Practical Applications of Statistical Process Control

Applying quantitative methods such as statistical process control to software development projects can provide a positive cost–benefit return. The authors used SPC on inspection and test data to assess product quality during testing and to predict post-ship product quality for a major software release.

Edward F. Weller, Bull HN Information Systems

You are in a ship readiness review. One goal for the software release is a two-to-one improvement in quality as measured by post-ship defect density. The system test group finds 50% fewer defects compared to the previous release of equal size. A review board member challenges the release quality, asking, “Why didn’t you find as many defects in the system test phase as in the last release?” How do you respond?

Quantitative methods such as statistical process control can provide the information needed to answer this question. For a major release of Bull HN Information Systems’ GCOS 8, a mainframe operating system for enterprise computing, we used SPC to analyze inspection and test data. We found that this helped us understand and predict the release quality and the development processes controlling that quality. It also gave us the hard data we needed to justify our results.

Prediction: Controlled vs. Uncontrolled Processes

A process’s behavior is predictable only if the process is stable, or under control. Statistical methods can help us evaluate whether an underlying process is under control. We can use control charts to calculate upper control limits (UCL) and lower control limits (LCL). (For background on UCL and LCL, see the related sidebar.) If a process stays within limits and does not exhibit other indications of lack of control, we assume that it is a controlled process. This implies that we can use its past performance to predict its future performance within these limits and can determine its capability relative to a customer specification.

Using SPC

Our goal in this release of GCOS 8 was to use defect density to predict the post-ship product quality with reasonable assurance.
We are aware of the problems with using defects to predict failures,\textsuperscript{1,2} but in the absence of other data or usage-based testing results, this was our primary means to evaluate release quality. We also expected to find fewer defects in the system test due to several process changes and needed to substantiate the presumed better quality quantitatively.

\textbf{Development phases}

Our first tasks were to determine that the inspection process was under control and then estimate the remaining defects. This sets the target for defect removal in the test phases.

\textbf{Upper and Lower Control Limit Basics}

We gather and analyze data as a basis for taking action. We use data feedback to improve processes in the next cycle, and data feed-forward to predict future events or values. Unless we understand the data’s characteristics, we might take incorrect action. Statistical process control is one analytical method for evaluating the data’s value for decision making. SPC lets us separate signals from noise in a data set.

One way we can express this is as

\[
\text{total variation} = \text{common-cause variation} + \text{ assignable-cause variation},
\]

The common-cause variation is the normal variation in a process, the result of normal interactions of people, machines, environment, and methods. These variations are the noise in the process. Assignable-cause variations arise from events that are not part of the normal process. An example would be a low problem-report input for one week followed by a high value the next week, caused by a failure in the problem-reporting system. These variations are the signals.

Upper and lower control limits (UCL and LCL) are two measures that help filter signals from the noise. Based on the Walter Shewhart’s work, UCL and LCL can be derived for two kinds of data:

- individuals or attributes
- variables.

Individuals or attributes data are counts related to occurrences of events or sets of characteristics. Variables data are observations of continuous phenomena or counts that describe size or status.\textsuperscript{1} Each data type requires a different technique for computing the UCL and LCL.

For individuals or attributes data, the Xmr (individuals moving range) chart is appropriate. This requires a time-ordered sequence of data, such as the number of problem reports opened per week. The formulas for the UCL and LCL are

\[
\text{UCL} = \text{Xbar} + 2.66 \times \text{mRbar}
\]

\[
\text{LCL} = \text{Xbar} - 2.66 \times \text{mRbar}
\]

where Xbar is the average of the values and mRbar is the average of the absolute differences of successive pairs of data. Table A shows data for building an Xmr chart.

We used this method to compute the data in Figure 10 in the main article. We use the same method for inspection preparation and inspection rates, where the attributes data are the rates for each inspection meeting.

An example of variables data is the defect density for a series of inspection meetings, which we can evaluate with u-charts. When we evaluate data from varying sample sizes, the plot looks like Figure 6 in the main text and the equations are

\[
\text{Ubar} = \frac{\text{Su}}{\text{Sa}}, \text{ or the total number of defects divided by the total size}
\]

\[
\text{UCL}_u = \text{Ubar} + 3 \times \sqrt{\text{Ubar} / a_i}
\]

\[
\text{LCL}_u = \text{Ubar} - 3 \times \sqrt{\text{Ubar} / a_i}
\]

where \(a_i\) is the sample size in lines of code.

