

Identify Outliers, Understand the Process

MARK C. PAULK
Carnegie Mellon University

KIM LASCOLA NEEDY AND JAYANT RAJGOPAL,
University of Pittsburgh

Identifying atypical performance in a software process or atypical entities in software data is important for statistically analyzing processes and products and for statistical process control (SPC) of the software process. In this research, data from the Personal Software Process (PSP) was analyzed using two different techniques for identifying atypical observations. It was found that simple techniques such as interquartile limits are approximately as effective as XmR control charts for identifying outliers in the absence of causal analysis. When outliers are appropriately removed from a process, the remaining data characterizes the "common-cause" system of typical implementation. The review rate in the PSP common-cause system was found to be faster than the recommended 100 to 200 lines of code per hour. Following recommended practice is a precursor to SPC.

Key Words

control charts, causal analysis, interquartile limits, outliers, Personal Software Process, PSP, software process, SPC, statistical process control, statistical thinking

SQP References

Closed-Loop Defect Removal Model Using Statistical Process Control

vol. 3, issue 1

Achamma Jose, Anju N. K.,
and S. K. Pillai

Integrating Improvement Initiatives:
Connecting Six Sigma for Software, CMMI,
Personal Software Process, and Team
Software Process

vol. 5, issue 4

Gary A. Gack and Kyle Robison

INTRODUCTION

Variation happens. The processes for building and maintaining software are performed by humans, not machines, and there can be order-of-magnitude differences in the performance of individuals (Curtis 1988). Managing by exception depends on the ability to identify exceptional events (outliers), which may be unusually difficult for software projects because of the high degree of intrinsic variation. This capability, however, is required for high maturity organizations as measured by models such as the Software Engineering Institute's Capability Maturity Model® for Software (Paulk 1995).

Identifying atypical performance in software processes or atypical entities in software data is important for knowing when to take action and what action is most appropriate. When processes are stable, but performance is unacceptable, systematic improvement is the appropriate action. When the normal behavior of the process is acceptable, but atypical (and undesired) behavior is occurring because of special circumstances, corrective action is needed.

Outliers may arise from a number of sources (Judd and McClellan 1989). First are errors in data entry, which must be addressed before further analysis is useful. Outliers may result from heterogeneous data sets (that is, they contain more than one separate type of entity), in which case the data must be split into homogenous sets for further analysis. In statistical process control (SPC), unusual events may be observed using the control chart (sometimes called the process behavior chart), which can be used to identify signals indicating assignable (or special) causes of variation, as opposed to data from the "common-cause" system being controlled (Wheeler and Chambers 1992; Wheeler and Poling 1998). Causal analysis of the signals allows the analyst to take appropriate corrective and preventive actions. Other statistical

analyses may be performed to answer questions such as whether a trend is present or whether a new process or tool provides improved performance. The analysis may be based on a designed experiment or through a retrospective analysis using opportunistically obtained data.

Outliers may skew the results of a statistical analysis, but outliers that are not clearly erroneous should not be completely discarded nor blindly included in an analysis (Judd and McClellan 1989; Neter, Kutner, and Nachtsheim 1996). Outliers may inspire insight into the process, but in a retrospective study, causal analysis of why outliers are atypical may not be feasible because the contextual information to support analysis is not available. Discarding outliers without root cause analysis, however, can adversely affect the validity of conclusions. Statisticians debate whether it is preferable to remove outliers before doing statistical analysis (Orr, Sackett, and DuBois 1991; Judd and McClellan 1989). If the results of a statistical analysis are consistent both with and without outliers, the credibility of the conclusions is stronger.

The purpose of this article is to compare techniques for identifying outliers in software processes and to identify some of the resulting insights into the common-cause system. The techniques considered for identifying outliers are XmR control charts and interquartile limits. Data from the Personal Software Process (PSP) were analyzed using these techniques. In the absence of causal analysis, it was found that simple techniques, such as interquartile limits, are approximately as effective as

XmR charts for identifying outliers. A vital precursor to SPC, however, is to follow recommended practice for effectively performing the process.

BACKGROUND

There are many different techniques for identifying outliers in a single-variable data set (Grubbs 1969), including control charts and interquartile limits. The terms “signal” (for control charts) and “outlier” (for interquartile limits) can be considered synonymous in this article. Control charts and interquartile limits are used to understand the PSP better.

The Personal Software Process

The PSP applies process discipline and quantitative management to the work of the individual software professional in a classroom setting. PSP is taught as a one-semester university course or as a multiweek industry training course. It typically involves the development of 10 programs, using increasingly sophisticated processes (Humphrey 1995). The life-cycle processes for PSP are planning, design, coding, compiling, testing, and a post-mortem activity for learning. The primary development processes are design and coding, since there is no requirements analysis step.

