robert v. binder
rbsc

Can a Manufacturing Quality Model Work for Software?

New views of mature ideas on software quality and productivity.

As software engineers, we often look to hardware analogies to suggest techniques that are useful in building, maintaining, or evaluating our software systems. In this issue, Bob Binder explains why the six-sigma approach to hardware quality simply does not work when applied to software.

—Shari Lawrence Pfleeger

SIX SIGMA IS A PARAMETER USED IN STAtistical models of manufactured goods' quality. It also serves as a slogan that suggests high quality. Some attempts have been made to apply 6 sigma to software quality measurement. Here, I explain what 6 sigma means and why it is inappropriate for measuring software quality.

SIX SIGMA DEFINED. Six sigma means six standard deviations. A standard deviation is a parameter that characterizes a set of measurements, just as the average can characterize such a set. One standard deviation is a value such that roughly two thirds of all values in a set fall within the range from one standard deviation below average to one standard deviation above average. Sets of values that can be characterized by the average and standard deviation can be modeled by the normal distribution, also known as the "bell-shaped curve." With a larger coefficient for sigma (1 sigma, 2 sigma, ... , 6 sigma) more of the set is included, corresponding to a larger area under the bell curve. For more on this, see any introductory text on statistics.

In most informal discussions, "x sigma" means the range x standard deviations above and below the average, or a span of $2x$ (that is, $\pm x$ sigma.) For 6 sigma the total range spans 12 standard deviations. As the sigma value increases, a larger area under the bell curve is included: 50 percent at ±0.67 sigma, 68.3 percent at ±1 sigma, 99.7 percent at ±3 sigma, and more than 99.999999 percent at ±6 sigma.

SIX SIGMA'S SIGNIFICANCE. As a concept in statistical quality models of physical manufacturing processes, 6 sigma has a very specific meaning (Bill Smith, "Six Sigma Design," *IEEE Spectrum*, Sept. 1993, pp. 43-46). Manufactured parts typically have a nominal design value for characteristics of merit. A *characteristic of merit* is any property of a component that we consider important to a particular application. For example, we would care about the size and weight of a particular part, but probably not about its color. Thus, we might note that the nominal design diameter of a shaft is 1.0 mm, its nominal weight is 45 mg, and so on. Processes that produce parts are typically imperfect, so the actual value for characteristics of merit in any given part will vary from nominal. Assemblies that use these parts must be designed to tolerate parts with such variances. Determining these tolerances is a classical systems engineering problem. However, once set, any part that doesn't fall within these limits is defective.

Components designed to 6 sigma tolerate more deviance than those designed for lower sigma values.

While it might seem that a 3 sigma tolerance is generous, it typically results in an unacceptable defect rate. With 6 sigma tolerances, a single part, and a stable production process, you'd expect to have only 2 defects per billion. However, manufacturing processes vary from one batch to the next, so the batch average for a characteristic often drifts between ±1.5 sigma. If the mean of a batch happens to be at either extreme, many of the parts will be defective: with a 3 sigma standard, this would yield 66,810 defective parts in a million. Allowing for a ±1.5 sigma process drift, 6 sigma tolerances will result in 3.4 defects per million parts. Components designed to 6 sigma tolerate more deviance from nominal values than those designed for lower sigma values. It is easier to achieve a conforming part or product designed to 6 sigma than to 3 sigma because more variance from nominal is acceptable. This is the opposite of what you might expect for a "high-quality" result.

Multipart products compound the drift problem.

Editor:
Shari Lawrence Pfleeger
Systems/Software
4519 Davenport St. NW
Washington, DC
20016-4415
s.pfleeger@ieee.org

0740-7459/97/$10.00 © 1997 IEEE

The expected number of defective units for a single-part product can be modeled by the distribution for that part. But as the number of parts in a product increases, the chance of getting a product with no defective parts drops. If all parts are designed to 3 sigma, the chance of getting a defect-free 100-part product is 0.001: out of a thousand units, only one would be completely good. However, if all design tolerances are extended to 6 sigma, the chance of a defect-free 100-part product is 0.9997. Or, putting a more cheery spin on this, you'd expect that at least 96 percent of the 10,000-part products would be defect-free with 6 sigma design limits and no more than ±1.5 sigma process drift.

Should the 6 Sigma manufacturing model be applied to software? In a word, no.

This is how Motorola and other companies have set their "Six Sigma" standard. But 6 sigma tolerances are only part of their strategy for high-quality manufacturing. There are related design goals: reduce the number of parts, reduce the number of critical tolerances per part, and simplify tooling and assembly processes.

SIX SIGMA AND SOFTWARE. Should the Six Sigma manufacturing model be applied to software? In a word, no. While high-quality software is a good thing, there are at least three reasons why the Six Sigma model does not make sense for software.

Processes. Software processes are fuzzy. Every software "part" is produced by a process that defies the predictable mechanization assumed for physical parts. Even at SEI level 5, the simple variation in human cognitive processes is enough to obviate applicability. The behavior of a software "process" is an amorphous blob compared to the constrained, limited, and highly predictable behavior of a die, a stamp, or a numerically controlled milling machine.

Characteristics. Software characteristics of merit cannot be expressed accurately as ordinal tolerances; that which makes software correct or incorrect cannot usually be measured as simple distances, weights, or other physical units. Six sigma applies to linear dimensions and outcome counts of identical processes. For several reasons, the ordinal quantification of physical characteristics of merit cannot be applied to software without a wild stretch of imagination.

