CMM Level 4 Quantitative Analysis and Defect Prevention

Al Florence
MITRE Corp.

The Software Engineering Institute's Software Capability Maturity Model® (SW-CMM) Level 4 quantitative analysis leads to SW-CMM Level 5 activities. Level 4 Software Quality Management (SQM) key process area analysis, which focuses on product quality, feeds the activities required to comply with defect prevention (DP) at Level 5 [1]. Quantitative Process Management (QPM) at Level 4 focuses on the process that leads to technology change management and process change management at Level 5. At Level 3, metrics are collected, analyzed, and used to status development and to make corrections to development efforts, as necessary. At Level 4, measurements are quantitatively analyzed to control process performance of the project and to develop a quantitative understanding of the quality of products to achieve specific quality goals. This paper presents the application of statistical process control (SPC) to accomplish the SQM and QPM and apply these results to DP. Real project results are used to demonstrate the use of SPC as applied to software development. An overview of control charts is presented along with Level 4 quality goals and plans to meet these goals.

An organization performing Level 4 quantitative analysis recognizes that it leads to Level 5 activities. This article presents this progressive relationship in project examples where statistical process control (SPC) is used to analyze measurements. Results of this analysis are used to gain a quantitative understanding of process capability, manage progress toward achieving quality goals, and for defect prevention.

The project used here is a large information system with a database of more than 30 million records developed on networked, client/server, workstations, and a multiprocessor parallel server utilizing C, UNIX, and ORACLE. The project was developed during five years with a staff of more than 100 with 300,000 lines of code and many commercial off-the-shelf packages. The project had achieved Level 3 in the SW-CMM, and the organization was pursuing Level 4. All Level 4 and Level 5 processes were installed and conducted on the project during a period of time.

This author was the software manager of the project at the time Level 3 was achieved. The author was also the SEPG lead during Level 4 activities and developed and installed Level 4 and Level 5 processes on the project.

The main quantitative tool used was SPC, utilizing control charts along with other methods. The project analyzed life-cycle data collected during development for requirements, design, coding, integration, and during testing. Defects were collected during these life-cycle phases and were quantitatively analyzed using statistical methods. The intent was to use this analysis to support the project in developing and delivering high quality products, and at the same time use the information to make improvements, as required, to the development process.

Rigorous statistics have been used in manufacturing but have had limited use in software development. The SEI's Capability Maturity Model Integrated[SM] (CMMI) calls for rigorous statistics at Level 4 and emphasizes SPC. This paper shows that control charts and other statistical methods can easily and effectively be applied in a software setting.

Control Chart Analysis

Figure 1 demonstrates how control charts are used for this analysis [2]. According to the normal distribution, 99 percent of all normal random values lie within +/- 3 standard deviations from the norm, 3-sigma [3]. If a process is mature and under SPC, 99 percent of all events should lie within the upper and lower control limits. If an event falls out of the control limits the process is said to be out of SPC; the reason for this anomaly needs to be investigated for cause, and the process brought back under control.

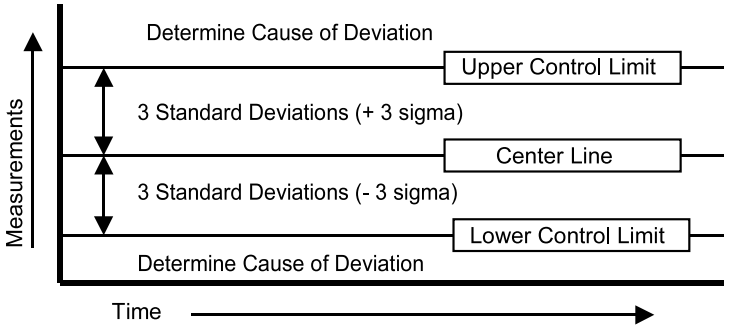
Control charts are used because they separate signal from noise. Thus when anomalies occur they can be recognized. They identify undesirable trends and point out fixable problems and potential process improvements. Control charts show the capability of the process, so achievable goals can be set. They provide evidence of process stability, which justifies predicting process performance [2].

Control charts use variables and attributes data. Variables data are usually measurements of continuous phenomena. Examples of variables data in software settings are elapsed time, effort expanded, and memory/CPU utilization. Attributes data are usually measurements of discrete phenomena such as number of defects, number of source statements, and number of people. Most measurements in software used for SPC are attributes data. It is important to use the correct data on a particular type of control chart [2].