There are four PSP major processes (PSP0, PSP1, PSP2, and PSP3), as outlined in Table 1, and three minor

TABLE 1 Description of the PSP processes and assignments

PSP Process	Process Description	Assignments
PSP0	The “current” process of the student at the beginning of the course. Basic measures of historical size, time, and defect data are collected to establish an initial baseline.	1A: Using a linked list, write a program to calculate the mean and standard deviation of a set of data.
PSP0.1	Adds a coding standard, process improvement proposals, and size measurement.	2A: Write a program to count program lines of code. 3A: Enhance 2A to count total program and object LOC.
PSP1	Adds size estimating and test reports.	4A: Using a linked list, write a program to calculate linear regression parameters.
PSP1.1	Adds task planning and schedule planning.	5A: Write a program to perform a numerical integration. 6A: Enhance 4A to calculate linear regression parameters and the prediction interval.
PSP2	Introduces design reviews and code reviews, personal reviews conducted by an engineer on his or her own design, or code to identify and remove defects efficiently.	7A: Using a linked list, write a program to calculate the correlation of two sets of data. 8A: Write a program to sort a linked list.
PSP2.1	Adds design templates for functional specifications, state specifications, logic specifications, and operational scenarios.	9A: Using a linked list, write a program to do a c2 test for a normal distribution.
PSP3	Introduces the concept of cyclic development – incrementally building a program in multiple cycles.	10A: Using a linked list, write a program to calculate the three-parameter multiple regression parameters and the prediction interval.

© 2009, ASQ

extensions to those processes. Each process builds on the prior one by adding a few engineering or management activities. This minimizes the impact of process change on the engineer, who needs only to adapt the new techniques into an existing baseline of practices.

PSP students are asked to measure and record three basic types of data: time (effort), defects, and size. All other PSP measures are derived from these three measures. The PSP size measure is lines of code (LOC), chosen because it can be counted automatically, precisely defined, and is well correlated with development effort. Size also is used to normalize other data, such as productivity (LOC per hour) and defect density (defects per thousand lines of code (KLOC)). Each PSP program involves some amount of new development, enhancement, and/or reuse. Developing new or modified code represents most of the programming effort in the PSP course; consequently, new and changed LOC are the basis for most size measurement in PSP.

XmR Control Charts

The traditional process control technique for identifying signals in the process is the control chart. A control chart is a run chart with upper control limits (UCL) and lower control limits (LCL) added that indicate the normal execution of the process. The control limits are normally based on $\pm 3\sigma$ boundaries for the underlying common-cause system that the process represents. Shewhart (1939) chose 3σ limits for control charts in part because the 3σ limits were considered economically reasonable. Wheeler's Empirical Rule suggests that it is most common for 99 to 100 percent of the data from the common-cause system to be within 3σ of the average (Wheeler and Chambers 1992). Control charts provide a statistical method for distinguishing between variation intrinsic to the common-cause system and variation caused by anomalies in the process.

The simplest rule for detecting a signal (a possible assignable cause) is when a point falls outside the 3σ control limits. Many other sets of detection rules, such as run tests, have been proposed (Wheeler and Chambers 1992), which make the control chart more sensitive to signals, but which also lead to more false alarms. The choice of which detection rules to use should be based on the economic trade-off between sensitivity and unnecessary work.

There are many different kinds of control charts, including XbarR charts and u charts (Wheeler and

Chambers 1992). Some charts, such as the u chart, make assumptions about the statistical distributions followed by the data; the XbarR and XmR charts are empirically based (Wheeler and Poling 1998). The traditional control chart is the XbarR chart, which was developed for a manufacturing environment where multiple samples can be measured at specified points in production as a rational subgroup. The Xbar chart captures the average performance of each subgroup; the R chart measures the variation within a subgroup. Charts such as the XbarR chart that use subgroups are usually not appropriate for software processes, since software data are typically collected as individual observations.

The most commonly used chart for individual data is the XmR chart (Wheeler 1998), also known as the individuals and moving range chart. Two graphs are generated in an XmR chart: an X chart for the individual values and an mR chart for the moving ranges, that is, $mR_i = |X_i - X_{i-1}|$. The UCLs and LCLs, sometimes called the natural process limits, for the X chart (UCL_X and LCL_X), and the UCL for the mR chart (UCL_R), are calculated by

$$UCL_X = \bar{X} + 2.66(\overline{mR})$$

$$LCL_X = \bar{X} - 2.66(\overline{mR})$$

$$UCL_R = 3.268(\overline{mR})$$

where the X 's are the individual values and the mR 's are the moving ranges between adjacent values. The LCL for the moving range chart is always zero. \bar{X} is the average of the individual values, and \overline{mR} is the average of the moving ranges.

