# 3

# Measurement Fundamentals

*Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules.*
—**Fenton and Pfleeger [1]**

## 3.1 INITIAL MEASUREMENT EXERCISE

What is so hard about measuring software, you may think? OK, so why don't you try counting the number of lines of strncat.

```
/* Strncat() appends up to count characters from string
src to string dest, and then appends a terminating null
character. If copying takes place between objects that
overlap, the behavior is undefined. */
char *strncat (char *dest, const char *src, size_t count)
  {
      char *temp=dest;
      if (count) {
             while (*dest)
                    dest++;
             while ((*dest++ =*src++)) {
                    if (--count == 0) {
                           *dest='\0';
                           break;
                           }
             }
             }  return temp;
  }
```

What number did you come up with? What rules did you decide to use? In experiments with students, the answers range from 11 to 19, with occasional counts over 100 if the data definitions are included. As with most everything, it depends on how you count and what the rules are. Did you count comment lines? Did you count lines with just a "}"? If instead, we have a well-specified model that tells us "the rules," then, counting lines becomes simple, and the students all arrive at the same answer.

## 3.2 THE CHALLENGE OF MEASUREMENT

In software, the issue is that so many things that we want to measure seem so "unmeasurable." How do you measure the complexity of a program? What does complexity even mean? How do you measure productivity? If someone can write 100 lines of code in two hours to program a function, but the software has five bugs in it, is it reasonable productivity? And what is that productivity? Better yet, if someone else can program the same function in one line of code, in one hour, what is their productivity? Whose productivity is better?

Software measurement is difficult—it is abstract and difficult to visualize and touch. It is also a relatively new discipline: we still are learning.

## 3.3 MEASUREMENT MODELS

The key to "making the unmeasurable measurable" is *models*. A model is an abstraction, which strips away unnecessary details and views an entity or concept from a particular perspective. Models allow us to focus on the important parts, ignore those that are irrelevant, and hypothesize and reason about an entity. Models make measurement possible.

We must have models of whatever we want to measure. For example, say we want to know how much of the total system development effort is testing. To determine that, we need a model of both the overall development process and the testing process, which specifies when testing starts and when it ends, what is included, and the number of people involved. If our model starts with unit test by the programmer, it is a different model and will give different results than one that includes only system test.

There are three types of models you can use—*text*, *diagrammatic*, and *algorithmic*—that is, words, pictures, and numbers.

### 3.3.1 Text Models

Text models tend to be the least effective, but the most common. It is difficult to adequately describe complex situations and dynamics using just words.

Here is a text model for software development [2]:

*Effort*: The time required to develop a product, expressed as increments of staff development time (e.g., staff months/hours). In general, effort is a function of size and results in cost.

*Features*: The requirements of the product to be developed.

*Size*: The magnitude of the product to be developed. In general, size is a function of features.

*Defects*: The incompleteness of the product. In general, defects are a function of size and schedule.

*Schedule*: The total development time; completion times for principal milestones. In general, schedule is a function of effort and resources.

*Resources*: The number of developers applied to the product development.

This text model has advantages and disadvantages. Each item is clearly defined and easy to understand, but the relationships between items may be difficult to visualize. But notice that this text model describes software development in such a way that we can discuss it, measure it, and predict it: if the size changes, the number of defects will change. This text model gives structure to the abstract concept of "software development."

We frequently use metaphors and heuristics to provide insight into the software development environment dynamics. These tend to work well, due to the breadth of meaning we associate with metaphors. The downside is that these models can limit, as all models, our creative thinking as they structure it [2]. Some examples of text model metaphors for software development are:

- The Wild, Wild West
- Agile Development (both a metaphor and a name)
- Death March
- Software Factory

Notice how each metaphor evokes a different mental image and response. You probably can envision the environment, the types of people, and processes from just the few words.

Some examples of heuristics models are:

- "Adding additional staff to late projects makes them later" F.P. Brooks [3]
- "Prototyping cuts the work to produce a system by 40%" L. Bernstein

EXERCISE: (a) What is a metaphor for the software development that you do? (b) Do you have a heuristic for the percentage of effort spent in coding versus system testing? What is it?

### 3.3.2 Diagrammatic Models

Diagrammatic models can be extremely powerful. There are many techniques for diagrammatic modeling, two of which are Weinberg's [4] and Senge's [5]. They allow you to model the entities, the relationships between them, and their dynamics. Use one of the formal diagram modeling techniques if you will be doing extensive modeling. Otherwise, simplistic flow diagrams (annotated circles and arrows) should suffice. Figure 3.1 is a simple diagrammatic model of software development, which matches the text model above.

