Statistical Process Control and Software Development

Richard E. Hasz
June 2010
# Table of Contents

Table of Figures ............................................................................................................................................. 2  
Introduction .................................................................................................................................................. 3  
Importance and Relevance of SPC & Software Development ........................................................................ 3  
Best Practices ................................................................................................................................................ 4  
Demonstration and Application .................................................................................................................... 6  
Conclusion & Reflection ................................................................................................................................ 8  
References .................................................................................................................................................. 10  

# Table of Figures

Table 1. Examples of the Use of SPC ........................................................................................................... 7
Introduction

Prior to introducing the overall topic, background information on Statistical Process Control (SPC) will be provided.

SPC is essentially a method of monitoring a process using control charts. Through the collection and analysis of data at various stages of a process, variations or trends that could affect quality can be detected. Consequently, variation in the process caused by special causes can be detected and corrected before in advance of the release of the end product. “Early detection of problems through SPC can reduce wasted time and resources and may detect defects that other methods would not. Additionally, production processes can be streamlined through the identification of bottlenecks, wait times, and other sources of delay by use of SPC” (Exponent Engineering and Scientific Consulting, 2008). SPC also provides a way for a company to demonstrate its quality capability which has become increasingly important in the global marketplace (Evans & Lindsay, 2008).

A special type of chart, called a control chart, is used with SPC. A control chart includes the element of time which is required to see if a process is in control or out of control. The control chart allows us to then distinguish between common causes of variation (normal variation within a process that does not require correction) and special causes of variation (unusual or abnormal variation that must be corrected to bring the process back into control). “Control charts have three basis applications: (1) to establish a state of statistical control, (2) to monitor a process and signal when the process goes out of control, and (3) to determine process capability” (Evans & Lindsay, 2008, p. 717).

Importance and Relevance of SPC & Software Development

It is an indisputable fact that software is daily becoming more important, vital, and critical to the survival of our modern culture. Whether for our enjoyment, or our convenience, or our health and safety, software is a must have. That being the case, it only makes sense that we look at improving the quality and lowering the cost of software through the use of tools that have proven their worth in other arenas. Since SPC has been shown to increase quality and reduce costs in a manufacturing environment, it is my opinion that SPC needs to at least be tried in the software development arena to determine if such gains in quality and cost can be achieved.

SPC is not very common in the software industry. This may be because SPC is viewed as being a manufacturing tool and is somewhat product-centric rather than process-centric (Komuro, 2006). It could also be that the reason that SPC is not commonly used in conjunction with software development is due to the characteristics of software development: (1) it is a creative activity and
is very human intensive, (2) multiple common causes can affect the output, and (3) inherent difficulties in obtaining large enough sets of data (Komuro, 2006).

Weller and Card (2008, p. 48) unequivocally state that the “question of whether Statistical Process Control (SPC) applies to software engineering projects is moot. Developers have successfully applied SPC to many software projects”. Raczynski and Curtis (2008, p. 49) dispute this conclusion quite emphatically. Their contention is that “the assumptions underlying control charts are so heavily violated (emphasis mine) in software data that their value in understanding and managing variation is severely diminished”. Some of the assumptions that they contend are violated include the use of data from homogenous sources, control limits are found to be too wide to be of any practical usefulness, combination of multiple common-cause systems onto single control charts, and a paucity of data points rendering a control chart less than useful. Raczynski and Curtis suggest that we have better tools available to predict performance in software development processes than control charts. Some of these techniques would include regression analysis, multivariate statistics, and parametric modeling that are much better suited “for characterizing performance and predicting results when faced with complex sources of variation” (p. 50).

Some of the reasons that Eickelmann and Anant (2003, p. 49) feel that a simplistic use of control charts in software development results in mistakes is (1) “people are the software production process”, (2) measurement of the software process could actually introduce more variation than the process itself, and (3) size metrics do not equate to discrete and identical units (in other words, a line of code does not necessarily equal a line of code). For these reasons, they recommend that the multiple measures in software development be viewed in context. One tool they recommend for this is a Kiviat (radar or spider) chart.