When attributes data are used for direct comparisons, they must be based on consistent *areas of opportunity* if the comparisons are to be meaningful. In general, when the areas of opportunity for observing a specific event are not equal or nearly so, the chances for observing the event will differ across the observations. When this happens, dividing each count by its area of opportunity normalizes the number of occurrences [4].

Charts for averages (X-bar charts) and range (R charts) are

Figure 1. *Control Chart*



Capability Maturity Model Integrated (CMMI) is a service mark of the Software Engineering Institute and Carnegie Mellon University.

used to portray process behavior when the option exists to collect multiple measurements within a short period of time under basically the same conditions. When the data are collected as such, measurements of product or process characteristics are grouped into self-consistent sets (subgroups) that can reasonably be expected to contain only common cause variation. The results of the subgroups are used to calculate process control limits, which in turn are used to examine stability and control the process [4].

The following is a list of control charts that should be used for variable data and for attributes data:

Attributes Data:	Variable Data:
• u charts	• X-bar charts
• Z charts	• R charts
• XmR charts	• XmR charts

Level 4 Leads to Level 5

Figure 2 shows how data collection, analysis, and management from Level 4 activities leads to Level 5 activities of defect prevention, technology change management (TCM), and process change management (PCM) [5].

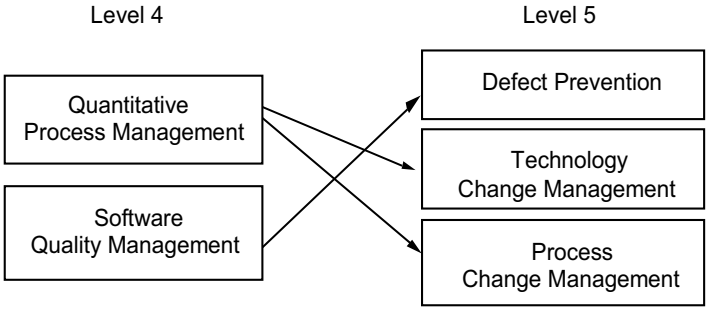


Figure 2. *Level 4 and Level 5 Paths of Influence*

Quantitative process management (QPM),which focuses on the process, leads to making process and technology improvements. Meanwhile software quality management, which focuses on quality, leads to preventing defects.

Level 4 Goals and Plans

The Capability Maturity Model V1.1 requires that Level 4 quality goals, and plans to meet those goals, be based on the processes implemented, that is, on the processes' proven ability to perform. Goals and plans must also reflect contract requirements. As the project's process capabilities and/or contract requirements change, the goals and plans may need to be adjusted.

The project this paper is based on had the following key requirements:

- Timing: subject search response in less than 2.8 seconds 98 percent of time.
- Availability: 99.86 percent seven days, 24 hours.

These are driving requirements that constrain hardware and software architecture and design. To satisfy these requirements, the system needs to be highly reliable and have sufficiently fast hardware.

The quality goals are:

- Deliver a near defect-free system.
- Meet all critical computer performance goals.

The plans to meet these goals are:

- Defect detection and removal during:
  - Requirements peer reviews.
  - Design peer reviews.
  - Code peer reviews.
  - Unit tests.
  - Thread tests.
  - Integration and test.
  - Formal tests.
- Critical computer resource monitoring:
  - General purpose million instructions per second (MIPS).
  - Disc storage read inputs/outputs per second (IOPS).
  - Write IOPS per volume.
  - Operational availability.
  - Peak response time.
  - Server loading.

The following quantitative analyses are real project examples applying SPC to real data over a period of time.

Example 1

Table 1 shows raw data collected during code peer reviews.

- Sample – series of peer reviews.
- Units – number of software units reviewed.
- SLOC – number of source lines of code reviewed.
- Defects – number of defects detected for each sample.
- Defects/1000 SLOC – defects normalized to 1000 SLOC for each sample.

Sample	Units	SLOC	Defects	Defects/KSLOC
1. Feb 1997	17	1705	62	36.36
2. Mar 1997	18	1798	66	36.70
3. Mar 1997	15	1476	96	65.04
4. Mar 1997	19	1925	57	29.61
5. Mar 1997	17	1687	78	46.24
6. Apr 1997	18	1843	66	35.81
Totals	**104**	**10,434**	**425**	

Table 1. *Code Peer Review Defects*

The calculations are shown in Table 2.