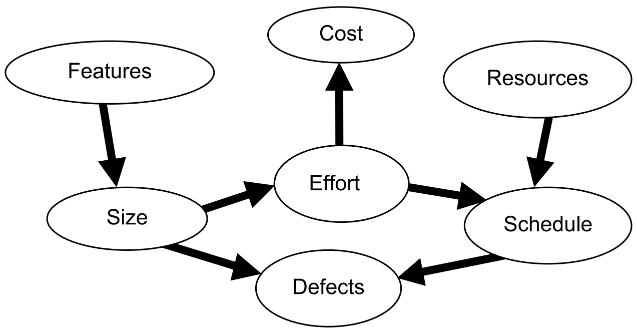

***Figure 3.1.*** *Diagrammatic model of software development.*

EXERCISE: Consider creating a diagram model of one of the software metaphors from above. What would it look like?

You may find this difficult, which is due to the connotations that we unconsciously associate with the metaphors. Although a picture may be worth a thousand words, sometimes using the right words is best.

### 3.3.3 Algorithmic Models

Algorithmic models are also called parametric models. In the right situations, they can be extremely powerful, as they can clearly describe the relationship between entities. Some examples of algorithmic models for software development are:

- Effort = Schedule $*$ Resource.
- % Defects Found During One Test Cycle = 30% of defects remaining in product.
- Effort = $A$ $*$ (Size-of-Program$^{B}$) + $C$, where $A$, $B$, and $C$ are all empirically derived constants.

### 3.3.4 Model Examples: Response Time

The three different types of models—text, diagrammatic, and algorithmic—can be used together or separately, as dictated by the situation. As a final model example, consider response time.

Text: The response time (RT) is measured from the return key to the first response on the screen. The metric is the average RT within a typical hour.

Diagrammatic: See Figure 3.2.

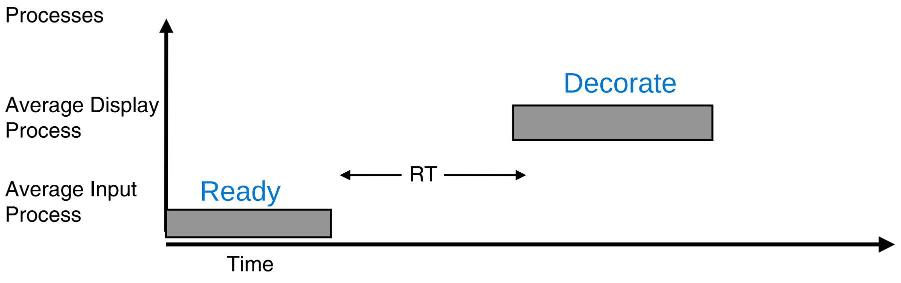

***Figure 3.2.*** *Response time model.*

Algorithmic: RT = Average (beginning of response – end of input) over a typical hour.

We must have a model of whatever we are measuring. Models document entities within a system and their interactions in a consistent manner. We cannot interpret data without reference to an underlying model, nor can we really reason about a system without an underlying model. Models allow us to consider improvement strategies and predict the expected results.

### 3.3.5 The Pantometric Paradigm: How to Measure Anything

You may be concerned about how to create a model. The Pantometric Paradigm [6] is a simple method to produce a purely visual and quantitative model of anything within the material world. You can use it to create an initial model that can evolve to meet your needs. The simple process is:

1. Reduce what you are trying to model to the minimum required by its definition. Strip away all extraneous information.
2. Visualize it on a piece of paper or in your head.
3. Divide it in fact or in your imagination into equal parts.
4. Then measure it (e.g., count the parts).

Now you have a quantitative representation (model) of your subject which matches your definition. You can now manipulate it, reason about it, experiment with it, and evolve it.

EXERCISE: How would you model a college metrics class?

*Answer*: It depends on how you want to define the class, and what is important to you. One model could be the lectures. Another might be the students taking the class. If it were the lecture, the model could be a list of the lectures by week. We could then measure the number of lectures and the number of lectures by topic.

## 3.4 META-MODEL FOR METRICS

Another method for creating models that takes abstract concepts to empirical measurements is from Kan [7], and is depicted in Figure 3.3. You begin with an abstract concept, define it, create an operational definition, and then specify a real-world measurement. An example of this methodology, using the same response time example, is shown in Figure 3.4.

