Do Not Get Out of Control: Achieving Real-time Quality and Performance

Craig Hale, Esterline Control Systems - AVISTA
Mike Rowe, Esterline Control Systems – AVISTA

Abstract. When lives are at risk if systems fail it is critical to minimize defects through the best software engineering processes possible. High-maturity processes are valuable for delivering quality, mission-critical software and supporting overall project performance. One standard tool used is Statistical Process Control (SPC). This allows a process to be monitored in real time to detect problems and eliminate their root causes. It also helps discover beneficial process improvements, so related organizational process improvements can be incorporated.

Introduction

Imagine the ability to see and adjust an activity before problems result. In a way, software development teams have this ability using high-maturity processes, such as those prescribed by CMMI® Maturity Level 5, to optimize and maintain performance levels. Certification and maintenance at that level involves organizational performance management [1]. This encourages organizations to make use of process metrics to refine and optimize their processes.

SPC is a technique that facilitates the monitoring of real-time process performance using control charts. Control charts plot key process parameters against historical organizational standards. One parameter of importance to many software organizations in our industry, and that will be utilized in this article to illustrate the technique, is defects per 1,000 lines of source code (KSLOC).

Run rules help identify non-random variations in process control charts. When non-random variation occurs, a process is considered to be "out of control." An out of control process triggers intervention to determine if anything outside the expected process performance has occurred using root cause analysis.

For example, if project performance to the defects per KSLOC baseline is determined to be out of control, then it is an important event and the project team should understand what has caused this to occur. Since SPC helps monitor process in real time, the root cause of this out of control event is probably still present. If an event is having negative consequences, then identifying and removing the root cause before too much damage has occurred should result in less rework time. If an event has a positive effect to the project, then the project team may change the process to encourage the continuation of the event. Let us look at how this works.

Control Chart Basics

Control charts provide a real-time graphical presentation of how a process is performing in relationship to a historical baseline. A historical baseline is produced by collecting and analyzing previous process data. Thus, it represents an organization's known process capability. If a process is not changed, one would expect that an organization's future process performance would fall within normal variation of this historical baseline.

Using the defects per KSLOC example, let us say an organization has historically been averaging 0.5 defects per KSLOC. There may be no concern if a data point jumps up to 1.5 defects per KSLOC. This is probably within normal variation of historical performance. But, if the figure goes above 4.0, or remains constantly above 1.5, then this might be a significant event. How do we determine whether to take action? This is where the additional components of a control chart come into play.

SPC generally makes use of two different charts simultaneously. One monitors actual process values or averages of values, and the other monitors process variability, such as a standard deviation or range.

Some software organizations use XmR control charts. The two sample control charts here illustrate how code defects per requirement review can be charted. The X chart plots and monitors actual individual process values, such as the total defects per KSLOC from each code review (see Figure 1). The mR chart plots and monitors a moving range (see Figure 2). A moving range is the absolute value of the difference in defects per KSLOC of two sequential code reviews. From the mR chart we can derive what to expect for normal variation.

Each control chart has several additional components that are useful in monitoring a process. The primary component of each is the data line, plotted with connected dots. The data points from the data line are labeled as "Included Data" on the X chart and "mR" on the mR chart. The lines represent the performance of a project's process.

The small "X"s on each chart are data points excluded from control chart computations. These points were investigated and were excluded from the included data and future calculations. One reason to exclude a data point would be that a defect was recorded against a software module because the associated requirement conflicted with another requirement. This type of defect is not really a code defect. It is a defect that was injected into the system at the requirements development stage. It is important to understand the capabilities and performance of defect injection in the code development process. Requirements-injected defects are monitored by another set of control charts.

The Center Line (CL) is the target value that we expect our process to perform around. For the X chart, the CL represents the unweighted average of the historical defects per KSLOC per

review for all similar projects within an organization. Thus, we would expect to see any typical code review utilizing this process to produce about the same number of defects per KSLOC.