**Best Practices**

In analyzing control charts, we are looking for one or more of the following conditions:

- One plot point outside of the Upper Control Limit or outside of the Lower Control Limit.
- Two out of three consecutive plot points in Zone A (2σ to 3σ) on one side of the Center Line.
- Four out of five consecutive plot points in Zone B (1σ to 2σ) on one side of the Center Line.
- A run of at least eight plot points on one side of the Center Line or a run of steadily increasing (run up) or decreasing (run down) plot points.
• A non-random pattern of plot points such as an increasing or decreasing trend, a fanning-out or fanning-in pattern, a cycle, or an alternating pattern.

If none of these five conditions are observed, then the process can be considered to be in control. However, if one or more of these five conditions are observed, then the process is considered to be out of control and action must be taken to find the cause of the out-of-control pattern and eliminate the cause to bring the process back into control (Bowerman, O'Connell, Orris, & Porter, 2008).

Evans and Lindsay (2008, p. 721) offer a list of four general rules to follow in determining whether or not a process is in control:

• “No points are outside control limits.
• “The number of points above and below the center line is about the same.
• “The points seem to fall randomly above and below the center line.
• “Most points, but not all, are near the center line, and only a few are close to the control limits.”

They expand upon these general rules by providing detailed information on different patterns that may arise in a control chart: one point outside control limits, sudden shift in the process average, cycles, trends, hugging the center line, hugging the control limits, and instability.

Meanwhile, Hitachi Software Engineering (HSE) uses only four rules: (1) a single point is outside the control limits (3 sigma), (2) in a series of three data points, two of the three are on the same side of the centerline at least one sigma away from the centerline, (3) in a series of five data points, four of the five are on the same side of the centerline at least one sigma away from the centerline, and (4) nine values in succession fall on one side of the centerline. HSE maintains that these four rules help to minimize false positives in their datasets (Komuro, 2006).

For SPC and control charts to be used effectively, the charts need to be used on a regular basis (daily) to properly monitor performance of the process, to quickly discover special causes as they arise, and to make corrections if called for. Not only does this allow for the rapid identification of problems, it also helps to prevent unnecessary corrections which can lead to productivity issues and increased variation in the process. The philosophy of SPC is that employees who run a process are responsible for the quality of that process. This means that the employees, and not their managers or inspectors, should be maintaining and analyzing control charts.

Komuro (2006) concluded that the use of SPC with data from periodic peer reviews rather than testing of completed software modules effectively controlled the software development process. Komuro recommends using the following measures from peer review to control the process:
• Review Speed (size of work product divided by hours spent at review)
• Defect Density (number of detected defects at peer review divided by size of work product)
• Early Bug Detection Rate (number of detected defects at peer review divided by the total number of defects)
• Review Efficiency (number of detected defects at peer review divided by time and effort spent on the review).

To be effective, peer review must be conducted at selected points in the development process as opposed to being conducted when the development phase is completed and all work products are reviewed at one time. A technique for determining the optimum module size for review (in terms of lines of code) has been developed to minimize the cost of inspection (peer review) of the module (Jalote, Mittal, & Prajapat, 2007).

Demonstration and Application

The release of a mainframe operating system for enterprise computing used SPC to predict the quality of the release and of the development processes that influence that quality. Bull HN Information Systems found that defect removal reached 75% (as opposed to the industry norm of 30% to 50%). SPC allowed Bull HN Information Systems to develop conclusions or inferences about their release processes that are fact-based rather than being based on solely intuition of sound engineering judgment. The worth of SPC has been shown through the revealing fact that only one defect has been found in limited use of the software release by customers as well as the discovery of process improvements than will be implemented for the next release of the software product (Weller E. F., 2000).