For SPC, the analysis occurs for a time-ordered process, where assignable causes represent violations or shifts of the common-cause system underlying the process. For statistical techniques such as analysis of variance, regression analysis, or tests of hypotheses, the data are assumed to be a random sample from a homogenous population. It can be argued that control charts should not be used for PSP data since the PSP assignments do not capture a process "over time" in the sense traditionally used for process control, but for a mature production process over an extended period of time, the sequence of production is not important (Hahn and Meeker 1993; Paulk 2005). Although identifying outliers may be a legitimate use of control charts, run tests are not appropriate in this context as a detection rule.

Control charts are primarily used in analytic studies, but they also can be used in enumerative studies (the converse is not true) (Deming 1975; Wheeler 1998).

Assignable causes of variation should be removed when calculating control limits, but causal analysis usually cannot be performed when doing a retrospective analysis (Liberatore 1995), because the information needed to identify why the signal occurred is not available to the analyst. If the analyst wishes to characterize the common-cause system, “robust” control limits can be calculated using a two-stage procedure that provides a reasonable compromise for identifying the “voice of the process” (Rocke 1988). Points outside the initial 3σ limits are removed, and robust control limits are calculated using the remaining data. In these analyses the authors use control charts in an enumerative, retrospective study to highlight common-cause systems in the software process and to illustrate the use of XmR charts on software process data, but the use of only one kind of detection rule and robust control limits is a consequence of this context.

Interquartile Limits

The interquartile range (IQR) is the difference between the Q1 (25 percent) and Q3 (75 percent) quartiles of a data set. A simple technique for identifying outliers in a data set is to set a lower interquartile limit (LIQL) at

$$\text{LIQL} = Q1 - (1.5 \times \text{IQR})$$

and an upper interquartile limit (UIQL) at

$$\text{UIQL} = Q3 + (1.5 \times \text{IQR})$$

This technique frequently is used to set the whiskers in a box-and-whisker chart (rather than the minimum and maximum values), and these limits are sometimes called “fences.” These IQLs provide an independent check on the plausibility of the robust limits as truly representing the voice of the process. The IQL technique is used to identify outliers after a data set is collected, while XmR charts are typically used for process control during process execution.

PREPARING THE PSP DATA

Before analyzing the PSP data, it is necessary to identify the relevant measures and check the validity and reliability of the data.

Measures for Process Control

The development processes to control in PSP are the design and coding processes, since there is no requirements analysis step. Defect data from the design and code reviews can be used to control design and coding. Causal analysis of signals in the defect data can point

to anomalies in either the production (design or code) or review processes. Review rates also are useful measures to consider and may provide insight into whether recommended engineering practice is being followed. The measures analyzed to identify the common-cause PSP system are, therefore:

- Program size, LOC
- Design review rate, LOC per hour
- Defect density in design review, defects removed in design review per KLOC
- Code review rate, LOC per hour
- Defect density in code review, defects removed in code review per KLOC

Program size is measured in LOC. Review rates are measured in LOC per hour reviewed for both design and code reviews based on the final number of LOC for an assignment. The recommended preparation rate for an inspection, which corresponds most closely to a PSP review, is 100 to 200 LOC per hour (Fagan 1986; Radice 2002). Defect density is measured in defects per KLOC.

Removing Invalid Data

Potential data errors were addressed before the outlier analysis began. Johnson and Disney (1999) identified a number of concerns for PSP data validity centered around the manual reporting of personal data by the students. Despite reviews by the instructor and exhortations to approach the course professionally, they found about 5 percent of the data to be defective. Some of the classes of data errors that they identified, such as errors of calculation, are irrelevant to this study because none of the analyses performed by the PSP students are used in this research—only the base measures reported. Entry errors are a concern since they occur in the data collection stage and are difficult to identify and correct, but fewer than 10 percent of the errors identified by Johnson and Disney were entry errors.

Data were available for 112 classes, with 1345 students providing data from 10,223 assignments where a program was successfully completed. Internal data consistency errors were identified for 2.8 percent of the reported data. Inconsistencies where the total number of defects injected did not equal the number removed (all defects identified are removed) resulted in the removal of 264 observations. For 21 observations, more defects were found in the design review than had been injected

by that point in the process. For four observations, more defects were found in the code review than had been injected by that point in the process. In summary, data errors were identified in 289 of the 10,223 observations in the PSP data set, leaving potentially 9934 for analysis.

Since many of the measures the authors wish to analyze are related to design and code reviews, which are introduced in assignment 7, only assignments 7-10 are candidates for analysis. To maximize the homogeneity of the problem being solved, to minimize the impact of learning curves, and to use relatively mature processes, only assignments 9 and 10 were used. To ensure that only one observation per subject is within a data set (to remove potential dependencies), they are analyzed separately. These data sets also were split by programming language to remove any potential technology bias. Table 2 lists the number of observations for each of the resulting four data sets.

The smallest of these data sets has 66 observations, which is ample for statistically significant analyses. Using four separate data sets allows one to replicate the analyses, providing greater confidence that the results are valid.