Now that we have the control limits, what do they mean? The variation of data points inside the control limits is due to noise in the process. When points fall outside the control limits, we assume that this has an assignable cause—a reason outside the process’s normal execution. When assignable causes for out-of-control data points exist, we say that the process is out of control. The bottom line is that we cannot use the data to predict the process’s future behavior.

We gather data, compute the UCL and LCL where applicable, and evaluate the process behavior. If it is out of control, we look at the data points outside the control limits, find assignable causes for these points, and attempt to eliminate them in future execution of the process. If the process is in control, we can use the UCL and LCL to predict the process’s future behavior.

\textbf{Reference}


\textbf{Table A}

\begin{center}
\textbf{Building Xmr Charts}
\end{center}

<table>
<thead>
<tr>
<th>Week</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incidents</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>15</td>
<td>Xbar is 6.9</td>
</tr>
<tr>
<td>Movement range</td>
<td>—</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>mRbar is 3.6</td>
</tr>
</tbody>
</table>

\textit{Xbar} is the average of the values, and \textit{mRbar} is the average of the absolute differences of successive pairs of data.
Estimating defect injection. With enough data, you can use SPC to establish ranges for defect injection rates, accuracy of estimates of the size of the project’s source code, and defect removal rates (Defect injection is the inadvertent or involuntary injection of defects during development. It is different from fault injection, a technique for evaluating test case effectiveness. The defect removal rate, or inspection effectiveness, is the percentage of major defects that are removed in each inspection phase or the percentage of defects that are removed during all inspections.) However, for two projects that were major contributors to this release, we did not have enough data to establish defect injection rates using SPC. So, basing our defect injection estimates on specific product and project history, we used SPC to evaluate the defect removal rates and estimate the number of remaining defects entering the unit test phase.

Inspection data analysis. We have used source code inspections in GCOS 8 development since 1990. The process is stable and provides data used by project management. These inspections provided our first opportunity to apply SPC on these two projects, which I will call projects one and two. The work on project one fell into two parts: revising existing code and creating a product feature.

Figure 1 shows a histogram of preparation rates for 30 code inspections. (Lower preparation rates—that is, more time spent in preparation—are generally believed to lead to higher defect detection rates.) Most inspections fell into the 150- to 300-lines-per-hour range, with a few outliers. The low rates of the three inspections that were below 150 lph were due to small amounts of code. Of the inspections that were above 400 lph, two had high rates due to small size, and one was very large. After investigating these three inspections, I concluded that only the very large one was problematic. This inspection occurred near the end of the coding phase, when familiarity with the product and time pressure typically cause higher preparation rates.

I then compared the preparation-rate distribution with the inspection-rate distribution (see Figure 2). When analyzing data, I generally look for patterns. Figure 2 appears to have a bimodal distribution. Because the data included inspections of new code and modifications to the existing base, I divided the preparation rates into two classes. Figure 3 shows the results.

New-code inspections should “behave better” than those of revised code. Any inspections of revised code are small (23 to
150 lines), causing a larger variance in preparation and inspection rates. Knowledge of the inspected revised code also might have a wider variance than that of the new code. Figure 3 is typical of much of the inspection data that I investigate. The new-code inspection rates in Figure 3a approximate a normal distribution as closely as you are likely to see with actual data.

Because we wanted to predict defect densities entering the test phases, understanding the inspection process’s effectiveness was critical. The X chart and moving-range chart in Figure 4 indicate that the inspection process was not in control for the new code. Inspection 13 caused an out-of-control point on both charts. In addition, the first seven points in Figure 4a are below the mean. An investigation of inspection 13 revealed that its high preparation rate was due to an inspector’s lack of preparation. When we remove inspection 13 from the dataset, inspection 8 falls outside the recalculated UCL for the X chart, as well as being above the UCL in the mR chart. Inspection 8 had a high preparation rate due to sections of “cut and paste” code. So, treating inspections 8 and 13 as assignable causes of variation (see the sidebar on control limits for more on assignable causes), we obtained a recalculated X chart that shows a controlled process (see Figure 5).

When performing such an analysis, consider these three points:

First, the variation in rates might be due to either a process violation or an unusual work product (in this case, no preparation by one inspector or a work product with repeated code).

Second, when you decide to remove a data point from the analysis, it must have an identifiable special cause. Look at the remaining data critically to see if poor preparation is a problem, even if the data is within the control limits. As we’ll see, the remaining data from the analysis I just described is well behaved, suggesting that the removal of the two data points is justified.

Third, if the inspection or preparation rates are out of control, the product—not lack of time—might be a cause. For example, the cause could be a poorly written document or cut-and-paste code.