In three real-world applications cited by Komuro (2006) from Hitachi Software Engineering, he drew the following conclusions:

1. Process measurement must be incorporated with traditional product measurement techniques used in SPC.
2. Individual data must be handled carefully. In particular, he recommends the use of XmR control charts rather than X-bar-R control charts (which he states seem to be useless in measuring processes in software development). This conclusion seems to answer some of the objections to the lack of homogenous data.
3. Stability of the data is more important than the size of datasets. Again, this seems to be an answer to some critics of SPC for software development as he states that “having a large amount of data often helps to establish a baseline in single cause system, but in multiple cause system you can not establish a baseline without separating the data set” (p. 584).
4. The psychological effect that occurs when improvement is attempted must be considered. He relates this to training and motivation.

Table 1 (The Data & Analysis Center for Software, 2010) provides a summarization of recent experiences of a number of organizations with SPC. Note that references in Table 1 are cited in the website at http://www.goldpractices.com/practices/spc/index.php.

Table 1. Examples of the Use of SPC

<table>
<thead>
<tr>
<th>SPC Technique</th>
<th>Application</th>
<th>Process/Attributes</th>
<th>Discussion/Results</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control chart</td>
<td>Military software unit maintaining almost 380,000 source lines of code</td>
<td>Defects per Person-Day and Defects per Thousand Lines of Code.</td>
<td>SPC techniques introduced along with software engineering project management techniques, inspections, and metrics. Process improvement resulted in 20% more code being produced than on previous version, rework in Integration and Test dropping from 4.7 to 0.09 person-years.</td>
<td>[French 1995]</td>
</tr>
<tr>
<td>X Control Chart</td>
<td>Process automation and consumer electronics projects in C++, ranging from 150 to 400 Function Points</td>
<td>Defects per Thousand Lines of Code for code review and testing. Lines of Code per Hour for inspection preparation.</td>
<td>Illustrates how using control charts identified out-of-control processes. Corrective actions were implemented to remove assignable causes.</td>
<td>[Jacob and Pillai 2003]</td>
</tr>
<tr>
<td>XmR Control Chart</td>
<td>GCOS 8 (an operating system)</td>
<td>Inspection review rate and defect-detection effectiveness of software inspections.</td>
<td>Identified sub processes (inspections of new code and revised code). Demonstrated inspections were under control. Estimated defects remaining in the code.</td>
<td>[Weller 2000]</td>
</tr>
<tr>
<td>XmR Control Chart and u-Chart</td>
<td>Space Shuttle Onboard Software</td>
<td>Inspection review rate, defect-detected in inspections and test.</td>
<td>Trial study of SPC. Identified inspection sub processes and natural variation of these processes.</td>
<td>[Florac, et al. 2000]</td>
</tr>
</tbody>
</table>
### Conclusion & Reflection

I have an inherent love of numbers, of mathematics, and particularly of statistics. When I chose this topic for my Quality Focus paper and decided to zero in on software development, I didn’t expect to find such divergent views of the efficacy of SPC used with software development. In fact, I thought I might gain some insights into the world of software development as I would like to expand my project management experience to include IT.

In my naiveté I always think that with numbers I will find black and white, yes and no, on and off – in other words, just a simpler world. But this assignment in particular has shown me the errors of my ways. I will not presume to judge the arguments for or against the use of SPC with software development as that would be arrogant to think that I could possibly know more than published experts in quality and statistics that are unable to agree amongst themselves.
However, one area of concern that I have with control charts, that I didn’t note being mentioned by any of the critics, is that the analysis is somewhat subjective. As noted earlier, Evans and Lindsay (2008), Bowerman, O’Connell, Orris, & Porter (2008), and Komuro (2006) have different criteria to determine when a process is or is not in control. Since control charts are intended to be used by employees to analyze processes they are responsible for, this could lead to problems due to analysis being subjective rather than objective. If we expect employees to make full use of control charts, then they must be competently trained in their use and analysis.

I still believe that with further study and experiments that SPC might find a place in the world of software development. At the very least, I continue to believe that this is an area of inquiry that is worthy of further study. Though in light of the arguments I have seen against the use of SPC and control charts, I would not be surprised if the final verdict was that more complex techniques (regression analysis, multivariate statistics, parametric modeling, etc.) will be required to predict and control the processes involved in software development.
References