IDENTIFYING OUTLIERS

Outliers for program size, review rates, and defect density in reviews are identified and discussed next.

Identifying Size Outliers

Size is the most commonly used factor in defect prediction models and is a critical software product measure. Atypically large programs are also atypical solutions to the PSP assignments. The graph in Figure 1 contains the initial X (the top chart) and *mR* charts for program size in LOC for the data set (C, 9A). The X chart graphically shows that four observations fall outside the initial UCL of 324 LOC.

As shown in Figure 2 for the robust X chart, the robust control limits are much narrower than the initial control limits after the initial set of four “outliers” is removed. The X chart graphically shows that three observations continue to fall outside the robust control limit, but only one iteration of removals is allowed using

the two-stage procedure, since one cannot perform a causal analysis in this retrospective study. Each of the seven data points above the robust UCL is a signal of a possible assignable cause. The *mR* chart is not shown because it adds no useful information in identifying signals in these analyses. Some statisticians recommend not plotting the *mR* chart in any case, arguing that the X chart contains all of the information available (Nelson 1982; Roes, Does, and Schurink 1993).

Figure 3 shows the box-and-whisker chart for the (C,9A) data set. It also identifies seven outliers. The UIQL of 284.0 is operationally identical to the UCL of the robust control chart of 284.3 (LOCs are integer values).

FIGURE 1 Initial X and *mR* charts for program size in (C,9A)

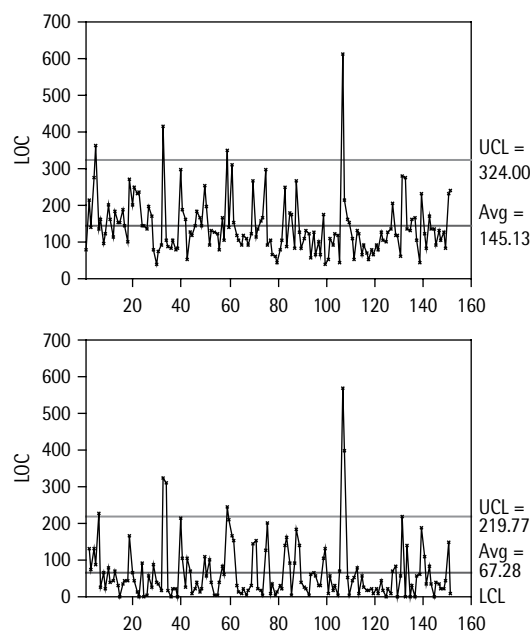


FIGURE 2 Robust X chart for program size in (C,9A)

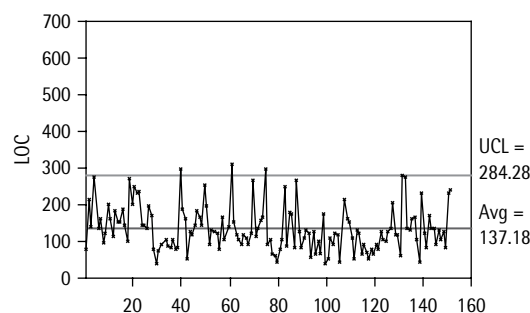


TABLE 2 Sample sizes for PSP data sets

Assignment	C	C++
9A	152	82
10A	133	66
7A-10A	1758	920

© 2009, ASQ

© 2009, ASQ

© 2009, ASQ

Table 3, which contains all the relevant information from the robust XmR chart and the box-and-whisker chart in a more compact form, lists the upper limits for both the XmR and IQL charts along with the number of signals or outliers identified using each technique. In three of the four data sets, the same outliers (or signals) are identified by both techniques. For the (C++,9A) data set, one additional observation is identified as an outlier by the X chart. There is only a 5 percent difference between the IQL and X chart upper limits (IQL/UCL_x). The average (\bar{X}) is included to indicate the typical program size.

An atypical size, that is, a point above the upper limit for program size, could indicate that an inappropriate solution was chosen for the problem. Differing emphases on aspects of a program such as flexibility and speed can have a dramatic impact on size, which has been characterized as the “problem of ambiguous programming objectives” (Weinberg 1998). LOC is frequently criticized as a size measure, especially by proponents of other measures such as function points (Jones 2008). For these size data sets, it can be observed that when implementing the same functionality in the same programming language, the typical program is roughly half the size in LOC of the largest program that could be reasonably expected from an empirical, statistical perspective. The insight one might draw is

that identical problem statements can have different solutions, depending on the style and preferences of the developer in writing code. To the degree that function points measure the solution rather than the problem, one obtains the same insight regardless of the size measure selected. The use of LOC in these analyses should not be considered a recommendation.

All of the lower limits for all of the PSP measures (size, review rate, and defect density) are less than zero for both XmR and IQL charts, which means that the effective lower limit is zero for the measures chosen.