**Product analysis.** If the inspection process is under control, defect density is an indicator of product quality, not process quality. The control chart in Figure 6 shows the defect density for all 30 inspections of new and revised code. (Although the chart does not
show this, the data for the new-code inspections “behaved better” than the data for revised-code inspections because the sample sizes for the new code were more uniform and larger.)

I used a u-chart for Figure 6 because the sample size (area of opportunity—in this case, lines of source code) varied considerably (see the sidebar on control limits for more on u-charts). Inspections 1 and 8, which exceeded the UCL, were for revised code. Looking back at the preparation rates (see Figure 3), we can reasonably assume that the data for these inspections comes from a different process and thus remove it from this set.

Project two in the system release involved analyzing data for 51 inspections. The results of analyzing the inspection process were basically the same as those for project one. For project two, the defect data was in control on all but inspection 51, the last inspection in the set.

Feedback to the development team. We discussed the data with the project teams at their weekly meetings for three main reasons. First, it sent a message that the data was being used to make decisions on the project. Second, keeping the estimates and data in front of the teams made them aware of the progress toward the quality targets. Third, we wanted to avoid the “metrics are going into a black hole” problem that causes metrics programs to fail.

We now had two sets of data showing inspections were performed reasonably well. We then used the inspection data to refine the prediction for the number of defects remaining to be found for projects one and two in the release. Based on the differences in inspection process data for new and revised code from project one, we increased the inspection effectiveness estimates for the new code and lowered them for the revised code. In the absence of data, this would seem the likely thing to do. However, basing the revised estimates on data rather than assumptions adds credibility to the prediction process and provides a baseline for future predictions.

At the end of the coding phase, we found more defects than we had estimated; however, we now had the actual product size in source lines of code. So, we replotted the estimate (see Figure 7).

Revising and verifying the defect predictions. We can now verify the defect predictions, using the inspection effectiveness estimate to verify the size and defect injection estimates. The inspection process data is important in determining which of the estimates to believe if discrepancies exist between them. If the inspection process is under control, we look to the size or injection estimates for correction. If the data suggests the inspection process was out of control, we should lower the effectiveness estimate by an amount based on an understanding of that process.

Tying it together. In project one, the size reestimate caused a 13% increase in estimated defects—not a large number, but significant as we enter the later test stages. Our adjustment might look as if we did it to make the estimates look better; however, we make the pretest data fit the actual data as much as possible to better estimate the number of defects remaining in the product. The reason for changing the estimates should be documented for assessing the lessons learned during the project and for using as input to future project estimates.

I cannot offer a hard-and-fast rule for reestimating. I look to the data that has the most “substance” and evaluate the inspection data first, using the statistical methods discussed in this article. I also poll the inspection team members for their assessment of the inspections. In addition, I consider the data’s source, the accuracy of previous estimates, what’s most likely to be suspect, and “estimator’s instinct.”

Test phases

During the unit, integration, and system
test phases we monitored the defects removed against the estimates developed at the end of coding. The integration and system test phases gave us another opportunity to apply SPC.

Unit and integration test. Both project teams kept accurate records of defects found during the unit test and integration test phases. They also developed unit test objective matrices and unit test plans and specifications. So, we expected defect removal to be more effective than the 30% to 50% industry norm. (As it turned out, defect removal for both our projects was approximately 75%.)

Figure 8 shows the defect removal data for project two. (We use a chart such as this in our monthly project review and at weekly team meetings.) The vertical line on the right indicates the furthest stage where defect removal is happening, which is just before the beta ship phase. The chart incorporates the reestimate for the number of defects injected because of a size reestimate. Project reviews focus on the gap between the estimated injected defects and the actual removed defects. Because defect removal in the unit test phase was higher than estimated, a small number of defects were removed in the integration test and system test phases. Without accurate defect removal data from the unit test phase, these low numbers would be of more concern with respect to product quality.

System test. In this phase, we used SPC to answer the popular question, “When will we be finished with testing?” We can use the estimate of remaining defects in the product and the removal rate to estimate the “end date.” An Xmr (individuals and moving range—see the sidebar on control limits for more on Xmr charts) chart is useful for evaluating defect removal during system test.

We began data collection as we entered release integration testing, the second part of the integration test phase. Figure 9 shows the weekly problem arrival rate is under the UCL through week 10, for both projects. (Problems are potential defects.) The LCL is negative and therefore set to zero. (Up to this point, the figures have shown actual data; however, I’ve altered the data in Figures 9 and 10 to avoid disclosing proprietary information.)