There are attributes of software that we can define and measure directly, such as the number of modules in a system or the number of people assigned to a project. These are called "direct measures." However, many of the attributes we want to measure are calculated, such as number of defects per thousand lines of code

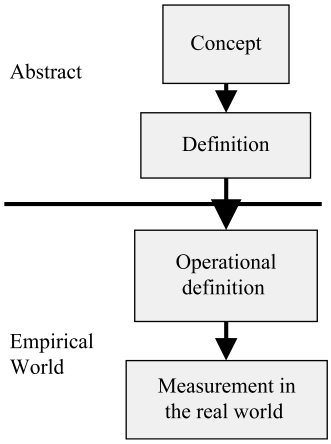

***Figure 3.3.*** *Meta-model for metrics.*

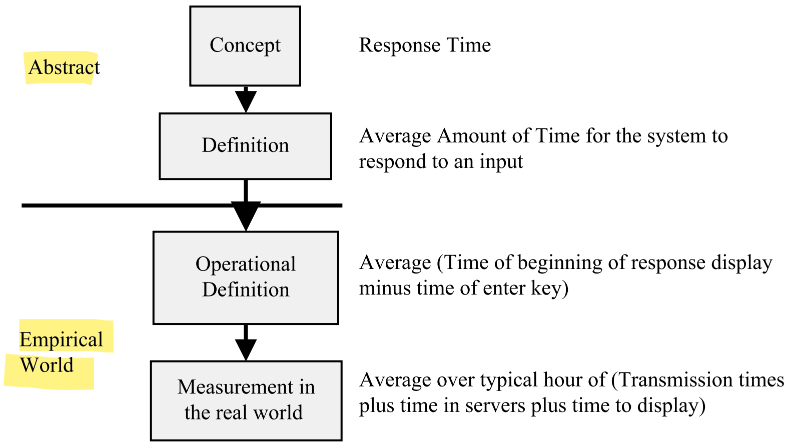
Abstract
Concept
Response Time
Definition
Average Amount of Time for the system to respond to an input
Operational Definition
Average (Time of beginning of response display minus time of enter key)
Empirical World
Measurement in the real world
Average over typical hour of (Transmission times plus time in servers plus time to display)

***Figure 3.4.*** *Example using meta-model for response time.*

(KLOC) or average number of lines of code (LOC) produced per week. These are called “indirect measures.”

EXERCISE: Define a four-level model for software reliability using the metrics meta-model.

## 3.5 THE POWER OF MEASUREMENT

Measurement is extraordinarily powerful, more so than a young professional ever suspects. What you measure is what you get. Or more accurately, it is what you get people to do. Measurement typically causes people to focus on whatever it takes to score well on that measurement. It is a specific expression of what is valued. You can easily cause unproductive behavior as people try to look the best on the wrong metrics. If you measure people by the number of lines of code they produce, they will create a larger system. If you measure people by the number of defects they create, the volume of defects will somehow start to decrease.

We both worked on a large project (400+ people) that measured field defects, which were reported to top management. One department within the project felt vulnerable and, although they had many satisfied customers, felt that they could not “afford” high-priority defect reports. Consequently, customers learned to call to talk about a problem if they wanted it fixed. If, instead, the customers filed an official defect report, it might take months (rather than days) to get the problem fixed.[1]

In the 44th Dilbert newsletter [8], we have an excellent example of measurement gone awry.

> I worked as an accountant in a paper mill where my boss decided that it would improve motivation to split a bonus between the two shifts based on what percentage of the total production each one accomplished.
>
> The workers quickly realized that it was easier to sabotage the next shift than to make more paper. Co-workers put glue in locks, loosened nuts on equipment so it would fall apart, you name it. The bonus scheme was abandoned after about ten days, to avoid all-out civil war.

Extensive productivity research was done between 1927 and 1932 at the AT&T Hawthorne plant in Cicero, Illinois. They studied manufacturing and believed that if the environment was changed in certain ways (such as more light), it would improve productivity. And they were right. Productivity improved. Then they changed another factor and measured the result. Productivity improved. Then they changed it back to the original state. Productivity improved. What was the conclusion? Whatever management paid attention to and measured improved. The difference was not the changes. It was the attention and measurement. This result is called “The Hawthorne Effect.” What you pay attention to, and measure, will improve.

[1]We do not condone or recommend this type of behavior. We only report that it occurs.

You need to believe in and remember the power of measurement. It drives behavior. If you ever question it, remember that when your 5th grade teacher told you that you had to write neatly to get an A, you did your best to write neatly.

## 3.6 MEASUREMENT THEORY

### 3.6.1 Introduction to Measurement Theory

Measurement theory allows us to validly define measurements and metrics and to use statistical analysis on our metrics data.

Measurement theory is a branch of applied mathematics. The specific theory we use is called the Representational Theory of Measurement. It formalizes our intuition about the way the world actually works. In this theory, intuition is the starting point for all measurement [1]. Any data manipulation of the measurements must preserve the relationships that we observe between the real-world entities that we are measuring.

Measurement theory allows us to *validly* analyze and manipulate our measurement data.

As an example, consider a customer satisfaction survey. Your users were asked what they thought of your customer support. The possible answers were:

1—Excellent
2—Good
3—Average
4—Inadequate
5—Poor

The result was that you ended up with a score of 3. So, you have average support, and it means you just need to improve a bit, right? Well, maybe. Or maybe not. When you looked at the data, you found out that your scores were 50% Excellent and 50% Poor. No one thought your customer support was average. Measurement theory dicates that taking the average of this kind of data is invalid and can give misleading results.

