity index, $\text{Plindex}$, and the process integrity index, $\text{Proclindex}$.

Your actual results depend on your specific set of circumstances. You need to examine your measurements to understand how the results may guide your product and process improvement activities. It is recommended that the data

---

**Figure 6–41** What is the relationship between your product and process integrity indexes? This figure suggests some possible interpretations.

**Diagram:**

- **Am I producing "good" products?**
  - **Plindex:**
    - High Scores
    - Medium Scores
    - Low Scores
  - **Proclindex:**
    - High Scores
    - Medium Scores
    - Low Scores

1. If the product integrity indexes are consistently high and the process integrity indexes are consistently low, it may mean that you are not following your process but that you are producing "good" products. Possibly, the development team is relying on the heroics of a few people. If your goal is to have "good" products consistently produced by a "good" process, then you may need to reconsider your process or goal.

2. If the product integrity indexes are consistently low and the process integrity indexes are consistently high, look to change your process. This situation may mean that you are following your process but that you are not producing "good" products.

3. If both integrity indexes are consistently low, it may be that you are not following significant parts of your process.

4. If both integrity indexes are consistently high, then you may have to fine-tune your process to push the indexes to one.

5. If both integrity indexes are consistently in the medium range, then you may have to make changes to your process to elevate the indexes.
be collected, reviewed, and discussed on a routine schedule. Data collection is a part of everyday work. Figure 6–42 illustrates this point.

An analog to integrated process measurement is the measurements that occur as an automobile moves. Such integrated measurements include speed, available fuel, engine temperature, and oil level. Among other things, these measurements indicate how well the automobile is functioning. These measurements also indicate when the automobile’s performance may need to be improved. Example performance improvement measurements include measuring the miles traveled between stops for gas to provide insight into fuel economy which, in turn, offers insight into which parts of the automobile may need to be serviced.

In Figure 6–43, we summarize the product and process measurement steps, and the possible relationships between the measurements. This figure can be used as a guideline for setting up, observing, collecting, and analyzing your measurements.

![Diagram](image-url)

**Figure 6–42** Applying metrics to the software systems development process should be part of the process itself.
This high-level procedure is to help you through the product and process measurement steps based on the concepts and examples introduced in this chapter. Figure 6–43

**Product Integrity Measurement Steps**

- Decide on the questions that you want and/or need to address (e.g., am I producing “good” products?).
- Select the products from your software systems development process that you want to measure (e.g., requirements specification).
- Identify the product attributes that you want to measure (e.g., for a requirements specification, you might identify an attribute as “ar — Meets cost expectations”).
- For each identified attribute (e.g., ar), define a value scale in everyday terms that are familiar to the organization (e.g., delivered for more than the cost estimate = 0.0, delivered for cost estimate = 0.9, and delivered for less than cost estimate = 1.0).
- Using the formulas given in this chapter, calculate the product integrity index value. For simplicity, use the formulas that yield values between zero and one. Select weighting factors to reflect your perception of the relative importance of your product attributes.

**Process Integrity Measurement Steps**

- Decide on the questions that you want and/or need to address (e.g., is my process consistently producing “good” products within budget?).
- Select the process components from your software systems development process that you want to measure (e.g., xt — Seller Project Planning).
- Identify the process component activities that you want to measure (e.g., for Seller Project Planning, you might identify an activity as “xt — Seller performs risk assessment”).
- For each identified activity, define a value scale in everyday terms that are familiar to the organization. Identify the specific item(s) or action(s) [i.e., measurement trigger(s)] that triggers the activity to be measured. For example, for Seller performs risk assessment, you might define scale values as “Seller did not perform risk assessment on customer’s SOW = 0.0”, and “Seller performed risk assessment on customer’s SOW = 1.0”.
- Use the measurement trigger when assigning a value to each activity (e.g., for Seller performed risk assessment, you observe that the customer’s SOW was assessed for risk according to the organization’s risk assessment procedure; therefore, the assigned value = 1.0).
- Using formulas given in this chapter, calculate the process component value based on the activity values. For simplicity, use the formulas that yield values between zero and one. Select weighting factors to reflect your perception of the relative importance of your process component activities.
- Using the formulas given in this chapter, combine the process component values into a process integrity index value. For simplicity, use the formulas that yield values between zero and one. Select weighting factors to reflect your perception of the relative importance of your process components.