Normally, you would want 16 to 20 samples from which to develop control charts, but the real world doesn’t always cooperate. With fewer samples, you should temper the conclusions drawn from data outside the control limits with the realization that you need more samples to establish the true process limits. With fewer than 15 data points, those points that are close to the limit might give you incorrect signals of out-of-control (or in-control) processes. In week 11, the problem rate hit 33, which was above the UCL. If you allow for establishing the control limits with 10 samples, this out-of-control data point suggests we look for a special cause. In this case, the system test phase started in week 11, using a more robust set of test cases and larger test configurations. Because a new test phase had begun, we recalculated the control chart starting from that week.

Without SPC, how would you reply to the project manager’s observation, based on the
data in Figure 10 up to week 20, that “There’s a downward trend for four weeks; looks like there is light at the end of the tunnel”? Week 20, of course, would dispel that conclusion. But what is more important, the data at week 20 doesn’t support a conclusion that the test cycle is drawing to a close. Based on SPC analysis, the only thing we can say is that the process is under control, with a predicted weekly problem arrival rate of between 2.4 and 27. To draw a valid conclusion, the input would have to be below 2.4 or above 27, or have a downward trend for seven weeks, none of which occurred. Note the difference at week 24, when the same four-week “trend” drops below the LCL, indicating an assignable or special cause of variation. This might be the “end of test” indicator if analysis determines the cause was indeed product related (not, for instance, a short work week, lack of progress due to a blocking problem, and so on).

Figures 9 and 10 illustrate what happens when data from two processes are mixed on one chart. It might be “obvious,” as in this example, or hidden, as in Figure 3.

Results

At this article’s beginning, I posed the question, “Why didn’t you find as many defects in the system test phase as in the last release?” For our product release, defect discovery was 48.2% of that for the previous release (normalized for size), and system test turns were halved. However, we could confidently defend our testing because we had

- sufficient process data from the inspections to be confident in their results, and
- sufficient data from the test phases to determine that the lower defect removal rate during system test resulted from better defect removal in earlier phases.

Although some of our conclusions and inferences were similar to those achieved through intuition or sound engineering judgment, we gained a fact-based understanding of many of our release processes. We were able to set quality goals, measure the results, and predict a post-ship rate with confidence. Our pre-SPC attempts to develop this information had limited success. Our focus on the process aspect of the release resulted in the identification of several improvements for the next release. And, to date, in limited customer use, only one defect has been found in these two products, and the release defect density is more than 10 times better than previous releases.

What was the additional cost? It included analysis of inspection data, collection of unit test data, and analysis of integration and system test data. Our inspection data is in a database, which lets us immediately extract the data necessary for the inspection SPC charts. We can insert the data into a spreadsheet template, which plots preparation rate and inspection rate X charts and defect density u-charts. The process takes less than five minutes. Unit test data collection is not free, but the savings in later problem analysis and tracking offset the cost. On a per-project basis, the initial data analysis cost is less than one to two hours per week. Of course, additional costs will be incurred as part of specific investigations initiated by the data analysis.

Further Reading

If you’re interested in exploring SPC in more depth, I suggest the following sources, on which I relied when preparing this article:


You should ask these questions about any analysis technique:

- Is it useful? Does it provide information that helps make decisions?
- Is it usable? Can we reasonably collect the data and conduct the analysis?

Our results show that SPC provides useful information to project managers, release managers, and development teams. The calculations are relatively easy, and spreadsheets make the process usable.

Acknowledgments

This article would not have been possible without the work of Phil Ishmael, Marilynn Sloan, Joe Wiechec, Eric Hardesty, George Mraz, Bill Brophy, Dave Edwards, Doug Withrow, Sid Andres, and other members of the development projects. Their willingness to collect the data made the data analysis possible. I also thank Dave Card, Mark Paulk, and Ron Radice for their reviews and suggestions.

References


About the Author

Edward F. Weller is a fellow at Bull HN Information Systems, where he is responsible for the software processes used by the GCS 8 operating systems group. He received the IEEE Software “Best Article of the Year” award for his September 1993 article, “Lessons from Three Years of Inspection Data.” He was awarded the Best Track Presentation at the 1994 Applications of Software Measurement conference for “Using Metrics to Manage Software Projects.” He is a member of the SEI’s Software Measurements Steering Committee. Mr. Weller has 28 years’ experience in hardware, test, software, and systems engineering of large-scale hardware and software projects and is a senior member of the IEEE. He received his BSE in electrical engineering from the University of Michigan and his MSE from the Florida Institute of Technology. Contact him at Bull HN Information Systems, 13430 N. Black Canyon, Phoenix, AZ 85029; e.weller@bull.com.