Measurement theory allows us to use statistics and probability to understand quantitatively the possible variances, ranges, and types of errors in the data.

Assume that you are responsible for estimating the effort and schedule for a new project. You use a prediction model for development effort, and it predicts that, on average, the effort will be 10 staff years with a duration of 1 year. What would you then tell your boss as your estimate? Maybe you would pad it by one staff year, just to be safe. What if you knew that the standard deviation is 2 staff years? A standard deviation of 2 means that 68% of the time, you expect the result to be between 8 and 12 staff years. This means that ~16% of the time it will be over 12 staff years. Now what would you estimate? 12 staff years? 13? Or 10? What if, instead, you knew that the standard deviation was 4 months? Then what would you estimate? 10.5 staff years? (By the way, if we were the boss, we would prefer an answer that included the range with probabilities. We would want the additional information to better trade-off the risks and the rewards.)

### 3.6.2 Measurement Scales

Consider two different types of scales that measure weight: one is a balance scale, the other is a bathroom scale. What is the difference? Which is better?

The balance scale is a relative scale. It compares the weights of objects. The bathroom scale is an absolute scale that gives you an absolute number. Initially, you may think the second scale is better. It does give you an answer. However, as you think more about it, you realize the first may have less error. Bathroom scales frequently are inaccurate. And as you think more and more about it, you probably reach the conclusion that the scales are neither better nor worse, just different ways of measuring.

In measurement theory, we have five types of scales: *nominal*, *ordinal*, *interval*, *ratio*, and *absolute*.

A *nominal scale* is an unordered set of named categories, such that the only comparisons that are valid are "belongs to." For example, you could have a nominal scale for the type of project, and the possible values would be "commercial," "military," or "MIS." The purpose of a nominal scale is to create a mapping from the real world into a set of categories that has some meaning in terms of how you will use it. For example, military projects are consistent in that the number of lines of code produced per staff hour tends to be less than an MIS project. Alternatively, you may wish to count the number of the different types of projects in your data sample. However, with nominal scales, there is no sense of ordering between different types, based on the categories. A project that is "commercial" may be larger or smaller than a "military" project. Using the nominal scale, you do not know how they compare. You need additional relationships and attributes, such as size and complexity, to make comparisons.

The *ordinal scale* is a linearly ordered set of categories, such that comparisons such as "bigger than" or "better than" make sense. An example would be the criticality of trouble reports (TRs). It could have the values of "critical," "major," and "minor." You know that one critical TR is worse than one major TR is worse than one minor TR. However, you do not know if two major TRs are worse than four minor TRs.

The *interval scale* is an ordinal scale with consistent intervals between points on the scale, such that addition and subtraction make sense, but not multiplication and division. Examples are the Fahrenheit and Centigrade temperature scales, where it does make sense to say that if it is 80° in Miami and 10° in Buffalo, then it is 70° warmer in Miami, but it makes no sense to say that it is 8 times hotter in Miami. Interval scales are rarely used in software measurement.

The *ratio scale* is an ordered, interval scale, where the intervals between the points are constant, and all arithmetic operations are valid. With temperature, the traditional example is the Kelvin scale, where there is an absolute 0, and 70°K really is 7 times hotter than at 10°K. With software measurement, examples abound, such as defects per module or lines of code developed per staff month.

The *absolute scale* is a ratio scale that is a count of the number of occurrences, such as the number of errors detected. The only valid values are zero and positive integers.

EXERCISE: Figure 3.5 is a "pain scale," which is a series of six faces, each of which shows progressively more pain and distress. You are at the doctor's office and you are asked to pick the face that best matches how you feel. What kind of scale is the face scale?

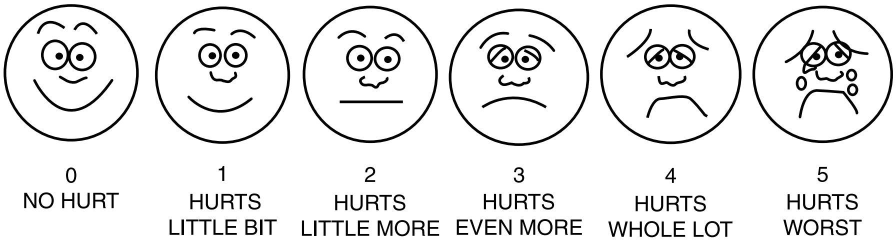

***Figure 3.5.*** *FACES Pain Scale [9].*

*Answer*: Ordinal. It is an ordered scale, but the intervals between items are not necessarily consistent.

EXERCISE: If you have five critical errors, two major errors, and five minor errors, what is the average error?