The formulas for constructing the control chart follow [2]. The control chart used is a u chart.

- Defects/1000 SLOC = Number of Defects * 1000/SLOC reviewed per sample (calculated for each sample). These are plotted as Plot.
- Control Limit (CL) = total number of defects/total number of SLOC reviewed * 1000.
- A(1) = SLOC reviewed/1000 (calculated for each sample).
- Upper Control Limit (UCL) = CL+3(SQRT(CL/a(1)) (calculated for each sample).
- Lower Control Limit (LCL) = CL-3(SQRT(CL/a(1)) (calculated for each sample).

The control chart is shown in Figure 3.

Table 2. *Calculations for Code Peer Review Defects*

Sample	Plot	CL	UCL	LCL	A(1)
1. Feb 1997	36.4	40.73	55.4	26.09	1.7
2. Mar 1997	36.7	40.73	55.01	26.45	1.8
3. Mar 1997	65.0	40.73	56.49	24.97	1.5
4. Mar 1997	29.6	40.73	54.53	26.93	1.9
5. Mar 1997	45.2	40.73	55.47	25.99	1.7
6. Apr 1997	35.8	40.73	54.84	26.63	1.8

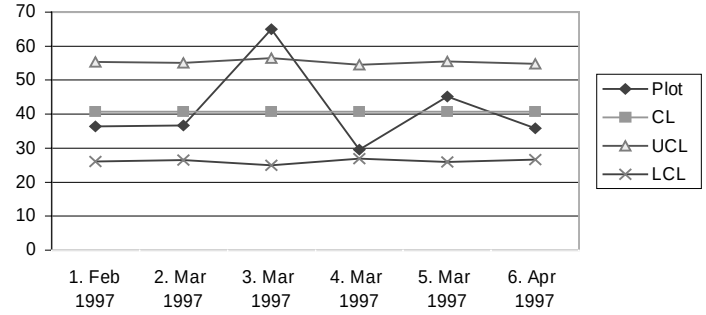


Figure 3. *Control Chart for Code Peer Review Defects*

The process is out of SPC in the third event. Causal analysis revealed this was caused when the project introduced coding standards and many coding violations were injected. The root cause is lack of knowledge of the coding standards. The defect prevention is to provide training whenever a new process or technology is introduced.

Example 2

Table 3 shows raw data collected at code peer reviews over a period of months:

Sample	Units	SLOC	Defects	Defects/KSLOC
1. Mar 1998	6	515	15	29.12
2. Apr 1998	10	614	16	26.06
3. Apr 1998	7	573	7	12.22
4. Apr 1998	7	305	7	22.95
5. Apr 1998	4	350	21	60.0
6. Apr 1998	3	205	2	9.76
7. Apr 1998	8	701	11	15.69
8. May 1998	3	319	3	9.40
Totals	**76**	**3,582**	**72**	

Table 3. *Code Peer Review Defects*

Table 4 shows the calculations. LCL is set to zero when it is negative.

Sample	Plot	CL	UCL	LCL	a(1)
1. Mar 1998	29.13	20.1	38.84	1.36	0.515
2. Apr 1998	26.06	20.1	37.27	2.96	0.614
3. Apr 1998	12.22	20.1	37.87	2.33	0.573
4. Apr 1998	22.96	20.0	44.45	0	0.305
5. Apr 1998	60.00	20.1	42.84	0	0.35
6. Apr 1998	9.76	20.1	49.80	0	0.205
7. Apr 1998	15.71	20.1	36.16	4.04	0.701
8. May 1998	9.40	20.1	43.91	0	0.319

Table 4. *Calculations for Code Peer Review Defects*

The control chart is shown in Figure 4.

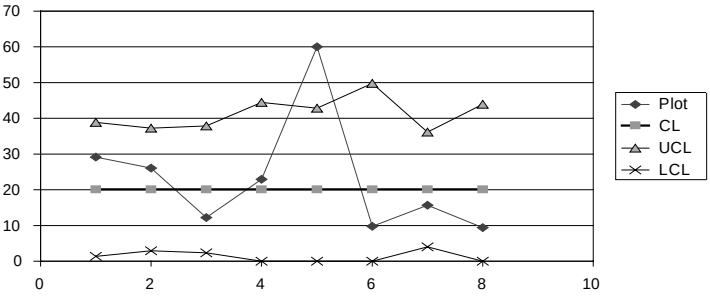


Figure 4. *Control Chart for Code Peer Review Defects*

An anomaly occurred in the fifth sample. Causal analysis revealed that data for that sample were for database code, all others were applications code. Control charts require similar data for similar processes, i.e., apples to apples analogy. The database sample was removed and the data charted again as shown in Figure 5.