**Possible Measurement Results**

- If the product integrity indexes are consistently high and the process integrity indexes are consistently low, it may mean that you are not following your process but that you are producing “good” products. Possibly, the development team is relying on the heroics of a few people. If your goal is to have “good” products consistently produced by a “good” process, then you may need to reconsider your process or goal.
- If the product integrity indexes are consistently low and the process integrity indexes are consistently high, look to change your process. This situation may mean that you are following your process but that you are not producing “good” products.
- If both integrity indexes are consistently low, it may be that you are following your process but not producing significant parts of your process.

* If both integrity indexes are consistently high, then you may have to fine-tune your process to push the indexes to one.
* If both integrity indexes are consistently in the medium range, then you may have to make changes to your process to elevate the indexes.

**Figure 6–43** This high-level procedure is to help you through the product and process measurement steps based on the concepts and examples introduced in this chapter.
You can use the annotated outline of an ADPE guideline in Figure 6–44 as a starting point for defining your organization’s measurement program. This outline consists of the following sections:

♦ **Purpose.** This section states the purpose of the guideline. The purpose sets the context and establishes the authority for the guideline.

♦ **Background and Measurement Issues.** This section provides an overview of your organization, business, customers, and types of contract vehicles (e.g., fixed price, time, and materials) that you use to conduct business. Measurement issues are identified and expressed in terms of specific questions that the organization or project wants or needs to have addressed.

♦ **Product and Process Improvement Approach.** This section defines how the product and process measurement steps introduced in this chapter are to be used. The section defines, details, and walks through the measurement steps. It is recommended that high-level figures be used to explain the steps. Depending on the level of detail appropriate for your organization, appendices can be used to explain the steps and responsibilities in more detail.

♦ **Product and Process Measurements.** This section defines the specific formulas to be used to answer your specific set of questions. Example calculations can be given to show how to use the equations. Suggested reporting formats may also be included.

♦ **Roles and Responsibilities.** This section presents the major organizational responsibilities for the measurement program.

♦ **Appendices.** Appendices are added as necessary. The main body of the guideline states the basics, and the appendices can add additional detail that embodies lessons learned, or provide tutorial information. As an organization matures in its engineering business processes, we recommend that the lessons be captured and incorporated into your ADPE elements. As people in your organization move on to other jobs, their knowledge can be incorporated into your ADPE elements, which serve, in part, as a piece of your organization’s corporate memory.

In closing our discussion of the application of **Object Measurement** to product and process “goodness,” we want to offer some additional remarks concerning (1) alternatives to vector length for representing and computing “goodness” indexes, (2) how our measurement concept can be extended to arbitrary levels of detail, and (3) the static viewpoint of process measurement.

**Alternatives to Vector Length for Computing Indexes** When we discussed measuring product “goodness,” we started by saying that product integrity is a multidimensional concept that associates attributes with a product. In looking for a way to quantify this multidimensional concept, we noted that mathematical and scientific disciplines often handle multidimensional quantities with entities known as “vectors.” This association then led us to use the concept of “vector length” as the basis for folding the measurements of a prod-
1.0 PURPOSE
This section states the purpose of the element. This purpose is the following:

- Identify the measurements to be performed to (1) quantify where your organization is product- and process-wise, (2) quantify differences from this baseline assessment, (3) establish quantitative process and product goals, and (4) quantify progress toward achieving these goals.
- Define the approach for incorporating process and product improvements based on the measurement activity.

2.0 BACKGROUND AND MEASUREMENT ISSUES
This section gives an overview of your organization, your business, your customers, and the types of contractual vehicles that you use to conduct business (e.g., fixed price, memorandum of understanding, time and materials). It also identifies measurement issues that your organization needs to address to improve the way it does software systems development business. One way to structure these issues is to use the multiple views introduced in this chapter. Example issues are the following:

- What is the likelihood of producing products with integrity?
- Is the product delivered on time?
- Is the product delivered within budget?
- Does the product do what the customer wants it to do?
- How maintainable is the product?
- How long does it take to get through our organizational development process?
- How are product integrity attributes quantified?
- How are process component activities quantified?

3.0 PRODUCT AND PROCESS IMPROVEMENT APPROACH
This section describes how the product and process measurement steps introduced in this chapter are to be used.

4.0 PRODUCT AND PROCESS MEASUREMENTS
This section describes your responses to the Section 2 issues in terms of measurements to be performed, including any mathematical formulas and other algorithms to be used to generate numbers. This chapter provides you with a starting point for defining measurements for your organization (e.g., product integrity index, process integrity index). This section can also include example measurements using the mathematical formulas introduced in the section.