*Answer*: It does not make sense to have an "average" error in this case. You could talk about the median, or that you have a bimodal distribution, but the "average" error is not major.

Scales can be subjective or objective. Two familiar subjective scales are the Likert Scale and the Verbal Frequency Scale. For example:

Likert Scale: This program is very reliable. Do you

- Strongly agree, agree, neither agree nor disagree, disagree, strongly disagree

Verbal Frequency Scale: How often does this program fail?

- Always, often, sometimes, seldom, never

These subjective scales are ordinal scales; there is an implied order and relationship. Averages or ratios are *not valid* (although frequently used).

### 3.6.3 Measures of Central Tendency and Variability

With measurement, you have data that you need to analyze, understand, and turn into information. The most basic analysis is to understand the data's central tendencies and variability. This section reviews basic statistics that can and should be part of every software engineer's repertoire.

**3.6.3.1 Measures of Central Tendency** The *central tendency* is the middle, or center, of a data set, be it the mean, median, and/or mode. Recall the following definitions:

Mean is the sum of all the occurrences, divided by the number of occurrences.

Median is the middle occurrence in an ordered set.

Mode is the most frequent occurrence.

For the *nominal scale*, only mode makes sense, since there is no ordering.

For *ordinal scales*, mode and median apply, but mean is irrelevant. For example, if you used a Likert Scale (e.g., a 5 point scale ranging from 5 for strongly agree through 1 for strongly disagree) and asked baseball fans to indicate their agreement/disagreement with the statement "I love the New York Yankees," you might get a distribution that had 40% strongly agreeing, 40% strongly disagreeing, and the remaining 20% evenly spread throughout the other three categories. It would be meaningless (and wrong) to then conclude that the average fan's response was neutral, even though the mean was 3. The most meaningful statements you could make would speak to the bimodal distribution.

For *interval*, *ratio*, and *absolute scales*, mean, mode, and median are all meaningful and relevant.

**3.6.3.2 Measures of Variability** The measures of central tendency indicate the center of the data. Not only do we need to describe the sameness, but we also need to describe the differences and variability.

The standard and simplest measures of variability are *range*, *deviation*, *variance*, *standard deviation*, and *index of variation*. Even without a statistics background, these measures are simple and useful enough that they can and should be part of every software engineer's knowledge base.

- *Range*: The range of values for the mapping of the data that is calculated by subtracting the smallest from the largest. For example, assume you have three modules—A, B, and C—which have sizes of 10 KLOC, 24 KLOC, and 50 KLOC, respectively. Then the range of KLOC per module would be from 10 to 50 KLOC or 40 KLOC.
- *Deviation*: The distance away from the mean of the measurement. In our example, the average size of a module = 84/3 = 28 KLOC. Then the deviations for our modules A, B, and C are 18, 4, and 22, respectively.
- *Variance*: A measurement of spread, which is calculated differently if it is for a full population or a sample of a population.

  For a full population, where $N$ is the number of data points,
  Variance = $\Sigma(\text{Deviations}^2)/N$.

For a sample population,
Variance = $\Sigma(\text{Deviations}^2)/(N-1)$.

In our example, the variance (module size) = $(324 + 16 + 484)/3 = 275$. To pick the right SD formula, you need to decide if your data is a complete population or a sample of that population. In this example, we assume that the data is for a complete population, that is, there are only three modules to consider. If instead the three were "a representative sample" of a larger set of modules, then we would divide by two rather than three.

- *Standard Deviation (SD)*: The "typical" distance from the mean. SD = $\sqrt{\text{variance}}$. In our example, the standard deviation = 16.6.
- *Index of Variation (IV)*: An index that indicates the reliability of the measurement. IV = SD/mean. In our example, IV = 16.6/28 = 0.59. This metric normalizes standard deviation by dividing it by the mean. A SD of 1 with a mean of 250 is entirely different from a SD of 1 with a mean of 2. The lower the index of variation, the less the variation.

Figure 3.6 illustrates lower and higher variance. With low variance, the values cluster around the mean. With higher variance, they spread farther out.

Standard deviation is the classic measurement of variation. As the variance increases, so does the standard deviation. It has additional significance for normal (e.g., bell-shaped) distributions, in that the standard deviation intervals define a certain distribution. The interval contained within 1 standard deviation (SD or $\sigma$) of the mean is ~68% of the population. The interval within 2 SD is ~95% of the population, within 3 SD is ~99.73% of the population, and within 6 SD is ~99.9999988%. The standard deviation percentages (68%, 95%, 99.7%) are numbers you should know.

