Cem Kaner on Rethinking Software Metrics

Evaluating measurement schemes

by Cem Kaner, J.D., Ph.D.

▶▶ QUICK LOOK

- Nine factors to assess a measurement scheme
- An examination of three common measures

Perhaps the most respected book on measurement in computing, Fenton and Pfleeger's *Software Metrics: A Rigorous and Practical Approach*, defines measurement as follows:

"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."

The problem with this definition is that there are lots of available clear-cut rules. We have to select *the right* clearly defined rule. Otherwise we could measure "goodness of testers" by the clearly defined rule called "counting their bug reports." This (as we'll see in a few moments) is ridiculous.

I prefer the following definition:

Measurement is the assignment of numbers to attributes of objects or events according to a rule derived from a model or theory.

Fenton and Pfleeger do point out the issue of the need for a model. They discuss this as an issue of the definition of the attribute. All that I'm doing here is making the issue much more visible.

The invisibility of underlying measurement models has led people to use inadequate and inappropriate "metrics," deluding themselves and wreaking havoc on their staffs. For a good read on this as a general problem, read R.D. Austin's 1996 book, *Measuring and Managing Performance in Organizations*—or, for that matter, Scott Adams' comic strip *Dilbert*.

Building a Theory of Measurement

Measurement theory addresses problems that run through many disciplines, including Computing. I learned about the theory of measurement primarily from Steve Link and A.B. Kristofferson, when I did my doctoral studies in psychophysics (also known as perceptual measurement). (For a thoughtful history of that field, read Link's 1992 book, *The Wave Theory of Difference and Similarity*.)

This article is a preliminary report of my attempts to pull together thinking from several disciplines into a more coherent, and I think more practical, approach to software-related measurement. My goal is to help you evaluate measurement schemes that people ask you to use, to help you explain why the bad ones shouldn't be imposed on your group, and to help you develop more useful alternatives.

In summary, I think that the theory underlying a measurement must take into account at least nine factors. This article defines these nine factors and applies them to a few examples.

The first five are intuitive:

1. **The attribute to be measured.** This is what you want to measure, such as the extent of testing that you've done, the complexity of the program you're testing, or the effectiveness of a tester.

2. **The instrument that measures the attribute.** Think of a measure as a reading from an instrument. For example, a stopwatch shows how much time has passed. Some people try to measure extent of testing with a coverage program, software complexity with a complexity program, or tester effectiveness with bug counts.

3. **The relationship between the attribute and the instrument.** What is your basis for saying that this instrument measures this attribute well? Here's a question that I find useful: Suppose that we increase the attribute by 20%. What do we expect to happen to the instrument, and why? How confident are we that the instrument reading (the measured value) will go up? What mechanism relates the attribute to the instrument?

4. **The probable side effects of using this instrument to measure this attribute.** When people realize that you are measuring something, how will they change their behavior to make the numbers look better? If you find a way to change the measurement by 20%, what has happened to the attribute? Some side effects are so bad that the result of trying to make the number bigger is to drive the attribute lower (for example, as discussed further below, driving people to increase bug count might increase the number of bugs reported—but make the testers much less effective).

When people realize that you are measuring something, how will they change their behavior to make the numbers look better? If you find a way to change the measurement by 20%, what has happened to the attribute?

5. **The scope of the measurement,** probably defined in terms of what this measurement will be used for.

The next four factors are more technical but are essential for an understanding of the attribute, the instrument, and their relationship:

6. **The appropriate scale for the attribute.** We'll discuss scale types in a moment.

7. **The variation of the attribute.** The attribute itself is probably subject to random fluctuations. You might be a more effective tester on Tuesday than on Wednesday. Therefore we need a theory of variation of the attribute.

8. **The scale of the instrument**. To be discussed shortly.

9. **The variation of measurements made with this instrument.** The act of taking measurements, using the instrument, carries random fluctuations. Thus, we need a theory of measurement error, or of variation associated with using and reading the instrument.

Example: Using a Ruler

Let's start with an example of the simplest case, measuring the length of a table with a one-foot ruler.

- **Attribute** The attribute of interest is the length of the table.

- **Instrument** The ruler is the measuring instrument. It's one foot long, so we'll have to lay it down six times to mark off a six-foot length.

- **Theory of Relationship** The relationship between the attribute and the instrument is direct. They're both on the same scale and (except for random error) a change in the attribute results in a directly comparable change in the measured value. (Example: Cut the table down to four feet and the next time you measure it, the ruler measurement will read four feet.)

- **Probable Side Effects** I don't anticipate any side effects of using a ruler to measure the length of the table.