In Figure 1, the CL is at 0.57. In the mR chart the CL is the unweighted average of the historical moving ranges for an organization. In Figure 2, this CL line is at 1.03. An unweighted average is a simple average without regard to the size or number of KSLOC in a review. A weighted average would take into account the size of each review when calculating this average.

Other components are Control Limits: Upper (UCL) and Lower (LCL). The UCL is generally set at 3.0 standard deviations above the CL, and the LCL is typically set at 3.0 standard deviations below the CL. In Figure 1 the UCL is at 3.3, and in Figure 2 the UCL is at 3.4. Setting the UCL at 3.0 standard deviations represents probability levels of roughly 0.001. A code review with 4.0 defects per KSLOC (see point 16 in Figure 1) exceeds the UCL of the X chart, indicating an unlikely occurrence purely by chance.

Since defects per KSLOC are dealing with small numbers, and obviously defects per KSLOC cannot go below zero defects per KSLOC, the LCL for the X chart is not used. For the mR chart, the LCL is set to 0.0 for obvious reasons. Although it is rare in software SPC, some processes may utilize the actual LCLs for X charts.

Some organizations select tighter control limits by setting them at a level less than 3.0 standard deviations. This will increase the number of events that trigger out of control events to investigate and the number of false alarms.

The shaded bands, or zones, on the X chart can help detect series of statistically unnatural events using run rules [2]. In SPC there are many possible run rules. For example, our organization uses four traditional run rules. They provide the power to discover unnatural variation for our needs. For organizations interested in information about run rules, Nelson Run Rules are a good starting point [3].

The zones are set at successive 1.0 standard deviation intervals from the CL. Zone C is within 1.0 standard deviations; zone B is from 1.0 to 2.0 standard deviations; and zone A is from 2.0 to 3.0 standard deviations.

If we observe two out of three consecutive points in zone A, then a run rule named "2 out of 3 in one A" triggers. The actually probability of such an event is, at most, 1.5 chances out of 1000. In addition to this run rule, we also monitor run rules called "4 points (on the same side of the CL) out of 5 in zone A or B" and "8 consecutive on the same side of the CL."

Finally, a grand mean is plotted on the control charts (see the dashed line on Figure 1). The grand mean is a weighted average of all defects divided by all KSLOC. Remember, the CL uses an unweighted average of defects per KSLOC per review. If a review had 2.0 KSLOC, it is counted with equal weight as a review containing 0.1 KSLOC. Grand means are generally not part of an XmR control chart, but they are useful and give the project team an overall sense of what the total defect rate for the project is running. Our team uses this defect rate to estimate rework effort.