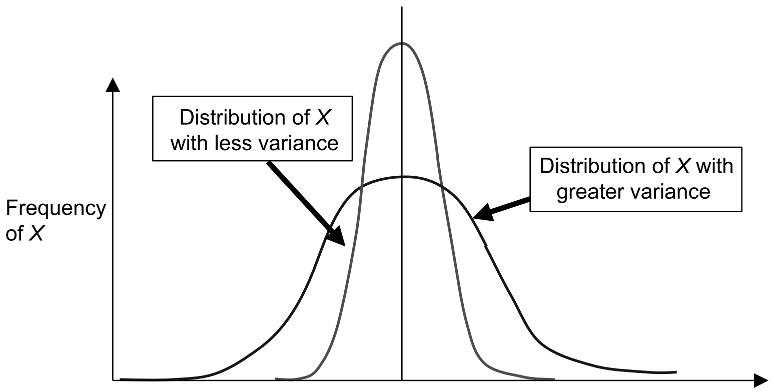

***Figure 3.6.*** *Examples of variance.*

### 3.6.4 Validity and Reliability of Measurement

If you had to choose between a valid measurement and a reliable measurement, which would you choose? Which *should* you choose? And what exactly is the difference?

The question of valid and reliable refers to the transformation of the abstract concept of a metric into the operational definition (recall the meta-model from Section 3.4). Is the operational definition a valid and/or reliable transformation of what we want to measure?

Valid measurements measure *what* we intend to measure. Completely valid measurements for complex concepts such as quality, productivity, and complexity are difficult, if not impossible, due to the difficulty of creating an operational definition that matches the breadth of the concept. If you define your quality metric to be the total number of defects per KLOC (i.e., the defect density) of a module, then how do you account for factors such as ease of use and elegance? If you define productivity as "the number of tested lines of code produced per day," how do you compare the productivity of one person who can code a feature in one line versus a person who codes the same feature in 100 lines?

A reliable measure is one that is consistent. Assume that you have a tool that counts defects, but it only counts the defects discovered during system and acceptance testing. This is a reliable measure. The tool will consistently count the defects. It is not totally valid as a count of all defects in a project, but it is reliable and useful. Another example is a watch. Some people set their watch five minutes ahead of time. This is a reliable measure of time, but not valid.

The index of variation is one measure of the reliability of a measurement: the smaller the IV, the more reliable the metric.

Theoretically, there are three types of metric validity [1]: construct, criterion-related, and content.

*Construct validity* refers to whether the metric is constructed correctly. For example, an invalid construct would be one that uses the mean as a measure of central tendency for a metric that has an ordinal scale. So far within this chapter, we have been discussing construct validity.

*Criterion-related validity* refers to the ability to use the operational definition to predict the behavior of the abstract concept, which is extremely important. Predictive measures consist of both mathematical models and predictive procedures. For example:

- Function points can predict lines of code (LOC) (based on language).
- Lines of code can predict effort, where the mathematical model is Effort $= A * \mathrm{LOC}^B + C$. The predictive procedures are the method for determining $A$, $B$, and $C$.

Predictive procedures need to be validated to understand their inherent accuracy. The validation is done by comparing empirical data with the outcomes predicted by the predictive models.

*Content validity* refers to the ability of the operational definition and measurement method to capture the full range of meanings associated with the abstract concept. Productivity measures based on functionality produced rather than lines of code produced have higher content validity.

There is a natural conflict between reliability and validity in measurement. Reliable measurements tend to be tightly and narrowly defined. Valid measurements tend to be broadly defined and composites of other metrics (such as a quality metric defined as a function of defect, ease-of-use, and ease-of-maintenance measures).

The traditional belief is that valid metrics are somehow better than reliable ones, if you have to choose. Both do serve their purposes, and in many cases, the reliability of a measurement that is invalid (think of the scale that always underweighs by 10 pounds) can be attenuated (by always adding 10 pounds to the result).

### 3.6.5 Measurement Error

What kind of errors might you make with the kind of foot measuring stick [10] shown in Figure 3.7? You might measure the wrong part of a foot and always get the wrong measurement. Or you might make an error in reading the measurement, reading a little too high or a little too low.

***Figure 3.7.*** *Foot measuring stick [10].*

The first type of error is a *systematic* error, meaning it is an error in the measurement system. It will show up every time. It affects the validity of the measurement. It is the "bias" with a measurement. The second type of error is a *random* error, meaning it is an error in the measurement itself, which will appear "at random." It affects the reliability of the measurement. It is the "noise" within a measurement.

In software development, if you were recording and calculating defects found during code inspections, a systematic error would occur if no one compared the code to the requirements, so those defects were never found or counted. Random errors would be errors that the inspector made in recording the number and types of defects found.

Figure 3.8 is a graph of random error. Random errors increase the variance but do not change the mean.

Figure 3.9 is a graph of systematic error. Systematic errors change the mean but not the variance.