- **Attribute's Scale** We'll have more to say about scaling soon. For now, note that the length of the table can be measured on a ratio scale. A table that is six feet long is twice as long as one that is three feet long. If we doubled the lengths, the twelve-foot table would still be twice as long as the six-foot table. This preservation of the ratio relationship ("twice as long") when both items are multiplied is at the essence of the ratio scale.

- **Attribute's Variation** If you measure the lengths of a few tables, you'll get a few slightly different measurements because there is some (not much, but a little) manufacturing variability in the lengths of the tables. Tables that are supposed to be six feet long might actually vary between 5.98 and 6.02 feet. The length of individual tables might not vary much, but if you went to an office supply store and asked for a six-foot table, the table that you would get would only approximate (perhaps very closely approximate) six feet.

- **Instrument's Scale** The instrument (ruler) measures length on a ratio scale.

- **Instrument's Variation** Try to measure a six-foot table with a one-foot ruler ten times. Record your result as precisely as you can. You'll probably get ten slightly different measurements, probably none of them exactly 6.00 feet. Even with ruler-based measurement, we have error and variability.

Example: The Scaling Problem

Suppose that Sandy, Joe, and Susan run in a race. Sandy comes in first, Joe comes in second, and Susan third. The race comes with prize money. Sandy gets $10,000, Joe gets $1000, and Susan gets $100.

- **Attribute** Let's say that our attribute of interest is the speed of the runners.

- **Instrument** The instrument is the simple observation of the order of the runners as they cross the finish line. There are two different sets of markers on this instrument (like the inch and centimeter markers on rulers). One set of markers says First, Second, and Third. The other markers are prize money amounts, say $10,000, $1000, and $100. You might prefer to measure speed with some *other* instrument, but (oops) you forgot your stopwatch and this is what you've got. (If this example looks oversimplified, please be patient with it; I'm trying to use something everyone understands in order to illustrate the problem that sometimes we are stuck with crude measuring instruments.)

- **Theory of Relationship** The mechanism underlying the relationship between speed and position in the race is straightforward. Faster speed results in a better position (First, Second, Third).

- **Probable Side Effects** I don't see obvious side effects (changing how people run races) in this particular case.

- **Attribute's Scale** The attribute's scale is probably best expressed in terms of miles (or meters) per hour. If so, this is a ratio scale (one mile per four minutes equals fifteen miles per hour).

- **Attribute's Variation** The race only gave us one sample of the speed of these runners. If they ran the same track and distance again tomorrow, they'd probably have different times.

- **Instrument's Scale** This instrument operates on an ordinal scale—not a ratio scale, and not an interval scale. (For detailed discussions of scale types, see S.S. Stevens' 1976 book *Psychophysics*, and Fenton and Pfleeger's *Software Metrics*.) The following comparisons should give you a sense of the differences among the scales.

If we were operating on a **ratio scale**, we would be able to say that Susan (measured as $100) was 1/100th as fast as Sandy (measured as $10,000).

If we were operating on an **interval scale**, we could say that the difference between Susan and Joe (3–2=1) was the same as the difference between Joe and Sandy (2–1=1) and that the difference between Susan and Sandy (3–1=2) was twice the difference between Susan and Joe.

On an **ordinal (or positional) scale**, all we know is that Sandy crossed the line first, Joe crossed it second, and Susan crossed it third. We don't know how much faster Joe was than Susan, but we do know that he was not as fast as Sandy.

The attribute itself is probably subject to random fluctuations. You might be a more effective tester on Tuesday than on Wednesday. Therefore we need a theory of variation of the attribute.

The ordinal scale doesn't tell us much about speed but it tells us more than a **nominal (or categorical) scale**. It would stretch the example too far to try to talk about a nominal scale for speed. But since we've been comparing the different scale types here, let's say a few words about nominal scales as well. Suppose that you had hundreds of different bug reports, and you sorted them into categories (this is a printing error, that is a usability error, etc.). We can assign numbers to the categories (printing = type 1; usability = type 2), but even though we might assign these numbers to the reports according to well-considered rules, the numbers are just labels.

The final scale to mention is the **absolute scale**. If you have one (1) pen, you have one (1) pen. If you cut it in half, you get a mess; not two halves of a working pen.

- **Instrument's Variation** There is probably not much measurement error in this example, unless the race was very close.

Measurement in the Real World

The examples of *length* and *position in the race* are toys. They are easy to figure out. The theories of relationship are clear-cut and the side effects are minimal.