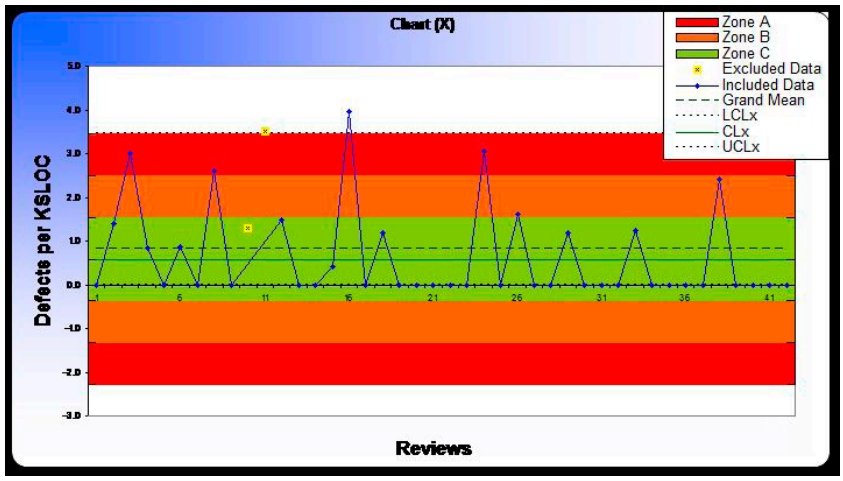

Figure 1: X Control Chart

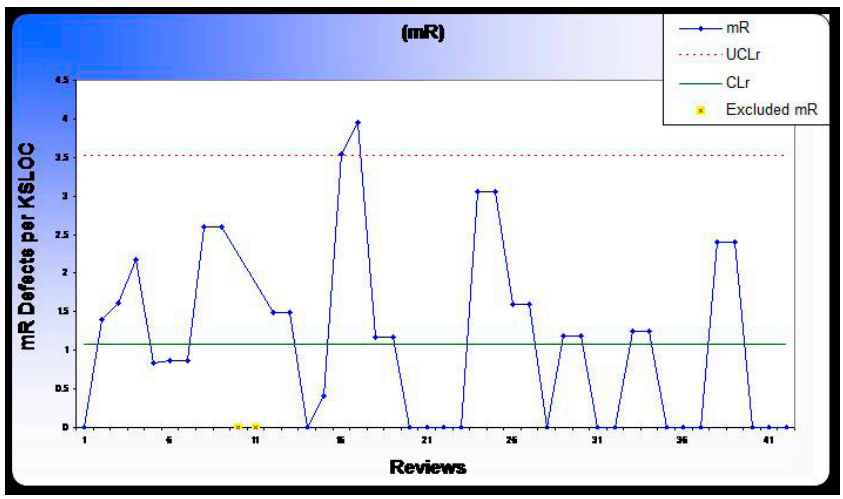

Figure 2: mR Control Chart

The original run rules for SPC were developed in the mid-1950s at a point prior to portable calculators. Using run rules is still standard SPC practice as they provide a quick and understandable way of identifying non-normal events without deep knowledge of statistics.

Control Chart Evolution: A Case Example

This section describes the evolution of the control charts used in our organization to develop life- and mission-critical software in a project-based environment. The evolution of our charts is driven by the data that we collect. With more data, we discover different, and often better, ways to utilize the data.

Our organization started looking at the "what and how" of measuring process performance back in 2004. We did this by defining organization-wide project metrics. Activities addressed include planning, system requirements, software requirements, software design, software coding, etc. These are based on regulatory requirements, such as DO-178B, "Software Considerations in Airborne Systems and Equipment Certification" [4], which regulates avionics software. Our team identified key quality and performance measures that could easily be captured and would provide meaningful information.

> Tracking total defects for a project is not very useful, since a review may have more total defects because they are developing a high number of lines of code. A smaller review may have fewer defects just because of the small size of the review. The performance of the process is understood by evaluating the variance. So, we derive measures to monitor, for instance defects per requirement or defects per KSLOC.

Within the code development activity, it is easy to track lines of source code, number of requirements implemented, code development time, code review time, number of defects, rework time (fixing defects), and rework verification time (time spent ensuring the defects were fixed properly). We track key activities utilizing some existing tools within our organization, with minimal burden on engineers or managers. Today there are COTS tools many organizations can use to do this.

Next came the development of organizational baseline control charts. Before claiming an organization baseline, our team requires two criteria be satisfied. First, there must be enough data points to ensure the baseline is statistically stable—a minimum of 40. Second, there must be a mix of enough different projects to generalize the findings to other projects. This is more of a qualitative decision based on knowledge of the projects.

After generating the first baselines, we noticed very high UCLs, indicating a great deal of variance. With a high UCL, some projects never generated out of control points, whereas other projects consistently triggered run rules. This was the result of developing a mixture of complex systems.

The team looked into the types of complex projects involved, such as avionics and medical systems. Organizational baselines were then split by application area. The resulting medical control charts became more useful at detecting outliers, but the avionics control charts were still not performing to the level desired.

The DO-178B guideline for avionics classifies features based on the criticality of failure from Level A to E. Failure of a Level A feature would result in a catastrophic failure condition for an aircraft. Level A design, development, testing and defects are treated much different than Level E failure, which are associated with defects that will not have an effect on aircraft operational capability or pilot workload [4].