Identifying Design Outliers

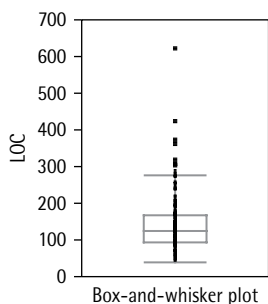
Outliers may be identified in the design process, for example, design effort or design review rate, or in the design, for example, design size, complexity, or defects. Design review rate and defect density in design reviews were selected as commonly used measures that illustrate the comparison of the XmR and IQL techniques.

Design review rate

Although inspections are performed by teams rather than individuals, it is worth noting that the recommended preparation rate for detailed design is about 100 to 200 LOC per hour. Although LOC is not a measure of design size, the recommended review rate for detailed design is the same as that of code (Fagan 1986), and LOC can be used to normalize the design effort in this context. PSP review time can be considered analogous to inspection preparation time, and PSP design to detailed design.

As shown in Table 4, the UCLs are well above the recommended rate of 200 LOC per hour for effective design reviews. If one considers the recommended review rate as a specification for properly performed design reviews, the upper limits for design review rate are not within the specification limits, and the design review process is not capable (in statistical terms), although even poor reviews are better than none at all. Since the design review process is not capable (even the average

FIGURE 3 Box-and-whisker plot for program size in (C,9A)



© 2009, ASQ

TABLE 3 Outlier statistics for program size (LOC)

Statistic	C 9A	C++ 9A	C 10A	C++ 10A
\bar{X} (for robust XmR)	137.2	159.2	187.9	245.3
UCL_x (for robust XmR)	284.3	380.9	360.4	552.6
XmR signals	7	2	7	2
UIQL	284.0	399.0	359.0	552.0
IQL outliers	7	1	7	2
IQL/UCL_x	1.00	1.05	1.00	1.00

© 2009, ASQ

TABLE 4 Outlier statistics for design review rate (LOC/hour)

Statistic	C 9A	C++ 9A	C 10A	C++ 10A
\bar{X} (for robust XmR)	450.6	502.0	474.6	457.8
UCL_x (for robust XmR)	1336.5	1637.3	1353.9	1557.1
XmR signals	13	10	10	5
UIQL	1278.6	1473.6	1443.1	1325.0
IQL outliers	14	10	9	7
IQL/UCL_x	0.96	0.90	1.07	0.85

© 2009, ASQ

design review rate exceeds the recommendation), the process improvement focus should be on ensuring that the design review rate conforms to recommended practice rather than analyzing signals for process control.

Points above the upper limit for design review rate indicate students who are not following the “normal design review” process as learned by most PSP students by this time in the course. Points inside the limits, but higher than 200 LOC per hour, suggest that the student’s behavior is normal, even if less effective than desired. For the 38 cases where there are signals for design review rate, based on the robust limits, the average defect removal effectiveness is 23 percent. Comparing this to the average defect removal effectiveness of 62 percent for design reviews with a rate less than 200 LOC per hour confirms that high review rates indicate less effective design reviews. This also corresponds to Fagan’s observation that 60 percent to 90 percent of defects are found in inspections that follow his rules (Fagan 1986).

Defect density in design reviews

As shown in Table 5, defect densities in design review can be as low as 27 defects per KLOC or as high as 53 defects per KLOC. Note that one also could have used the number of defects rather than defect density, but defect density allows one to look across multiple data sets to observe how comparable they are.

Points above the upper limit for defect density in design review could indicate that the design review was unusually effective in identifying defects or that the student was having difficulty with the application domain or methods. Causal analysis in the former case might identify process improvements helpful in improving the review process. In the latter case, it might suggest opportunities for teaching the student more effectively.

Identifying Coding Outliers

Similar to design, outliers may be identified in the coding process or in the code. Code review rate and defect density in code reviews are commonly used measures of the coding process and the code.

Code review rate

The recommended code review rate is 100 to 200 LOC per hour (Radice 2002). Similar to the case for design reviews, both the UCLs and the average code review rate exceed the recommended code inspection rate of 200 LOC per hour, as shown in Table 6, and the code

review process cannot be considered “capable” if one is comparing the “voice of the process” with the specification for an effective inspection.

Points above the upper limit for code review rate suggest that the student is not following the typical PSP code review process, similar to the case for design review rate. Points inside the limits, but higher than 200 LOC per hour, suggest that the student’s behavior is normal, even if less effective than desired. For the 28 cases where there are signals for code review rate, based on the robust limits, the defect removal effectiveness is 19 percent. Comparing this to the average defect removal effectiveness of 50 percent for code reviews with a rate less than 200 LOC per hour confirms that high review rates indicate ineffective code reviews.

Defect density in code review

As shown in Table 7, defect densities in code review can be as low as 53 defects per KLOC or as high as 77 defects per KLOC. These may be larger numbers than expected since PSP counts all defects found by the developer, including many that would not be reported in an industry setting. It should be pointed out that PSP defines a defect as a flaw in a system or system component that causes the system or component to fail to perform its required function. While all defects are counted in PSP, defects are not “cosmetic”; that is, a defect, if encountered during execution, will cause a failure of the system.

