point

Applying SPC to Software Development: Where and Why

Ed Weller and David Card

Statistical Process Control focuses on key subprocesses in the overall software development process. Under the right conditions, SPC is another useful tool in our toolkit.

The question of whether Statistical Process Control (SPC) applies to software engineering processes is moot. Developers have successfully applied SPC to many software projects. Better questions are, where and why to apply it?

SPC attempts to make processes predictable or stable. "When will you be done? What will it cost? How good will it be?" We've all heard these questions at one time or another on projects we've worked on or led. As we become more proficient in project management and measurement, our answers to these questions become more accurate. Statistical analysis helps to refine our estimates to narrower ranges with a higher, and known, degree of confidence.

SPC focuses on controlling key subprocesses of the overall process. To be effective, SPC needs three attributes: sufficient observations, a controllable process, and a performance objective relevant to business.

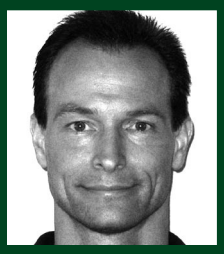
SPC guru Don Wheeler suggested that, for reasonably accurate SPC control charts, you need 15 or more *observations*, which aren't necessarily the same as plotted points on the control chart ("Good Limits from Bad Data," *Quality Digest*, Apr. 1997, p. 53). However, he also stated that, if you're careful, you can make some reasonably good process-stability evaluations with fewer observations. This indicates that you can apply SPC only where a subprocess has multiple executions.

The subprocess must be amenable to control—that is, it must be well-defined and of relatively short duration. We must have a mechanism for acting on it or adjusting its performance. If a process result isn't where we need it to be, we must have an associated step or activity that we can change to bring the process back into range. If defect detection is low in inspections and is related to the preparation rate, what planning process can we change to affect the outcome? To apply SPC properly to software processes, we need to decompose the process and clearly understand how the subprocess steps affect process performance.

The most frequently reported software development application of SPC is to inspection subprocesses—for example, using preparation rate, defect density, and inspection-meeting rate. Most of these reports talk about controlling the inspection process and identifying ways to improve defect detection. An unfortunately large number of them don't address subgrouping into homogeneous data sets, which often means control limits are so wide as to be useless for process control or evaluation. Organizations that segregate data into groups with like properties see a more useful set of control charts, typically with narrower control limits. For example, new code might be more structured and easier to read, and with fewer defects than modifications to legacy code. Separating the new code from the old might yield different means and narrower control limits than analyzing them as a single group. A few reports show

Continued on page XX

0740-7459/08/$25.00 © 2008 IEEE




counterpoint

Software Data Violate SPC's Underlying Assumptions

Bob Raczynski and Bill Curtis

Control charts, the primary tool used in Statistical Process Control, have profoundly influenced manufacturing, but are they appropriate for software processes? We believe the assumptions underlying control charts are so heavily violated in software data that their value in understanding and managing variation is severely diminished.

Consider the order-of-magnitude ranges among data points falling within the three-sigma limits for defect densities reported by Alice Leslie Jacob and S.K. Pillai ("Statistical Process Control to Improve Coding and Code Review," *IEEE Software*, May/June 2003, pp. 50–55). These control charts are typical of those seen in many high-maturity organizations. Would most managers consider a process with order-of-magnitude variation to be predictable or under control? On the basis of his work at IBM Rochester, Stephen Kan stated, "Many assumptions that underlie control charts are not being met in software data. Perhaps the most critical one is that data variation is from homogeneous sources. … The control limits in software applications are often too wide to be useful" (*Metrics and Models in Software Quality Engineering*, Addison-Wesley, 1995).

What are homogenous sources of variation?

Walter Shewhart's original formulation separates the causes of performance variation into common or chance causes and assignable causes (*Economic Control of Quality of Manufactured Product*, Van Nostrand, 1931). A process affected only by chance causes of variation (homogenous sources) is said to be operating within a common-cause system; because the variation is random, you can't identify its causes. An assignable cause creates performance variation that isn't random and causes detectable changes in a process's mean or variance.

A single person repeatedly producing components of similar difficulty under similar circumstances approximates a common-cause system. Control charts let us characterize the performance of this process and predict within limits how the person will perform in the future (Shewhart's original objective). If the person is assigned components of widely varying difficulty, a source of variation is injected into the process that will increase the person's performance variation, widen the control limits, and introduce greater error into our predictions. One way to solve this problem is to disaggregate the data onto different control charts for different levels of component difficulty.

Now, if we assign the components we're building to several different people who vary widely in skill, the variation and resulting control limits will be even wider because it intermingles common-cause systems. The error introduced into our predictions will likewise be far greater. If we disaggregate these data, we end up with too many control charts, most of which have too few data points to be useful. If we don't disaggregate the data, the control lim-

Continued on page XX

Software tasks are an intermingled mix of skill levels, component complexities, and project conditions that severely diminish the power of SPC techniques such as control charts.

0740-7459/08/$25.00 © 2008 IEEE

point

continued from page XX

how the results of inspection-data statistical analysis can predict defect-removal rates and residual defects entering test. These reports demonstrate the most powerful SPC applications to software.

A stable and controlled inspection process lets you predict the defect-removal rate. When you couple it with a full life-cycle defect history, you can predict the defects injected, removed by inspection, and remaining upon entering test. You can then use test-activity defect-removal rates, which might or might not be statistically derived, to predict post-ship defect rates. This is an important quality attribute for both your customers and your own maintenance-effort planning. While peer reviews often become the focus of SPC applications, other appropriate subprocesses include fixing a problem, processing a requirements change, and coding a software unit.