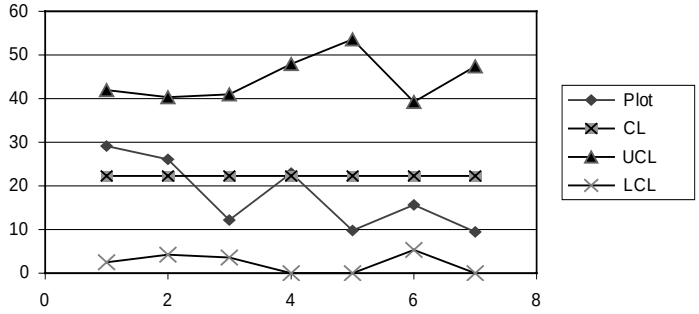


Figure 5. *Control Chart without Database Defects*

The process is now under SPC. The root cause is that data gathered from dissimilar activities cannot be used on the same statistical process on control charts. Data from design cannot be combined with data from coding. The process for database design and code is different from that used for applications design and code as are the teams and methodologies. The defect prevention is against the process of collecting data for SPC control charts.

Example 3

During integration testing, the defects were categorized against the test plan, test data, code logic, interfaces, standards, design, and requirements. Defects against these attributes are shown in Table 5.

Samples	Test Plan	Test Data	Logic	Interface	Standards	Design	Requirements
1	2	6					
2		10					
3	1	9	3				
4	2	1	13				
5		1	7				
6		10	14				
7		4	2				
8		28					
9			6				
10	1	3				2	
11		10					
12		9	1				
13		6	2	1			
14		5	7				
Totals	6	102	55	1		2	

Table 5. *Integration Test Defects*

Figure 6 plots the defects discovered during integration tests.

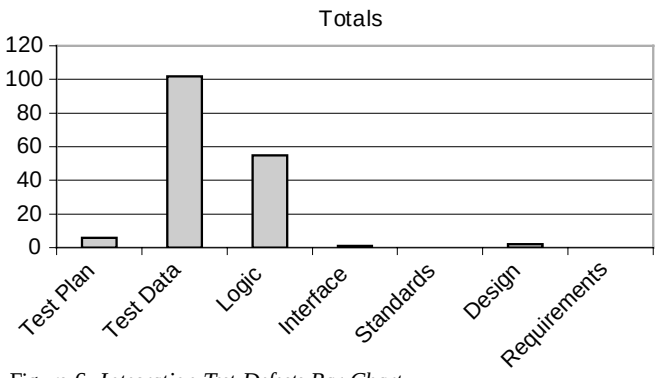


Figure 6. *Integration Test Defects Bar Chart*

Test data would not be expected to have the majority of defects. The root cause was that the test data in the test procedures had not been peer reviewed. The defect prevention mechanism is to peer review the test procedures and the test data.

Example 4

During preliminary design and prior to acquiring hardware, a simulated performance model was used to monitor critical computer resources. Figure 7 shows some results of monitoring the required MIPS. The model tracked estimated usage, the amount of MIPS required based on functional requirements to be implemented; availability, the amount of available MIPS in the model's design; and threshold, the number of MIPS that threaten the availability that requires remedial action.

Around November 1995, many new requirements were added to the system and the architecture's MIPS threshold was threatened because of increased computations. In May 1996, additional MIPS were added to the hardware design and the problem was corrected.