♦ A "software" either conforms or it doesn't. It is meaningless to attempt to define greater or lesser conformance. What would the units be? What would be the upper bound of conformance—would we reject too much conformance? What is the lower bound? That people and systems will tolerate egregiously buggy software is not the same thing as establishing a quantitatively meaningful scale. Performance profiles—such as "fast busy no more than one minute in 10,000"—are the only truly quantifiable design tolerance for software, but are never the only characteristic of merit for any real system or single software "part."

♦ The mapping between software faults and failures is many to many, not one to one.

♦ Not all observed anomalies are faults.

♦ Not all failures are caused by the application; in many cases, the fault lies in another application system or the software infrastructure: the operating system, GUI, and so on.

♦ We can't accurately measure the total number of faults in a given software component, we can only count those revealed by failures. A physical measurement provides an unambiguous value, limited only by the known precision of the measuring mechanism. Attempts to measure the extent of faults in software, by testing, are weak estimates. Exhaustive testing could provide such a measurement with certainty, but is impossible for any practical purpose. Nor does proving help, because a single fault is sufficient to disprove. The exact loss in "precision" for feasible testing strategies cannot be estimated with any certainty. Estimating reliability (the field failure rate) is not the same as estimating fault density.

♦ Jeffrey Voas and Michael Friedman argue convincingly that it is not the gross number of detected faults that is important for software reliability, but the ability of a system as a whole to hide as yet undetected faults (*Software Assessment: Reliability, Safety, and Testability*, John Wiley & Sons, 1995).

The characteristic of merit is *implicitly* redefined from ordinal to cardinal in every discussion of 6 sigma software I've encountered. This redefinition is problematic, erasing the ordinal model's analytical leverage and rendering ambiguous that which is being counted.

Even if a cardinal interpretation were valid, more problems remain: 6 sigma of what? A fault density of x or less per instructions, noncomment lines of source code, function points, or what? A failure rate of x or less per CPU seconds, transactions, interrupts, or what? Shall we use either 3.4×10^{-6} (the "drift" number) or 2.0×10^{-9} (the absolute number) for x? Other interpretations are certainly possible—this is exactly the problem.

As a point of reference, here are several reported defect densities for released software; KLOC stands for thousand lines of code.

♦ NASA Space Shuttle Avionics have a defect density of 0.1 failures/KLOC (Edward Joyce, "Is Error-free Software Possible?" *Datamation*, Feb.18, 1989).

♦ Leading-edge software companies have a defect density of 0.2 failures/KLOC. These companies are achieving 0.025 user-reported failures per function point or better (Capers Jones, *Applied Software Measurement*, McGraw Hill, 1991, p. 177).

♦ A leading reliability survey found an average defect density of 1.4 faults/KLOC in critical systems (John D. Musa, Anthony Iannino, and Okumoto Kazuhira, *Software Reliability: Measurement, Prediction, Application*, McGraw Hill, 1990, p. 116).

♦ Surveys of military systems indicate at best a defect density of 5.0 faults/KLOC and at worst a defect density of 55.0 faults/KLOC (Joseph P. Cavano and Frank S. LaMonica, "Quality Assurance in Future Development Environments," *IEEE Software*, Sept. 1987, pp. 26-34).

For the sake of argument, assume that a 6 sigma software standard calls for no more than 3.4 failures per million lines of code

Continued on page 105

Continued from page 102

(0.0034 failures per KLOC.) This would require a software process roughly two orders of magnitude better than current best practices. It is hard to imagine how this could be attained, as Joyce reports that the average cost of the shuttle code is already $1,000 per line.

Uniqueness. Software is not mass produced. Even if software components could be designed to ordinal tolerances, they'd still be one-off artifacts. It is inconceivable that you would attempt to build thousands of identical software components with an identical development process, sample just a few for conformance, and then, post hoc, try to fix the process if it produces too many systems that don't meet requirements. We can produce millions of *copies* by a mechanical process, but this is irrelevant with respect to software defects. Quantification of reliability is a whole 'nother ballgame.

SIX SIGMA AS SLOGAN/HYPE. I'm not against very high quality software—my consulting practice exists because very high quality software is hard to produce, but nearly always worth the effort. However, slogans like "six sigma" can confuse and mislead, even when applied to manufacturing (Jim Smith and Mark Oliver, "Six Sigma: Realistic Goal or PR Ploy?" *Machine Design*, Sept. 10, 1992). Used as a slogan, 6 sigma simply means some (subjectively) very low defect level. The precise statistical sense is lost.

Discussions of "6 sigma" software based on this vague sloganeering ignore the fundamental flaws of applying a model of physical ordinal variance to the nonphysical, nonordinal behavior of software systems. This is not to say there are no useful applications of statistical process control in software process management. My favorite is to use u-charts—which normalize the count of defects to the size of the item in which the defect is found—for inspection and test results.

I say we leave 6 sigma to the manufacturing guys. Our time would be better spent figuring out how to routinely get very high field reliability in software-intensive systems and agreeing on an operational definition for reliability measurement. ◆

President and founder of RBSC Corporation, Robert V. Binder has over 23 years of software development experience in a wide range of technical and management roles. He has published several books and writes a regular column on testing for Object Magazine. *Binder is a senior member of the IEEE and a member of the ACM. He can be reached at rbinder@rbsc.com.*