When it comes to things that we would *really* like to measure, life is more difficult. Examples of the kinds of things that testers are routinely asked about include:

- **Productivity** Who's doing a better job of testing or programming (or whatever)?
- **Extent** How much testing has been done?
- **Progress** Where are we relative to some plan?
- **Reliability** What is the actual and probable future rate of failure?
- **Usability** What, for example, is the probable user error rate?
- **Support Burden** What will this cost to support?

How do we measure these? Each one involves complex issues. Typically, they involve a lot of judgment (which is subjective). Additionally, several of the most interesting dimensions involve human behavior. That's hardly a surprise—we are working in a field of human endeavor, called computing, whose essential work product is the stuff of mental creation. The essence of "quality" is "qualitative." As Gerald Weinberg wrote in the 1993 book *Quality Software Management*, "Quality is value to some person."

There is a bias in computing against measurements that involve subjective quantities. Somehow, some people have developed the idea that subjective issues are immeasurable and unscientific. (This article isn't the place to refute that directly; but if you're of that view, go read Link's wave theory book.)

Tom DeMarco provided an example of this bias almost twenty years ago. Although he has taken a different approach in his more recent book, *Why Does Software Cost So Much?*, his original presentation in 1982's *Controlling Software Projects* is a well-written, still influential example. DeMarco asked how to measure customer interface complexity, and provided an example of how *not* to do it—the development team were asked to rate the customer interface complexity of their own projects as normal, greater than normal, or less than normal. He pointed out some biases associated with using developers to rate their own code, and then concluded that "any exercise that tries to give a numeric value to an unquantum without doing any real measurement along the way is a bit of a fraud." An "unquantum" is, according to DeMarco, "a relevant factor that is unmeasured." Evidently, rankings by humans don't count as measurements. Instead, he said that measures like the following were "true metrics."

- **Change Rate:** customer-introduced changes per unit time
- **Change Impact:** unit cost of average change
- **Customer Dissatisfaction:** cost of change during a fixed period

I accept DeMarco's opinion that the developers' rankings of their own work are unusably biased, but to say on the basis of this that human ranking of complexity is some kind of unquantum because it isn't expressed in easy-to-count numbers that are much less directly related to the value we want to measure (that is, the complexity of the thing to humans) seems…well, I guess we just disagree [on that 1982 conclusion]. I think it would be interesting to ask customers who interacted with the system to rate the different areas' customer interface complexity. The fact that there are lots of ways to do this badly doesn't create an excuse for walking away from a fundamental point: that if you want to talk about the complexity of a human-machine interface, the human's sense of that complexity *is* a key measure—perhaps the most direct and the important measure—of that complexity.

Software-related attributes often involve psychological or subjective components. Our measurements of them are questionable when they fail to take these factors into account.

Let's consider three common attempts to develop software metrics:

Example: Bug Counts and the Theory of Relationship

Should we measure the quality (productivity, efficiency, skill, etc.) of testers by counting how many bugs they find? Leading books on software measurement suggest that we compute "average reported defects/working day" and "tester efficiency" as "number of faults found per KLOC" or "defects found per hour." [See complete references at the end of this article, specifically Grady et al. 1987, Fenton et al. 1997, and Fenton et al. 1994.] These authors are referring to averages, not measures of individual performance, and they sometimes warn against individual results (because they might be unfair). However, I repeatedly run into managers (or, at least, the test managers who work for them) who compute these numbers and take them into account for decisions about raises, promotions, and layoffs. For that matter, are these even valid measures of the efficiency of the group as a whole?

Let's do the analysis and see the problems with this measure:

- **Attribute** The attribute of interest is the goodness (skill, quality, effectiveness, efficiency, productivity) of the tester.
- **Attribute's Scale** I don't know. Neither do you.
- **Attribute's Variation** I don't know. But there is variation. Joe probably does better work on some days than others.
- **Instrument** We don't have an obvious, easy-to-use, unambiguous direct measure of tester effectiveness, skill, etc. So instead we use a surrogate measure, something that is easy to count and that seems self-evidently related to the attribute of interest. In this case, the instrument is a counter of bug reports.
- **Instrument's Scale** Bug counts are an absolute scale (two half-reports do not equal one whole-report).
- **Instrument's Variation** There's not much random variation in the counting of bug reports (although there is some, such as bugs classified as duplicates). There *is* systematic measurement-related variation, as we'll see in the discussion of side effects.
- **Theory of Relationship** There is hardly any theory of relationship between the attribute and the instrument here. We count bugs because they are easy to count, not because this is an essential measure of the worth of the tester. Yes, testers should search for bugs. But when you increase a tester's true effectiveness by 20%, you might get someone who goes after harder-to-find bugs, takes on code that is more stable and has problems that are much more subtle, or who writes better test documentation, mentors the rest of the staff, spends more time building tools, or who does other great stuff that never reflects directly on her personal bug count. Bug counts would seriously mismeasure such testers.
- **Probable Side Effects** People are good at tailoring their behavior to whatever they're being measured against. If you ask a tester for more bugs, as Weinberg and Schulman have pointed out, you'll probably get more bugs. Probably you'll get more bugs that are minor, or similar to already reported bugs, or design quibbles—more chaff. But the bug count will go up. In general, this measurement system creates incentives for activities that generate certain results (lots of bugs) and disincentives for anything else. The predictable result is dysfunctional—people changing their behavior in ways that bring up their bug counts, at the expense of unmeasured variables that might have much more to do with the genuine effectiveness of a tester in a group. (Austin talks about side effects—or, as he calls it, "dysfunction"—in detail in his 1996 book.)