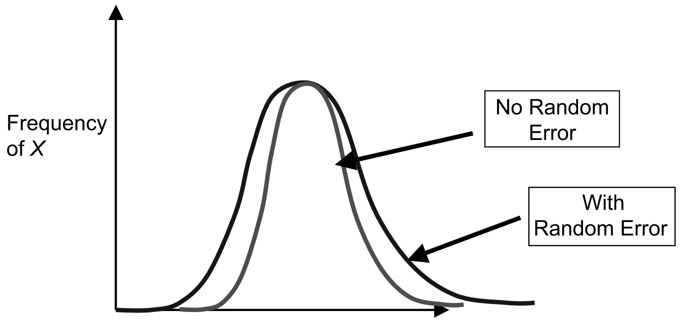

***Figure 3.8.*** *Distributions of X with and without random error.*

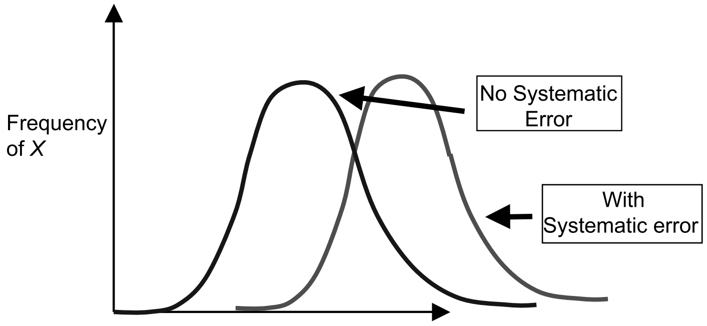

***Figure 3.9.*** *Distributions of X with and without systematic error.*

Valid measurements can have random errors. Reliable measurements can have systematic errors.

You can reduce and manage measurement errors by:

1. Attenuating the measurement for systematic errors. Once you realize Mary always underestimates by 25%, increase her estimates.
2. Test pilot your measurements, looking for both systematic errors and sources of random errors. If you know that 10 defects were found in testing the last week, and the official measurements only show 8, find out what went wrong.
3. *Triangulate* your metrics. Use different measurements and measurement processes without the same systematic errors. All effort estimation methods have high variance. Use at least three methods, and look at the mean and standard deviations of the results.
4. Use statistics to determine and adjust for random errors.

Figure 3.10 is one final look at measurement error. If reality is the vertical line on the left, then the distance from the actual mean to reality is the bias, or systematic error. The clustering/variation around the mean is the random error.

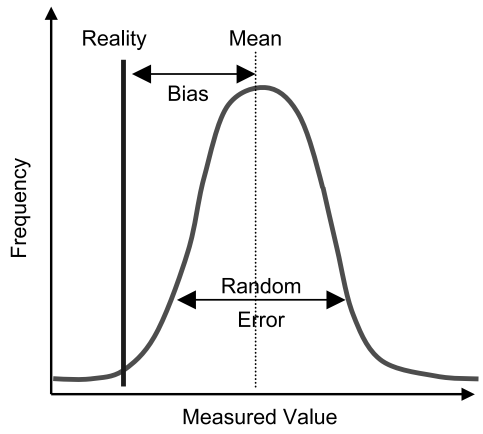

*Figure 3.10.* Measurement error.

## 3.7 ACCURACY VERSUS PRECISION AND THE LIMITS OF SOFTWARE METRICS

*It is the mark of an instructed mind to rest satisfied with the degree of precision which the nature of a subject admits, and not to seek exactness when only an approximation of the truth is possible*

—Aristotle, 330 B.C.

The accuracy of a measurement is the degree to which a measurement reflects reality. Precision of a measurement represents the size of differentiation possible with a measurement. In software engineering, we measure our processes and our products, and use those measures to control, predict, monitor, and understand. Many of these measures have inherent variations, making highly precise measurements impossible or irrelevant. Consider counting the size of the code. There isn't one standard way of counting, which would make it easy to compare your size measurements with another company's.

Estimation is not precise by its very nature, and software estimation is no exception. You will find that size estimates are considered excellent if they are within 10%. This lack of precision means that reporting an estimate to be 1253 LOC is misleading. It implies that you actually believe this estimate is a better estimate than 1254 LOC, which is highly doubtful. Instead, if your accuracy is $\pm 10\%$, then report the estimate to be 1250 LOC. Better yet, estimate 1250 LOC $\pm 10\%$, which is more accurate, although less precise. Also, recall and use what you learned about significant digits in grade school. If you are calculating an indirect metric, it should not have more significant digits than the factors going into it.

## 3.8 SUMMARY

Measurement can be a challenge, especially in software, due to its abstract nature. An entity and the measurement of it are not equivalent. For example, aspects of the quality of software can be measured by using defect data, or defect data and performance data, but these are only certain aspects, not the whole story.