5.0 ROLES AND RESPONSIBILITIES
This section presents the major organizational responsibilities for performing and managing product and process measurement activities.

APPENDICES
Appendices can contain such things as (1) mathematical derivations and other details underlying any formulas introduced in the main body, (2) alternative measurements, (3) limitations of Section 4 measurements, (4) acronyms, and (5) definitions of key terms.

Figure 6–44 An annotated outline for getting you started in defining a product and process measurement approach for your organization. This ADPE element can also be used to refine a measurement approach you already (informally) have in place.
uct’s attributes into a single number that we called a product integrity index, or \( P_l \text{index} \). We chose \( P_l \text{index} \) as a way to measure product “goodness.” In a similar fashion, we defined a process integrity index, or \( P_r \text{index} \). \( P_r \text{index} \) folded measurements of the extent to which activities that make up a process are performed into a single number. We chose \( P_r \text{index} \) as a way to measure process “goodness.”

Throughout this book, we stress that there is no one “way” to develop software systems. Similarly, there is no one way to measure, for example, product “goodness” or process “goodness.” Earlier in this chapter, we illustrated how to use \( \text{OM} \) as a way to measure product “goodness” and process “goodness.” To calculate values for a “goodness” index, we needed a computational mechanism. We based our computational mechanism on a physical model of a line in space. This approach affords us the opportunity to “visualize” what the product and process integrity indexes mean. Namely, these indexes correspond respectively, roughly speaking, to the length of a line in product attribute space and to a length of a line in process activity space. Of course, we cannot really see such lines. The point is that, by abstracting from the underlying mathematical model of a vector, we are afforded the opportunity to sense what the indexes mean.

The preceding discussion leads to the following essential point regarding the \( \text{OM} \) approach:

\[
\text{The most important part of this approach is setting up the value scales. As we discussed earlier in this chapter, measurement results must be meaningful to the intended audience. To make meaningfulness happen within the \( \text{OM} \) framework means that the customer and seller need to mutually agree on the (1) number of tick marks to put on each scale, (2) words to be used for the marks, and (2) numeric values to associate with these words.}
\]

How to combine the measurements into a single number is less important. But, again, the customer and seller need to agree on the computational procedure to be used. And, of course, the overriding issue regarding the computational procedure is that the procedure needs to make sense to both the customer and the seller.

We now illustrate another example computational procedure for \( P_l \text{index} \) (this example can be straightforwardly extended to the computation of \( P_r \text{index} \)). Again, as with the discussion earlier in this chapter, this example computational procedure is offered as a starting point for you to derive a computational procedure that you may feel more comfortable with than that given earlier in the chapter. The example that we now offer is an adaptation of one suggested to us by others during presentations of the material in this chapter.

Equation 6.3–2 gives the formula for computing \( P_l \text{index} \) from the measurements of product attributes. As we stated earlier, because we based our computational mechanism on a physical model of a line in space, the computational procedure involves squaring numbers and taking square roots. Even though using such a
computational procedure is not a burdensome task because of the wide availability of computational tools such as spreadsheets, we now present a computational procedure that does not involve squaring numbers and taking square roots. This procedure has the advantage that, even for computations involving five to ten attributes, \(Plindex\) can be quickly estimated in one’s head. We call this computational procedure a linear one because it does not involve powers of attribute measurements (and powers of weighting factors). The following formula is a linear analog to Equation 6.3–2:

\[
Plindex = \frac{\sum_{i=1}^{n} w_i a_i}{\sum_{i=1}^{n} w_i (\text{maximum}[a_i])}
\]  

(6.7–1)

The quantities \(w_i\), \(a_i\), and \(\text{maximum}[a_i]\) have the same definitions as those given for these quantities in Equation 6.3–2. Earlier in this chapter, we applied Equation 6.3–2 (all weighting factors set to one, all value scales running from zero to one, and the normalization factor chosen to restrict \(Plindex\) to the range zero to one) to compute \(Plindex\) for several example products. With these stipulations, we found, for instance, for a requirements specification, a value for \(Plindex\) of 0.72. With these same stipulations, we find for this same requirements specification, using Equation 6.7–1, the following result:

\[
Plindex = \frac{0.5 + 0.5 + 0.5 + 1.0 + 0.9}{5} = 0.68.
\]