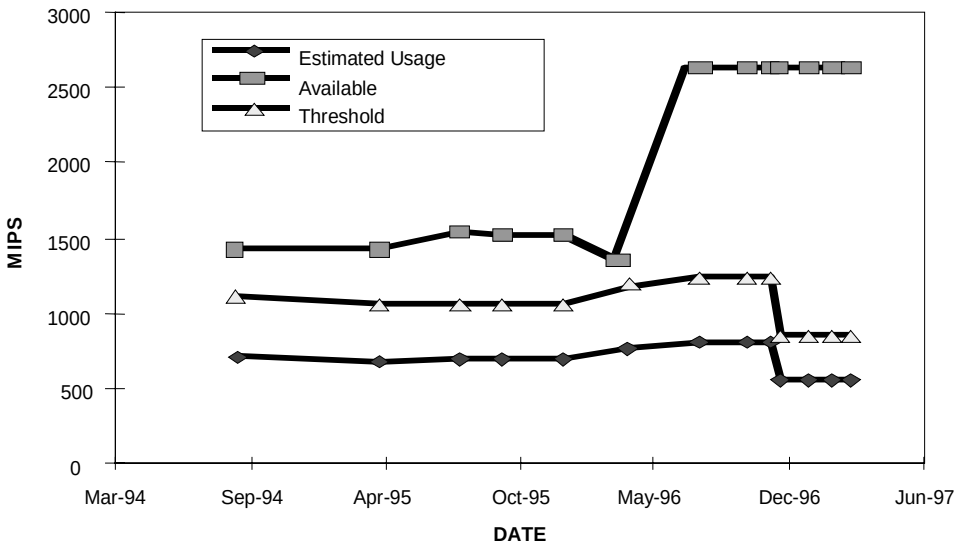


Figure 7. *General Purpose MIPS*

Conclusion

The use of rigorous statistics using SPC (control charts) and other statistical methods can easily and effectively be used in a software setting. SPC can identify undesirable trends and can point out fixable problems and potential process improvements and technology enhancements. Control charts can show the capability of the process, so achievable goals can be set. They can provide evidence of process stability, which can justify predicting process performance. SPC analysis can provide valuable information used in defect prevention and for lessons learned. SPC is new to software development but shows great promise that, in this author's opinion, will support process improvement, and will improve the productivity of development and the quality of products.◆

References

1. Paulk, Mark C.; Curtis, Bill; Chrissis, Mary Beth; Weber, Charles V., February 1993, *Capability Maturity Model for Software, V1.1*, Software Engineering Institute (SEI).
2. Florac, William A., Park, Robert E., Carleton, Anita D., *Practical Software Measurement: Measuring for Process Management and Improvement*, SEI, April, 1997.
3. Baumert, John H., McWhinney, Mark S., *Software Measures and the Capability Maturity Model*, SEI, 1992.
4. Florac, William A., Carleton, Anita D., *Measuring the Software Process, Statistical Process Control for Software Process Improvement*, SEI, 1999.
5. Radice, Ron, Getting to Level 4 in the CMM, 1997 SEI Software Engineering Process Group Conference, San Jose, Calif.

Suggested Readings

- Pyzdek, Thomas, *An SPC Primer*, Quality America Inc., 1994.
- Carleton, Anita; Paulk, Mark C.; Statistical Process Control for Software, 1997 Software Engineering Symposium, Pittsburgh, Pa.
- Chambers, David S.; Wheeler, Donald J., *Understanding Statistical Process Control*, SPC Press, 1995.
- *Juran's Quality Control Handbook*, 4th Edition, McGraw-Hill Book Company, 1988.
- Florence, Al, CMM Level 4 and Level 5 Plan, 1999 Software Engineering Process Group Conference Proceedings, Atlanta, Ga.
- Donald J. Wheeler, *Advanced Topics in Statistical Process Control*, SPC Press, 1995.
- Deming, W. Edwards, On Probability As a Basis For Action, *The American Statistician*, November 1975,Vol. 29, No. 4, pp.146-152.
- Florence, Al, CMM Level 4 Quantitative Analysis and Level 5 Defect Prevention, 2000 Software Technology Conference Proceedings, Salt Lake City, Utah.
- Humphrey, Watts S., *Managing the Software Process*, SEI Series in Software Engineering, Addison-Wesley Publishing Company, September 1997.

About the Author

Al Florence has worked at Hughes Aircraft, TRW, Martin Marietta, Science Applications International Corp., and currently at the MITRE Corporation. He has worked in all software life-cycle phases from concept to retirement in several disciplines, including systems, software, development, test, configuration management, and quality assurance, as both a developer and manager. He has diversified experience in real-time command and control, aircraft, spacecraft, missiles, weapon systems, particle accelerators, simulation, and information systems projects. He has developed processes for all CMM® key process areas at all CMM levels and is a trained evaluator and assessor. He has a bachelor's degree in mathematics and physics from the University of New Mexico and did graduate work in computer science at the University of California Los Angeles and the University of Southern California.

MITRE Corp.
1820 Dolley Madison Blvd.
McLean,Va. 22102-3481
Voice: 703-883-7476
Fax: 703-883-1889
E-mail: florence@mitre.org