Of course, as Niels Bohr said, "Prediction is very difficult, especially about the future." An inescapable factor of prediction is that you must know the characteristics of the project team members and the product. The variations they introduce can make both control and prediction hazardous. Sometimes these factors make statistical methods chancy at best and misleading at worst. However, the bottom line is that, under the right conditions, SPC is another useful tool in our toolkit.

It's sometimes asserted that you can achieve SPC's goal through more complex techniques such as regression and variance analysis. These techniques typically make two key assumptions: that the data is normally distributed and that variances are relatively constant (that is, the variances in different samples are the same). Unfortunately, analysts often don't check these assumptions. A good SPC implementation explicitly addresses these two concerns. Of course, that means following the SPC process, not just jumping into control charts. Many SPC failures in software are due to naïve implementations, rather than weaknesses in the approach itself.

Ed Weller is the founder and president of Integrated Productivity Solutions, LLC. Contact him at efweller@aol.com.

David Card is managing director of Det Norske Veritas IT Global Services USA. Contact him at david.card@dnv.com.

counterpoint

continued from page XX

its become too wide to help us manage and predict process performance. For this reason, no manufacturer would mix data from different operators on different machines producing different parts on the same control chart. Why do we do this in software?

The interpretation of control charts rests on the assumption that a process operates within a single common-cause system. Unfortunately, most software development control charts plot data drawn from multiple intermingled common-cause systems. The result is many identifiable sources of variation acting simultaneously to violate the assumption of homogenous variation caused by chance factors. For instance, most peer-review control charts intermingle sources of variation for differences among developer skills, component complexities, and reviewer knowledge. Rather than representing 'the voice of the process,' most software control charts represent a cacophony of voices from many intermingled processes, each affected by its own unique sources of variation.

Software tasks are a continually changing mix of intermingled skill levels, complexities, project conditions, and other critical factors that are nearly impossible to disaggregate. When known sources of variation are intermingled in computing control limits, the limits become so wide that potential assignable causes go undetected, and claims for a process's statistical stability have no legitimate basis. Because we can't disaggregate the sources of variation in most software control charts, the statistical basis for their interpretation has been undermined. We should therefore typically interpret them heuristically (Kan's "pseudo-control charts").

Perhaps we would be better served by emphasizing the predictability of process outcomes over process stability. We would use knowledge of variation caused by sources such as developer skills and component complexities to predict performance with far greater precision than what's possible from the excessively wide control limits of most software control charts. Wouldn't it be more helpful to have methods that predict and evaluate what we expect from a person of skill X working on a component of complexity Y under condition Z?

Regression, multivariate statistics, statistically based simulations, parametric models, and other techniques are designed for characterizing performance and predicting results when faced with complex sources of variation. Unfortunately, the disproportionate focus on control charts for statistically controlling software processes and quality has diverted attention from other statistical methods that might provide far greater insight into and predictability from sources of variation affecting the software process.

Bill Curtis is the chief executive officer of Enturity. Contact him at curtis@acm.org.

Bob Raczynski is a staff software quality engineer at Lockheed Martin Corp. Contact him at bobraczynski@computer.org.

Weller and Card Respond

Raczynski and Curtis's primary concern with SPC and control charts seems to focus on the large variability in software processes. This ignores some SPC principles. The magnitude of variation isn't a factor in determining whether a process is stable. As long as the variability is relatively constant, even if large, the process is in control because it's statistically predictable. We can predict the average and standard deviation of future performance, even though we might not be able to predict any individual point with great accuracy. Thus, a process being executed by staff with varying degrees of skill could be judged stable, even if the variability is large. However, if we change the mix of staff—for example, to everyone having similar skill levels—we would destabilize the process relative to its initial performance, although it might restabilize at a condition of lesser variability. So, instability doesn't result from wide variation in skill levels but from changes in the skill-level mix.

Obviously, a process with a lot of variability, even if statistically stable, isn't as desirable as one with less. However, SPC's long-term goals are to reduce variability and to change the average performance level in the direction of "better." SPC accomplishes this by systematically analyzing the sources of variation.

The bottom line is that real software engineering organizations have produced and used control charts effectively to make practical decisions for some time. It doesn't make sense to argue on theoretical grounds that SPC and/or control charts don't apply to software processes, given this reality. SPC and control charts are logical first steps in learning to think statistically about process performance.

Raczinski and Curtis Respond

The only moot point is the value of drawing lines on charts of software data. However, debating SPC's value in constructing such lines is not moot. We agree that many SPC failures in software result from naive implementations, frequently caused by the difficulty of disaggregating software data into separate common-cause systems.

Weller and Card effectively describe appropriate conditions for control chart use. The fundamental question is, how much of a software project's data can be disaggregated into common-cause systems for which useful control charts can be computed?

In "Statistical Process Control: Analyzing a Space Shuttle Onboard Software Process" (IEEE Software, July/Aug. 2000, pp. 97–106), William Florac, Anita Carleton, and Julie Barnard disaggregated inspection data by inspection type, module type, and module size to approximate common-cause systems. Many segments of their 2 × 2 × 2 decomposition had too few data points for computing useful limits. Project developers too often find that they can construct useful control charts for only a limited portion of their data.

A typical special cause of variation discovered with control charts is inadequate preparation before inspections. Do we need control charts to detect this? Shouldn't this problem have been discovered by a process assurance team or by the inspection moderator using threshold measures for preparation time characteristic of CMMI Level 3?

If we can't disaggregate common-cause systems, control charts can offer little more insight than is available from lower-maturity practices. Under such circumstances, developers can use other statistical techniques to explore sources of variation and to predict outcomes. We question whether SPC provides the right statistical paradigm for these purposes.