Problems like these have caused several measurement advocates (specifically Grady and Caswell, and Austin) to warn against measurement of attributes of individuals unless, as DeMarco suggested in 1995, the measurement is being done for the benefit of the individual (for genuine coaching or for discovery of trends) and otherwise kept private.

With only a weak theory of relationship between bug counts and tester goodness, and serious probable side effects, we should not use this measure (instrument).

Example: Code Coverage

Suppose that you want to know how much testing has been done. How would you measure that?

One approach is to compute "code coverage." The most common definition of coverage involves the percentage of statements tested, or the percentage of statements plus branches tested. Supposedly, a higher percentage means more testing. Some people (vendors included) go further and foolishly say that 100% coverage means complete, or sufficient, testing.

There are several other types of coverage beyond statement and branch coverage (some examples are described in my 1995 article, listed in the complete references at the end of this feature). Each of these involves measuring the percentage of a certain type of test that you have run, or a certain level of thoroughness of checking for a specific type of error. We are never using the population of all possible tests of a product as our baseline when we compute code coverage—if we were, coverage would always be 0.00%, a rather boring number. But because we are not accounting for all possible tests, we can have a 100% covered product that still has undiscovered defects.

- **Attribute** Extent of testing completed.

- **Instrument** Number of tests run of a certain type, number of lines touched by the tests, etc. Depends on the definition of coverage.

- **Theory of Relationship** We could represent the possible bugs in a product in terms of a disjoint collection of sets: E1, E2, E3, and so on through EN. I don't know how big N is. E1 is the set of all errors of type 1 (such as failure to initialize a variable). E2 is the set of all errors of type 2 (such as a syntax error in a line of code). A coverage measure tells you that you have made great progress against certain types of errors—perhaps at 100% coverage, you have tested for all possible errors of types E2, E3, and E4. As a normal part of the process, you will stumble over some other types of errors, so coverage-driven testing is broader than just the collection of errors that the tool is focused on. But still, suppose that we increase the extent of testing by running a bunch of tests that this particular coverage tool is insensitive to. For example, you can run through every line of code (statement coverage) without ever testing a boundary value. Go back to test all boundary values and you might find new bugs, but you won't increase statement coverage at all.

There is hardly any theory of relationship between the attribute and the instrument… We count bugs because they are easy to count, not because this is an essential measure of the worth of the tester.

- **Probable Side Effects** Brian Marick has repeatedly pointed out the side effects of driving testing by using coverage metrics. People focus their efforts on the types of tests that will drive up the percentages, and they tend to stop when they have reached the target percentage (85% "coverage" might be good enough in some companies, 95% the target in others.) The result is that they stop testing when they have found most of *some* types of errors, without necessarily realizing that they have no assurance that they have found most of several other types of errors.

- **Scope** Do these side effect problems mean that we should not use coverage tools? No, of course not. It depends on what we use coverage for. I think it is very useful to evaluate coverage in order to discover that some code is completely untested, but that it is also quite dangerous to use a coverage "metric" as an indicator of how close to completion you are.

In sum, our measures of the extent of testing—like so many measures that we take in software—are numeric. This might make them look more "scientific," but they are fundamentally judgment-driven. A theory of testing and of testing adequacy is embedded (often hidden) in such measures.

Example: Code Complexity

McCabe's complexity metric is often enough criticized as incomplete (see, for example, Fenton and Pfleeger in *Software Metrics*). But let's apply our model to this metric.

- **Attribute** Let me suggest that "complexity" is a fundamentally psychological concept. If the term means anything, it deals with how complex the software is to a human. Indeed, the complexity metrics are sometimes explicitly referred to as measuring "psychological complexity."

- **Instrument** A count of the number of number of nodes on the graph, essentially of the branches in the program, certainly looks at one aspect of complexity. It *is* easy to count…but does this counting give us a true measure of complexity?