TABLE 5 Outlier statistics for defect density in design review (defects/KLOC)

Statistic	C 9A	C++ 9A	C 10A	C++ 10A
\bar{X} (for robust XmR)	10.5	12.0	6.1	8.3
UCL_x (for robust XmR)	45.1	53.0	26.6	37.1
XmR signals	15	6	7	3
UIQL	50.6	46.3	29.7	31.8
IQL outliers	10	7	7	5
IQL/UCL_x	1.12	0.87	1.12	0.86

© 2009, ASQ

TABLE 6 Outlier statistics for code review rate (LOC/hour)

Statistic	C 9A	C++ 9A	C 10A	C++ 10A
\bar{X} (for robust XmR)	365.7	355.9	438.7	429.4
UCL_x (for robust XmR)	999.9	1024.6	1090.6	1253.1
XmR signals	12	4	10	2
UIQL	981.7	962.3	1172.5	1031.8
IQL outliers	12	6	8	3
IQL/UCL_x	0.98	0.94	1.08	0.82

© 2009, ASQ

Points above the upper limit for defect density in code review could indicate that the code review was unusually effective in identifying defects, that the design inputs were poor quality, or that the student was having difficulty with the application domain or programming language. Causal analysis could therefore result in improvements to the code review process, changes to the design or design review processes, or assistance to the developer.

Implications for Process Control

Could XmR charts be effectively used for process control in a PSP-like context? Without causal analysis of the signals in the control chart, one cannot reach a firm conclusion, but it seems reasonable to observe that, since for both design and code review rates the control limits failed to fall within recommended best practice, it might be more desirable to ensure that recommended practice is followed! The most effective PSP reviews were observed to align with the recommended review rates of 100 to 200 LOC per hour. The broad range of review rates in PSP is hardly surprising, since students are learning how to perform their reviews effectively. Charts for defect density in reviews could add value once the review rate satisfies the recommended value of 200 LOC per hour or less.

The interquartile limits are about as effective as XmR charts in identifying outliers. The comparison suggests that detection rules such as run tests are needed to maximize the benefits of XmR charts. Use of more sophisticated control charting techniques, such as u-charts, depends on an analysis of the distributional assumptions made by those techniques.

Since the lower limit is always zero in these analyses, one cannot obtain any insight on “low” signals for PSP. A low signal for review rate might indicate a meticulous review, perhaps suggesting concern on the part of the

programmer about the quality of the design or code, or it might indicate a belief that the work product is particularly important. A “low” signal for defect density might indicate an inadequate review, which could suggest a high escape rate, or it might indicate a high-quality work product. The charts used in this article are unable to identify such cases because there is no useful lower limit.

The process instability index, S_t , provides a heuristic for determining whether a process is reasonably stable (Pierce 2005). S_t is the percentage of signals in the complete data set, and $S_t > 3$ percent is considered statistically unstable. For these data sets and variables, 81 percent of the processes are considered unstable. This suggests that there may be opportunities for the PSP instructors to provide additional support to some students, but this is an educational context, and large differences in student performance must be expected.

This analysis confirms the high variability and instability of individual performance, even when following a disciplined process at the end of the PSP course, where the variation has decreased relative to the *ad hoc* process at the beginning of the course. The range of performance between the average and the UCLs for each of the variables is 2X or more, indicating the differences in individual performance even when relatively disciplined processes are followed. Learning has been observed to continue after the PSP course is complete (Ferguson et al. 1997; Hayes 1998). In a sense, this analysis identifies worst-case bounds for performance when following a disciplined software process. It is difficult to conceive of an industrial setting that could match the potential for individual variation found in the PSP environment, since the preferred unit of study in an industry setting is the team or project.

THREATS TO EXTERNAL VALIDITY

Concerns are sometimes raised as to whether data from assignments done by students can be generalized to industry projects. PSP classes are frequently taught in an industry rather than an academic setting, however, and the developers in the authors’ sample reported up to 34 years of experience, with a median of seven years of experience. The PSP students therefore look more like typical industry developers than typical computer science students.

These analyses emphasized design and code review by individuals. The recommended review rate is 100 to 200 LOC per hour, and the recommended length for an

TABLE 7 Outlier statistics for defect density in code review (defects/KLOC)

Statistic	C 9A	C++ 9A	C 10A	C++ 10A
\bar{X} (for robust XmR)	21.2	22.6	15.7	16.0
UCL_x (for robust XmR)	73.9	77.0	53.3	52.8
XmR signals	9	6	2	3
UIQL	69.0	74.8	61.5	55.8
IQL outliers	12	6	2	3
IQL/ UCL_x	0.93	0.97	1.15	1.06

inspection meeting that the developer would prepare for is two hours. Thus, the typical size of the work product one might expect to see in an industry inspection would be about 200 to 400 LOC. The sizes of the PSP assignments fall roughly within these bounds; in only one of the four data sets is the upper limit larger than 400 LOC. Although industrial projects on the whole are, of course, much larger than the typical PSP assignment, the relevant task size is comparable.