Thus, the value for \(Plindex\) obtained from Equation 6.7–1 is less than the value obtained from Equation 6.3–2. This result is a general one; that is, Equation 6.3–2 yields results that are less than the corresponding ones obtained from Equation 6.7–1 (they are equal for the cases \(Plindex = 0\) and \(Plindex = 1\)). The differences between the Equation 6.3–1 results and the corresponding Equation 6.7–1 results first increase and then decrease in going from \(Plindex\) values near zero to \(Plindex\) values near one. In the requirements specification example given above, this difference is 0.04. For the updated user’s manual example given earlier in this chapter, Equation 6.3–2 gave 0.32 while Equation 6.7–1 gives the following result:

\[
Plindex = \frac{0.0 + 0.5 + 0.0 + 0.0 + 0.5}{5} = 0.20.
\]

In this case, the difference is 0.12.
Regarding such differences, it is appropriate here to comment on the accuracy issue associated with setting up value scales and calculating index values obtained from measuring attributes. As we stated above, the construction of value scales involves getting the customer and seller to mutually agree on the (1) number of tick marks to put on each scale, (2) words to be used for the marks, and (3) numeric values to associate with these words. Clearly, this procedure is not an exact science. The procedure essentially involves a subjective process. For instance, in the product attribute “fulfills customer needs” that we considered earlier in this chapter, we set up three tick marks—0.0, 0.5, and 1.0. The 0.0 value corresponded to “changes to be negotiated.” This association involved making the subjective value judgment that the most undesirable situation that could arise after a product went through a product development process was that the customer received a product that was at gross variance with what he/she wanted. If the 0.0 value is to represent the most undesirable situation regarding fulfilling customer needs, other descriptions for this 0.0 value are certainly possible. For example, the customer might have stipulated that certain requirements were absolutely critical and if any one of these requirements were not addressed in the delivered product, then that product would be considered unacceptable. Similarly, the 0.5 value corresponded to “accepted with minor changes.” Issues associated with this assignment include the following:

- How is “minor” defined?
- How many minor changes must there be in a document until the number is so great that the collection of minor changes requires “changes to be negotiated”?
- Why lump all collections of minor changes into a single value (0.5)?
- Why shouldn’t a product that requires, say, five minor changes be rated higher than one requiring, say, ten minor changes?

17 Speaking somewhat loosely, at least some branches of science strive for objectivity when it comes to measurement. For example, a research technique often used to objectively evaluate the efficacy of a drug is the “double-blind procedure.” In this procedure, neither the researchers nor the subjects know who is receiving the drug and who is receiving a placebo. Independent third parties who know which subjects received which substances can then evaluate the results. These evaluators will have some degree of confidence that the participants in the procedure did not influence outcomes (i.e., introduce subjectivity) by the way they may have conducted themselves during the procedure. On the other hand, some scientific procedures may not be objective, either by design or by the nature of what is being studied. For example, wine-tasting procedures have at least some degree of subjectivity. What constitutes “good-tasting wine” is, at the most fundamental level, determined by how a wine taster reacts to his/her taste-bud sensations (and, perhaps, sense of smell). Some objectivity may be introduced into the procedure by having a number of wine tasters participate in the procedure. The dependence of the wine-tasting results on an individual wine-taster’s taste buds is thereby reduced by, say, averaging in some sense the wine-tasting results on an individual wine-taster’s reactions. So, for instance, if ten tasters participated in the evaluation, and nine of them sensed that the wine was cloyingly sweet, the results might be reported as follows: In taste tests, 90% of the tasters reported that the wine was cloyingly sweet. [Note: We are here using science in the broad sense of “methodological activity, discipline, or study.” This definition of science is the third one given in Webster’s II New College Dictionary (Boston: Houghton Mifflin Company, 1995).]
In dealing with these value assignment issues, we offer the following general guidance:

♦ At the outset of your measurement activity, you should view the index values as indicators to spur closer looks at situations. If, for example, you have index values set to the range zero to one, do not attach significance to differences of a few hundredths or even a tenth or two. Look at index values of a stream of products and note the ones that deviate by a few tenths from the others. For example, the user’s manual considered earlier in this chapter had an index value several tenths smaller than the index values of the other three products considered—0.32 versus 0.72, 0.87, and 0.96.