- **Theory of Relationship** We can easily drive up true complexity (for example, replacing all the variable names with random numbers) without affecting the branch-driven complexity measure at all. The relationship is weak and without much, if any, theoretical basis.

- **Probable Side Effects** Distortions may occur in the code as we pay tremendous attention to a counter of paths—and less attention to the unmeasured complexity issues like clarity of expression, variable naming conventions, algorithmic sanity, and operating environment.

In Sum

In his 1993 book, *Making Software Measurement Work: Building an*

Effective Measurement Model, Bill Hetzel pointed out that

> "Practitioner attitudes [toward measurement] tend to range from barely neutral to outright antagonistic. It is rare to find the practitioner who really thinks of measurement as a useful and indispensable tool for good software work. Most feel that they get back very little from the measurement activity. . . The psychological dislike and distrust our practitioners have about measurement is a significant challenge facing us. From my perspective, we've been pretty unsuccessful in serving working engineers and practitioners."

Hetzel suggests an alternative, bottom-up approach to measurement that is worth looking into. He wants to use metrics to stimulate questions, to explore the engineering activity, rather than to use them to immediately focus on setting targets and goals to control engineering.

The approach that I'm suggesting isn't incompatible with Hetzel's. Or with Fenton and Pfleeger's, or many other common approaches to software metrics. But what I'm proposing here is more explicit about some issues that we too often approach too casually.

Measures are not made acceptable simply because they are easy to compute and seem relevant. They are not valuable merely because they have something to do with the latest goal-of-the-week. They *work* when they actually relate to something we care about, and when the risks associated with taking the measures (the probable side effects), in the context of the scope of use of those measures, are insignificant compared to the value of information we actually obtain from them. To understand that value, we must understand the underlying relationship between the measure and the attribute measured.

This material was first publicly presented at the Pacific Northwest Quality Conference in October, 1999. This model was reviewed and extended at the Eighth Los Altos Workshop on Software Testing in December, 1999. I thank the LAWST attendees, Chris Agruss, James Bach, Jaya Carl, Rocky Grober, Payson Hall, Elisabeth Hendrickson, Doug Hoffman, Bob Johnson, Mark Johnson, Brian Lawrence, Brian Marick, Hung Quoc Nguyen, Bret Pettichord, Melora Svoboda, and Scott Vernon, for their critical analyses. STQE

Cem Kaner, Ph.D., J.D., is the senior author of Testing Computer Software and of Bad Software: What to Do When Software Fails. He consults and teaches courses on software testing and practices law, focusing on the law of software quality. Contact him at kaner@kaner.com, www.kaner.com, or www.badsoftware.com.

Editors Note: Because this article introduces a new theory into the body of industry literature, we have included its complete list of references.

References

Austin, R.D. *Measuring and Managing Performance in Organizations.* Dorset House, 1996.

DeMarco, T. *Controlling Software Projects: Management, Measurement, and Estimation.* Dorset House, 1982 (p. 51).

DeMarco, T. "Mad about measurement" in *Why Does Software Cost So Much?* Dorset House, 1995.

Fenton, N.E., Pfleeger, S.L., and Glass, R.L. "Science and Substance: A Challenge to Software Engineers," *IEEE Software* (July 1994).

Fenton, N.E. and Pfleeger, S.L. *Software Metrics: A Rigorous and Practical Approach,* 2nd ed. PWS Publishing, 1997 (pp. 5, 36, 39).

Grady, R.B. and Caswell, D.L. *Software Metrics: Establishing a Company-Wide Program.* PTR Prentice-Hall, 1987 (p. 227).

Hetzel, B. *Making Software Measurement Work: Building an Effective Measurement Model.* QED Publishing Group, 1993 (p. 20).

Johnson, M.A. "Effective and Appropriate Use of Controlled Experimentation in Software Development Research" (master's thesis in computer science, Portland State University, 1996).

Kaner, C. "Software Negligence and Testing Coverage," *Software QA Quarterly* 2, no. 2 (1995): 18 [available at www.kaner.com/coverage.htm].

Link, S.W. *The Wave Theory of Difference and Similarity.* Lawrence Erlbaum, 1992.

Marick, B. "How to Misuse Code Coverage" (conference paper, 1999) [available at ftp://ftp.rstcorp.com/pub/papers/coverage.pdf].

Stevens, S.S. *Psychophysics.* Wiley, 1975.

Weinberg, G.M. *Quality Software Management, Volume 2, First-Order Measurement.* Dorset House, 1993 (p. 108).

Weinberg, G.M. and Schulman, E.L. "Goals and performance in computer programming," *Human Factors* 16, no. 1 (1974) (pp. 70-77).