PSP design and code reviews are performed by individuals rather than teams, so only those parts of the inspection (or peer review) process that are performed by individuals can be considered comparable. That means the PSP review rate corresponds to the inspection preparation rate, and no conclusions can be legitimately drawn with respect to inspection meeting rates. It is interesting to note that the defect removal effectiveness of PSP reviews is comparable to that reported for inspections given this caveat. This suggests that review rate is a critical driver of review effectiveness, but it also has been noted that high preparation effort may be a consequence of a high defect rate as well as a cause of detecting defects (Porter et al. 1998).

CONCLUSIONS

Analyses using XmR control charts and IQL charts provide similar insights in this retrospective study. Both demonstrate that outliers, or signals of assignable causes, occur in the PSP data more than would be expected by chance. One or two points outside the limits can be expected for these data sets using Wheeler's Empirical Rule that a homogenous data set will have approximately 99 to 100 percent of the data within the 3σ limits (Wheeler and Chambers 1992). Numbers greater than three or four for these PSP data sets suggest instability, that is, some students are not consistently following the PSP process. It also may be that some students had a bad day independent of their use of PSP or are simply poor programmers.

The latter suggestion highlights the concern that the results of statistical analysis should not be used for motivational purposes; "grades" in PSP should depend on learning how to program better, not on performing well relative to the other students. Many of the PSP students have not arrived at "industry best practice" with respect to review rates. The objective of PSP to induce learning based on personal data is an ongoing process that is not completed within the confines of the course, as has been

observed in previous studies (Hayes 1998; Ferguson et al. 1997). PSP instructors could potentially use these techniques to identify students who need additional help because they are atypical of PSP students, not because they had not yet implemented best practices.

Researchers have frequently concluded that investments in eliminating the lower tail of the individual differences distribution, whether via training or reassignment, provides the greatest potential benefit in improving performance (Curtis 1988). Disciplined processes can contribute significantly to addressing these issues (Paulk et al. 1995), but effective performance, which can be supported by SPC, is needed as well as consistency.

The contributions of this research are in providing these two insights: 1) XmR control charts using only out-of-bounds signals are roughly as effective as interquartile limits in identifying outliers likely to be assignable causes of variation (and implying that run-based signaling techniques would be useful additions for process control); and 2) when the stable process is identified, it may not be capable in terms of recommended practice, reinforcing the need for continual improvement to continue after the course.

To receive the best value from statistical techniques, a consistently implemented process is necessary but not sufficient for statistical control. The effective implementation of recommended practices also is needed before the adjectives "disciplined" or "mature" are appropriate. Statistical techniques such as XmR charts and box-and-whisker charts (with IQL whiskers) can be applied in industry software projects, and comparable insights directly pertinent to the real-world setting can be derived.

ACKNOWLEDGMENT

PSP data is provided by the Software Engineering Institute. Researchers interested in access to PSP data should contact Jodie Nesta at jsn@sei.cmu.edu.

Capability Maturity Model and CMM are registered with the U.S. Patent and Trademark Office by Carnegie Mellon University.

References

- Curtis, B. 1988. The impact of individual differences in programmers. In *Working With Computers: Theory Versus Outcome*, ed. G.C. van der Veer, 279-294. London: Academic Press.
- Deming, W. E. 1975. On probability as a basis for action. *American Statistician* 29, no. 4 (November):146-152.
- Fagan, M. E. 1986. Advances in software inspections. *IEEE Transactions on Software Engineering* 12, no. 7 (July):744-751.

Identify Outliers, Understand the Process

Ferguson, P., W. S. Humphrey, S. Khajenoori, S. Macke, and A. Matvya. 1997. Results of applying the Personal Software Process. *IEEE Computer* 30, no. 5 (May):24-31.

Grubbs, F. E. 1969. Procedures for detecting outlying observations in samples. *Technometrics* 11, no. 1 (February):1-21.

Hahn, G. J., and W. Q. Meeker. 1993. Assumptions for statistical inference. *The American Statistician* 47, no. 1 (February):1-11.

Hayes, W. 1998. Using a Personal Software Process to improve. In *Proceedings of the Fifth International Software Metrics Symposium*, Bethesda, Md., 61-71.

Humphrey, W. S. 1995. *A discipline for software engineering*. Reading, Mass.: Addison-Wesley.

Johnson, P. M., and A. M. Disney. 1999. A critical analysis of PSP data quality: Results from a case study. *Empirical Software Engineering* 4, no. 4 (December):317-349.