♦ Over time, it may make sense to put more tick marks on your value scales as people in your organization become acclimated to the measurement process. At that stage, it may make sense to, for example, take the value scale for “fulfills customer needs” and put intermediate tick marks such as 0.25 and 0.75 (in addition to 0.5). Adding such tick marks means that now index value differences of, say, 0.1 have significance so that indicator sensitivity associated with index values has increased. As we stated at the outset of this chapter, it is meaningless to try to measure lengths down to the nearest sixteenth of an inch with a ruler that contains only quarter-inch marks.

♦ The bottom line is to get acclimated to a set of value scales and determine how well they are helping you answer the questions that you wanted to answer when you set up the value scales in the first place. If these questions are still the ones you want answered but the measurements are falling short in helping you answer them, then refine the value scales or come up with new ones. If these questions have changed, then decide on a new or modified set of attributes to measure and set up the scales with a small set of tick marks. Then, again, over time, refine the value scales.

In closing this discussion of alternatives to vector lengths for computing indexes, we note that this discussion is just a glance at a multitude of considerations regarding how to apply OM to measure multidimensional quantities. To illustrate this point, we list below some considerations regarding value scale definition that we did not address. You may want to experiment with some of these considerations (or come up with some others) in setting up a measurement program for your organization.

♦ The use of negative numbers on value scales.
♦ The use of value scales where the minimum value is a desired result and the maximum value is an undesired result.
♦ The use of value scales containing irrational numbers, such as pi, as well as rational numbers. Included in this consideration is setting up value scales consisting at least in part of continuous values. Such continuous values might, for example, be defined by a function that is continuous over some interval pertinent to the measurement activity, such as the continuous function \( y = x^2 \).
One of our objectives in this chapter was to define and illustrate how to apply **Object Measurement** to real-world measurement problems in the software systems development domain. It was not our intent to give a comprehensive treatment of this measurement technique.

**Extending Measurement Formulas** The formulas for the product and process integrity indexes can be extended, if desired, to arbitrary levels of detail. For example, regarding the process integrity index, if it is desired to partition activities into subactivities, this extension can be accomplished as follows:

\[
x_{tij} = \frac{\sum_{k=1}^{N_j} w_{ijk}^2 x_{tijk}^2}{\left(\sum_{k=1}^{N_j} w_{ijk}^2 \text{maximum}[x_{tijk}]\right)^2}
\]

(6.7-2)

where

- \(N_j\) = number of subactivities making up activity \(x_{tij}\), the \(j^{th}\) activity of process component \(xt\),
- \(w_{ijk}\) = weighting factor for subactivity \(x_{tijk}\) of activity \(x_{tij}\),
- \(\text{maximum}[x_{tijk}]\) = maximum value of \(x_{tijk}\), for each \(k\),
- \(i\) is the process component label,
- \(j\) is the component activity label,
- \(k\) is the subactivity label.

The subactivities are measured directly by setting up value scales for each subactivity. Then, the contribution of each process component \(xt\) to the process integrity index is computed from the \(x_{tij}\) using the formula previously given. And, finally, \(Princt\) is computed from the \(x_t\) using the formula previously given.

Corresponding comments apply to the product integrity index. For example, each product attribute can be partitioned into subattributes \((at_{ij})\). The subattributes are measured directly by setting up value scales for each subattribute. Then, the contribution of each product attribute \(at\) to the product integrity index is computed from the \(at_{ij}\) using a formula like the one given for computing \(xt\) from \(x_{tij}\). And, finally, \(Pinct\) is computed from the \(at\) using the formula previously given.

One final comment is in order regarding the process integrity index. We suggest, that until you acquire experience using the formulas given down to the activity level, you restrict your measurements to this level. Remember, for
processes of even moderate complexity, the number of activities will gener-
ally be ten or more (the process considered earlier had sixteen activities).
Thus, unless some activities are heavily weighted, no one activity will make a
dominant contribution to the index. Consequently, if you partition the activi-
ties into subactivities, the contribution of any particular subactivity to
Proclindex will not be dominant unless it is heavily weighted. Similar com-
ments apply to the use of subattributes to determine a product integrity index.

We say that a product has integrity if it manifests the attributes at, that we
have chosen for it. If we were not interested in quantifying these attributes,
then we would say that a product lacks integrity if one or more of the chosen
attributes is missing. When we quantify these attributes (as we have done in
this chapter), the product integrity index that we calculate from these quanti-
fied attributes is a way of saying how much integrity a product has. Thus, for
example, if we evaluate Plindex on a scale ranging from zero to one and if
Plindex = 0.60, then we say that the product is 60 percent along the way to-
ward manifesting the attributes chosen for it (or, equivalently, the product is
lacking in integrity by 40 percent).