For each concept we want to measure, such as quality or complexity, we need to decide which aspects are the most relevant and important to us and use those aspects to create a model of the concept. There are different techniques to take these abstract concepts and create real-world metrics that measure them. The underlying key is *models*—you need to have models to allow you to create the specific definitions of abstract concepts.

Measurement theory and statistics are tools that allow us to validly analyze and compare the metrics data. This chapter contains an introduction to both, including measurement scales, measures of central tendencies and variation, and measurement error. These concepts are the basic building blocks of measurement and metrics.

## PROBLEMS

**3.1** If the definition of quality was the defect density of a software release, what would be (a) an operational definition and (b) a measurement in the real world?

**3.2** You decide to measure productivity by lines of code produced per month, but you do not count comments. What do you think will happen?

**3.3** True or False: You can measure anything in the material world.

**3.4** True or False: Models are nice to have but not crucial to software measurement.

**3.5** True or False: Algorithmic models are preferred because they are the most precise.

**3.6** Suppose we take a sample of the number of defects in different modules. We get the following data: 10, 20, 15, 18, and 22. Calculate the mean, SD, and IV for the number of defects per module.

**3.7** True or False: The mean is a valid measure of central tendencies for a metric using the ordinal scale.

**3.8** You are trying to understand customer satisfaction with your reliability, so you survey and ask the following: How satisfied are you with the reliability of the release?

1 = Extremely
2 = Very Satisfied

3 = Average Satisfaction
4 = Not Satisfied
5 = Very Disappointed

Three of your customers chose #1, two chose #2, and one chose #5. What would be the valid way to report the central tendency of the answers to this survey question?

**3.9** True or False: A higher IV indicates a more reliable metric.

**3.10** True or False: If there is random error in a measurement, the standard deviation will be greater than that of the true values being measured.

**3.11** True or False: Precision is more important than accuracy in a metric.

**3.12** Suppose I want to evaluate which programmer is the best programmer. I've decided that I will look at two criteria—quality and productivity. Quality is defined as the number of bugs found per month and productivity is defined as the number of lines of code written per month. What conclusions can I draw from the following table?

| Programmer | Quality | Productivity |
|---|---|---|
| A | 2 | 2500 |
| B | 5 | 500 |
| C | 25 | 200 |
| D | 35 | 1000 |

**3.13** I also define a Good Programmer Index $(n)$ = Productivity $(n)$ / Quality $(n)$. What conclusions can I now draw from the previous data? What is the scale of this index? Is it a reliable measure? Why or why not?

**3.14** True or False: For a normal distribution, one $\sigma$ (one standard deviation, above and below the mean) includes $\sim$80% of the population.

**3.15** Your officemate decides to measure his code quality by the percentage of comments in his code. You tell him it is an invalid metric. To which *type* of metric (in)validity are you referring: construct, criterion, or content? Is this metric reliable?

**3.16** What is a metric which has predictive procedures, other than effort estimation? How would you validate it?

**3.17** Why do you think variance is defined differently for samples of populations compared to complete populations?

## PROJECTS

**3.1** Consider the concept of productivity. Define it with (a) a text model, (b) a diagrammatic model, and (c) an algorithmic model.

**3.2** Define your own metric for programmer productivity. Then evaluate it. Is it valid? Reliable? Precise? What type of systematic errors would you expect to encounter? What type of random errors do you expect? How difficult will it be to obtain the data?

**3.3** If you work on a project, find out how quality is measured. Evaluate the metric for reliability, validity, and cost.

## REFERENCES

[1] N. Fenton and S. Pfleeger, *Software Metrics*, 2nd ed., PWS Publishing Company, Boston, 1997.

[2] D. Pitts, "Why is software measurement hard?" [online] 1999. Available from http://www.stickyminds.com. Accessed Jan. 6, 2005.

[3] F.P. Brooks, *The Mythical Man Month*, Addison-Wesley, Reading, Mass., 1974.

[4] G. Weinberg, *Quality Software Management*, *Volume 2: First-Order Measurement*, Dorset House Publishing, New York, 1993.

[5] P. Senge, *The Fifth Discipline*, Doubleday, New York, 1990.

[6] A.W. Crosby, *The Measure of Reality*, Cambridge University Press, Cambridge, United Kingdom, 1997.

[7] S. Kan, *Metrics and Models in Software Quality Engineering*, 2nd ed., Addison-Wesley, Boston, 2003.

[8] Dilbert newsletter. Available from http://www.unitedmedia.com/comics/dilbert/dnrc/html/newsletter44.html. Accessed Jan. 6, 2005.

[9] M.J. Hockenberry, D. Wilson, and M.L. Winkelstein, *Wong's Essentials of Pediatric Nursing*, 7th ed., Mosby, St. Louis, 2005.

[10] Photograph available from http://www.howcool.com/pleasersshoes/Measuring-Stick.jpg. Accessed Jan. 10, 2005.