Initially, these levels appeared to be a great way to partition the work. However, the problem partitioning baseline control charts by DO-178B is that a project may have requirements at multiple levels, so it is too difficult to accurately track all measures to these levels.

The team also looked at whether the project dealt predominantly with embedded or non-embedded software. This partitioning could be applied once at the very beginning of a project, simplifying the identification of the project type. Embedded avionics and non-embedded avionics organization control charts had very different CLs and UCLs. The variation from the control limits to the CL was statistically significant for the two control charts. But, we were still not done.

Within the embedded/non-embedded control charts, some reviews had much lower defect rates than others. The major differences were based on what was being reviewed. Higher defect rates were associated with new code that was reviewed for the very first time. Lower defect rates were associated with existing code that had been reworked and were undergoing a subsequent review due to additional code change.

Certain projects differed based on customer and type of work, so customer- and job-specific control chart baselines were created. Some of these differences are based on a customer's process maturity levels. A customer's process maturity level can impact how well they define system and software-related requirements. It also influences how stable the system and software requirements or target hardware remain during a project.

Even with stable organizational baselines the team strives to reexamine the CLs and control limits every six months. This ensures the process improves over time, since as a service-based organization our customer and project mix is changing over time, and with new technologies.

Hypothesis Testing

Evidence is gathered when considering whether to partition an organizational control chart. Some evidence is based on knowledge and experience with our business. Our team also uses statistical tools to identify and study potential differences.

When an organizational control chart is updated, part of the process is to look for differences among customers and projects. When differences are identified, they are statistically analyzed using t-tests or analysis of variance to determine whether the differences are actually statistically significant.

When customer or project data are statistically different the project management team determines why the difference exists. This drives the decision as to whether a more specific organiza-

tional control chart should be created. A new organizational chart is warranted if we expect to be doing similar projects in the future. One-time projects that are unlikely to be repeated are excluded from the organizational control charts. The information is documented just in case projects similar to this do start popping up.

Benefits

Maintaining and evolving organization control charts has required a significant effort. Without both tangible and intangible results we would not have continued this effort. The organization realizes benefits both before projects begin and while projects are executing.

Root Cause Analysis and Pilot Studies

When process outliers are detected by run rules, the points are quickly investigated. Generally, this is an informal process—a discussion between a project lead and an engineer. If it is a significant, repeated issue, a formal root cause analysis process is performed. This method uses fishbone or Ishikawa diagrams [5], where possible causes for the outliers are listed, followed by discussion and study of likely causes.

In some cases a pilot study or experiment is designed to test whether a change in the process will help prevent or repeat the desired effects in the future. The process is piloted on one or more projects, and data is collected and tested for efficacy. With baseline data, results are empirically compared before and after the process change to determine if it should be institutionalized.

One example of a process change that resulted from such a study was a guideline for review sizes. As review sizes got larger disproportionately fewer defects were identified. The guideline was to keep reviews at or below a certain size, where possible, by not bundling too many modules together. Removing defects as early in the process as possible in a lifecycle has been shown to be more cost-effective [6]. Even without the economic savings, removing as many defects as possible is particularly important for mission-critical software.

Project Estimation

The use of the data collected and analyzed is not restricted to active projects. It also helps estimate future projects. Many companies have a rough idea how long it takes to deliver an average requirement for an average project. Our historical results provide much better estimators for each lifecycle phase and for specific project types. As discussed earlier, organizational control charts provide estimators by lifecycle for avionics versus other types of projects, embedded versus non-embedded, particular customers and project types, and so on.

Since the organizational control charts provide both an average and the variability around the average, project estimates in the form of confidence intervals are produced. So, rather than just estimating a project cost, it is possible to estimate that the project cost has a 85% chance of being more than a low estimate and less than a high estimate. In a competitive marketplace, this can provide a significant edge in winning contracts and with profitability.

These more accurate and precise estimates also provide an advantage in scheduling and resource utilization. Keeping projects reasonably staffed helps avoid engineer burnout, and keeps stakeholders happy by delivering projects on time with high quality.