Regarding process integrity, we say that a process, when performed as part of
software product development, has integrity if the components and associ-
ated activities that make up the process are performed as part of product de-
velopment in accordance with ADPE element content. By assigning value scales to
the activities (and, thus, by implication, to the components) and measuring
the extent to which the ADPE element activities are performed as part of
product development, when we calculate Proclindex we are making a state-
ment about the extent to which the ADPE element activities and compo-
nents are performed as part of product development. Thus, for example, if
Proclindex is set up to measure the project planning process component con-
sisting of, say, ten activities as specified in an ADPE element, and if by mea-
suring these activities while we are producing a project plan it turns out that
Proclindex = 0.75 (on a scale ranging from zero to one), then we say that the
ADPE process used to produce the plan lacked integrity by 25 percent. By ex-
amining the associated Kiviat diagram or the activity values themselves, we
would obtain quantitative insight into the extent to which each project plan-
ning activity was carried out.

As we discussed, there are various combinations of Plindex and Proclindex that
can arise in practice. By analyzing these combinations, an organization can
get insight into whether (1) ADPE processes need to be changed because fol-
lowing the processes (i.e., the processes had high integrity values) leads to
products with low integrity values, or (2) the organization is falling down in
performing certain ADPE activities and the resultant products have low in-
tegrity values, or (3) the ADPE processes are okay because they are being fol-
lowed and products with high values of integrity are being produced.

Static Viewpoint of Process Measurement Our previous discussion of
process measurement is in terms of process components, component activi-
ties, and activity value scales. We stressed the importance of observing and
recording the degree to which component activities were performed. We showed you how to set up various types of value scales (e.g., binary, discrete, continuous) and suggested how you might use the observed results to improve software systems development processes. We chose to introduce our process measurement concept from this dynamic or performance-based viewpoint. However, we did not want to leave you with the impression that the performance-based viewpoint is the only way to implement process measurement.

We believe that there is a static or nonperformance-based viewpoint that deserves your consideration when setting up a process measurement program. Our car analogy in Figure 6–42 introduced the idea that performance-based process measurements can indicate when the automobile’s performance may need to be improved. However, there are times when performance does not reflect the automobile’s primary value. For instance, when the automobile designer sits down to improve the existing car line or to create a new car line, the value of the automobile can be expressed in static terms. Here, we might stress the importance of observing and recording the degree to which the automobile and supporting infrastructure exists. Is the design done? Is the design approved? Is the assembly line in place and ready to manufacture the automobile? Has documentation been prepared for the automobile dealers and their service departments? Once the automobiles are shipped to the dealerships, then the performance-based viewpoint might be more appropriate. After the automobile is ready for the junk yard, nonperformance-based measurements might be more useful than performance-based measurements. At this point in the automobile’s life, the value might be expressed in terms of automobile components that are still of value. From the junk-yard owner’s viewpoint the automobile may have valuable parts that can be salvaged (e.g., the new set of tires you just bought before the car died). From the automobile designer’s viewpoint there may be valuable lessons learned that can be incorporated into the next automobile design.

The point is that there is a temporal dimension that impacts the value of the automobile. As a result, the automobile’s value can be expressed in terms of performance, nonperformance, or some combination.

Just as the automobile’s value can be expressed in different terms, so can a process’s value be expressed in different terms. For example, as an organization is implementing an improved or new process, the value of the process may be reflected by its design, approval, documentation, and associated training. The design value scale values may be set up, for instance, as follows:

- 0.0 if the process design is not completed
- 0.5 if the process design is completed, but not approved
- 1.0 if the process design is completed and approved

18Remember, the moving automobile represents software development processes at work and the gauges represent measurement of that work.
Such a nonperformance-based value scale reflects the process’s early life. As the process is implemented, then performance-based value scales, as we have previously presented, can be constructed to reflect whether or not the process is being followed. And as the process matures, its value scales can change yet again. Regardless of how you choose to set up your measurement program, we suggest you start simple.

We have completed our discussion of product and process measurement. The next chapter is concerned with the human issues dealing with an organization undergoing a cultural change. The chapter presents cultural change issues from the following perspectives: (1) the organization responsible for developing and promulgating process elements, (2) seller project participants and project managers, (3) buyer/user project management, (4) buyer/user senior management, and (5) seller senior management.