Jones, C. 2008. *Applied software measurement, third edition*. New York: McGraw-Hill.

Judd, C. M., and G. H. McClellan. 1989. *Data analysis: A model-comparison approach*. San Diego: Harcourt Brace Jovanovich.

Liberatore, R. L. 1995. Performance and efficiency of individuals charts in applications to obtain statistical control. Ph.D. diss., University of Pittsburgh.

Nelson, L. 1982. Control charts for individual measures. *Journal of Quality Technology* 14, no. 3 (July):172-174.

Neter, J., M. H. Kutner, and C. J. Nachtsheim. 1996. *Applied linear statistical models, fourth edition*. Chicago: Irwin.

Orr, J. M., P. R. Sackett, and C. L. Z. DuBois. 1991. Outlier detection and treatment in psychology: A survey of researcher beliefs and an empirical illustration. *Personnel Psychology* 44:473-486.

Paulk, M. C., C. V. Weber, B. Curtis, and M. B. Chrissis. 1995. *The Capability Maturity Model: Guidelines for improving the software process*. Boston: Addison-Wesley.

Paulk, M. C. 2005. An empirical study of process discipline and software quality. Ph.D. diss., University of Pittsburgh.

Pierce, V. D. 2005. Process stability analysis. In *Accenture Process Improvement Conference*, Chicago, June.

Porter, A. A., H. P. Siy, A. Mockus, and L. G. Votta. 1998. Understanding the sources of variation in software inspections. *ACM Transactions on Software Engineering and Methodology* 7, no. 1 (January):41-79.

Radice, R. A. 2002. *High quality low cost software inspections*. Andover, Mass.: Paradoxicon Publishing.

Rocke, D. M. 1988. Robust control charts. *Technometrics* 31, no. 2 (May):173-184.

Roes, K. C. B., R. J. M. M. Does, and Y. Schurink. 1993. Shewhart-type control charts for individual observations. *Journal of Quality Technology* 25, no. 3 (July):188-198.

Shewhart, W. A. 1939. *Statistical method from the viewpoint of quality control*. Mineola, New York: Dover Publications.

Weinberg, G. M. 1998. *The psychology of computer programming: silver anniversary edition*. New York: Dorset House.

Wheeler, D. J., and D. S. Chambers. 1992. *Understanding statistical process control, second edition*. Knoxville, Tenn.: SPC Press.

Wheeler, D. J., and S. R. Poling. 1998. *Building continual improvement: A guide for business*. Knoxville, Tenn.: SPC Press.

BIOGRAPHIES

Mark Paulk is a senior systems scientist in the Institute for Software Research at Carnegie Mellon University in Pittsburgh. He researches and teaches on best practices for software engineering and service management. This includes empirical research and case studies of best practice, with an emphasis on measurement and statistical thinking. From 1987 to 2002, he was with the Software Engineering Institute at Carnegie Mellon University, where he led the work on the Capability Maturity Model for Software. He was the lead author of *The Capability Maturity Model: Guidelines for Improving the Software Process*. Work related to the Software CMM included research into high maturity practices, statistical thinking as applied to software processes, and contributions to IEEE and ISO standards. He was co-project editor with Al Graydon of ISO/IEC 15504:2 (Process Assessment: Best Practices Guideline) from 1992-1995. He is a senior member of the IEEE, a Senior member of ASQ, and an ASQ Certified Software Quality Engineer. Paulk can be reached by e-mail at mcp@cs.cmu.edu.

Kim LaScola Needy is department chair and 21st century professor of industrial engineering at the University of Arkansas. She received her bachelor's and master's degrees in industrial engineering from the University of Pittsburgh, and her doctorate in industrial engineering from Wichita State University. Prior to her academic appointment, she gained significant industrial experience while working at PPG Industries and The Boeing Company. Her first faculty appointment was at the University of Pittsburgh. Needy's research interests include engineering economic analysis, engineering management, integrated resource management, and sustainable engineering. Results from her research are published in numerous scholarly journals including *The Engineering Economist*, *Engineering Management Journal*, and *International Journal of Production Research*. Needy is currently serving as president of the American Society for Engineering Management.

Jayant Rajgopal has been on the faculty of the Department of Industrial Engineering at the University of Pittsburgh since January 1986. He holds a doctorate from the University of Iowa. His current research interests are in mathematical programming, analysis of global supply chains, production/operations, and the modeling and application of RFID technology. In the past he also worked on reliability modeling, nonlinear (geometric) programming, and applied statistics. Rajgopal has taught, conducted sponsored research, or consulted in all of these areas. His publications have appeared in numerous journals including *IIE Transaction*, *Operations Research*, *Mathematical Programming*, *Naval Research Logistics*, *Technometrics*, *European Journal of Operational Research*, and *International Journal of Production Research*. He is a senior member of the Institute of Industrial Engineers and the Institute for Operations Research & the Management Sciences.