Another use of the data provided by organizational control charts is in building predictive models. One such model allows more precise predictions of later lifecycle effort, rework hours to fix defects, based on early lifecycle effort, review time and number of defects found. If it is taking longer to review and there are more defects, then the number of hours it will take to rework defects can be estimated. The corollary is also useful in knowing that a project will require less rework time. Having this knowledge helps with resource planning and allows mid-course corrections to help projects stay on schedule.

Conclusion

Overall, SPC has had several positive benefits for the work our team performs. The data provides team leaders the ability to make corrections in real time. This increases the likelihood of achieving the project goals. In other words, utilizing SPC can detect and correct issues before it is too late and encourage repeating desired process improvements.

Using control charts also helps identify significant process improvements. Analyzing before and after data helps determine if improvements are effective.

The organizational control charts have allowed us to very precisely understand performance levels. We understand how long tasks take, defect rates, how much rework can be expected, the variability in the complexity of requirements, and more. This is a significant factor for estimating contracts, as well as keeping projects on time, on budget and of high quality.

As has been described above, deploying SPC into an organization is not an easy overnight process. The following actions will help with effective preparation and management of SPC:

- Institutionalize software processes, so results from historical projects can be generalized to future projects.
- Create strong data collection systems that do not burden engineers with record keeping. In our case, these systems actually make an engineer's job easier. We already invested in these systems, so it was not as significant an effort as starting from scratch. Our data collection system was internally produced and has been refined over the last 15 years to satisfy our organizational and customer needs. Companies just getting started in data collection, may be able to find adequate COTS systems that meet their needs.
- Understand the business to determine which key parameters are actually useful to track to help optimize the business and processes.
- Study outliers to discover how and why their processes are producing them. Following this, the organization must try to prevent undesirable events and repeat desirable events by modifying their process.
- Finally, update the organizational control charts as processes and technologies evolve.

ABOUT THE AUTHORS

Craig Hale is a process improvement manager at Esterline Control Systems – AVISTA. He was a lead member of the company's successful CMMI Maturity Level 5 appraisal effort, and he serves as the software engineering process group lead to ensure continuous improvement. Hale holds an Associate degree in computer programming and systems analysis from Southwest Technical College. He is a certified Project Management Professional, and a member of SEI.

Esterline Control Systems – AVISTA
1575 E Business Highway 151
Platteville, WI 53818-3844
Phone: (608) 348-8815
E-mail: craig.hale@esterline.com

Mike Rowe is a professor of software engineering at the University of Wisconsin-Platteville and also works for Esterline Control Systems – AVISTA. He first used statistical process control in the early 1980s to study test yield rates on the HARM missile while with Texas Instruments. Rowe holds two Ph.Ds.—one in physiological psychology, with a minor in statistics, from the University of North Dakota, and the other in computer science from the University of North Texas.

Esterline Control Systems – AVISTA
1575 E Business Highway 151
Platteville, WI 53818-3844
Phone: (608) 348-8815
E-mail: mike.rowe@esterline.com

REFERENCES

1. Chrissis, Mary Beth, et al, CMMI for Development (3rd Edition), Westford, MA.: Pearson Education, Inc., 2011.
2. Western Electric Company, Statistical Quality Control Handbook. (1st Edition), Indianapolis, Indiana: Western Electric Co., 1956.
3. Nelson, Lloyd S., "Technical Aids," Journal of Quality Technology 16, no. 4, (October 1984): pages 238-239.
4. RTCA, Inc., "Software Considerations in Airborne Systems and Equipment Certification – RTCA/DO-178B," Washington, D.C., 1992.
5. Ishikawa, Kaoru (Translator: J. H. Loftus), Introduction to Quality Control, Tokyo: 3A Corporation, 1990.
6. Boehm, Barry W., Software Engineering Economics, Englewood Cliffs, N.J.: Prentice-Hall, 1981.