

CMM Level 4 Quantitative Analysis and Defect Prevention With Project Examples

Al Florence

MITRE

The views expressed are those of the author and do not reflect the official policy or position of
MITRE

KEY WORDS

- Quantitative Process Management
- Software Quality Management
- Defect Prevention
- Quantitative Analysis
- Statistical Process Control
- Control Charts

ABSTRACT

The Software Engineering Institute's (SEI) Software (SW) Capability Maturity Model (CMM) Level 4 Quantitative Analysis leads into SW-CMM Level 5 activities. Level 4 Software Quality Management (SQM) Key Process Area (KPA) 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 which leads to Technology Change Management (TCM) and Process Change Management (PCM) 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) in accomplishing the intent of SQM and QPM and applying the results to DP. Real project results are used to demonstrate the use of SPC as applied to software development. The main quantitative tool used was Statistical Process Control utilizing control charts. 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 using the information to make improvements, as required, to the development process.

Rigorous statistics have been used for many years in the manufacturing industry to improve quality and productivity but have had limited use in software development. The SEI's Integrated CMM calls for rigorous statistics at Level 4 and emphasizes the use of statistical process control. This paper shows that SPC, using control charts and other statistical methods, can easily and effectively be applied in a software setting. Presented are the processes that the author formulated, launched and conducted on a large software development effort. The organization had obtained SW-CMM Level 3 compliance and was pursuing Level 4 and Level 5. All Level 4 and Level 5 processes were installed and conducted on the project over a period of time.

An overview of control charts is presented along with a review of the Level 4 KPAs and Defect Prevention at Level 5. Next, Level 4 quality goals and plans to meet those goals are described followed by some real project examples in applying SPC to real project data.

INTRODUCTION

Control Charts

Figure 1 shows a control chart and demonstrates how control charts are used for this analysis.[3] According to the normal distribution, 99% of all normal random values lie within ± 3 standard deviations from the norm, 3-sigma.[3] If a process is mature and under statistical process control, 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 statistical process control and the reason for this anomaly needs to be investigated for cause and the process brought back under control.

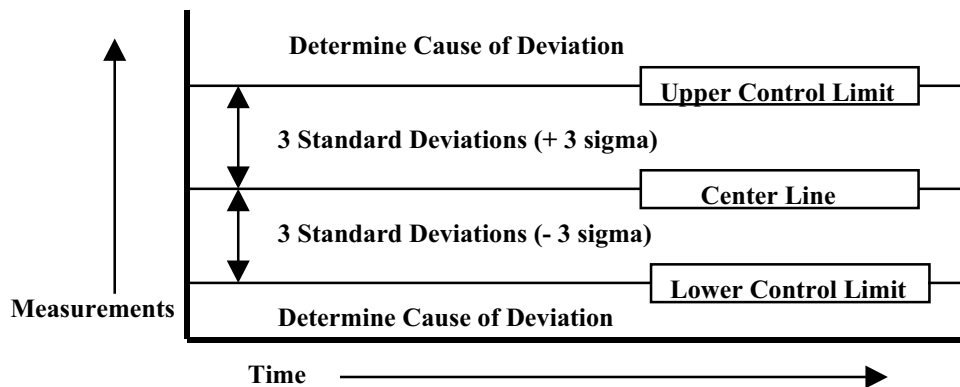


Figure 1. Control Chart

Control charts are used because they separate signal from noise, so 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.

Control charts use two types of data: variables data and attributes data. Variables data are usually measurements of continuous phenomena. Examples of variables data in software settings are elapsed time, effort expended, 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.[3]

Quantitative Analysis Flow

Figure 2 shows the Level 4 Quantitative Analysis process flow for Software Quality Management and for Quantitative Process Management.[1] When conducting quantitative analysis on project data the results can be used for both Software Quality Management and for Quantitative Process Management. If the data analyzed are defects detected, the intent is to reduce the defects during the activities that detected the defects throughout development, thus

satisfying SQM. When out of statistical control conditions occur, the reason for the anomaly is investigated and the process brought back under control which satisfies QPM.

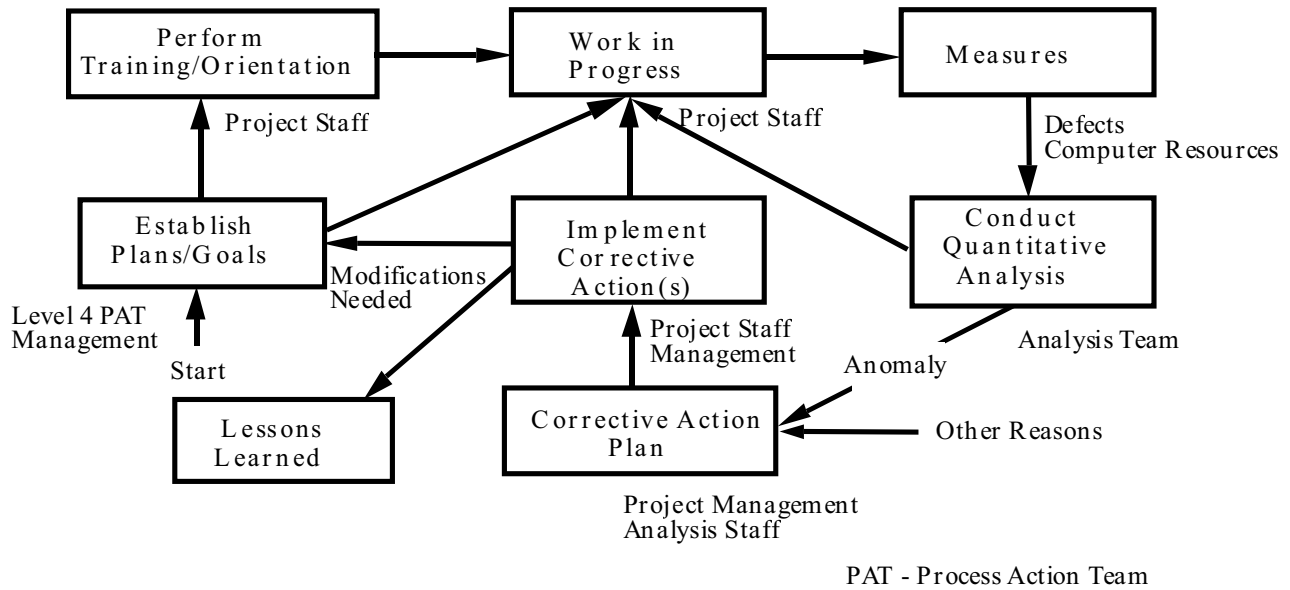


Figure 2. SQM and QPM Flow

Defect Prevention Flow

Figure 3 shows the Level 5 Defect Prevention process flow.[1]

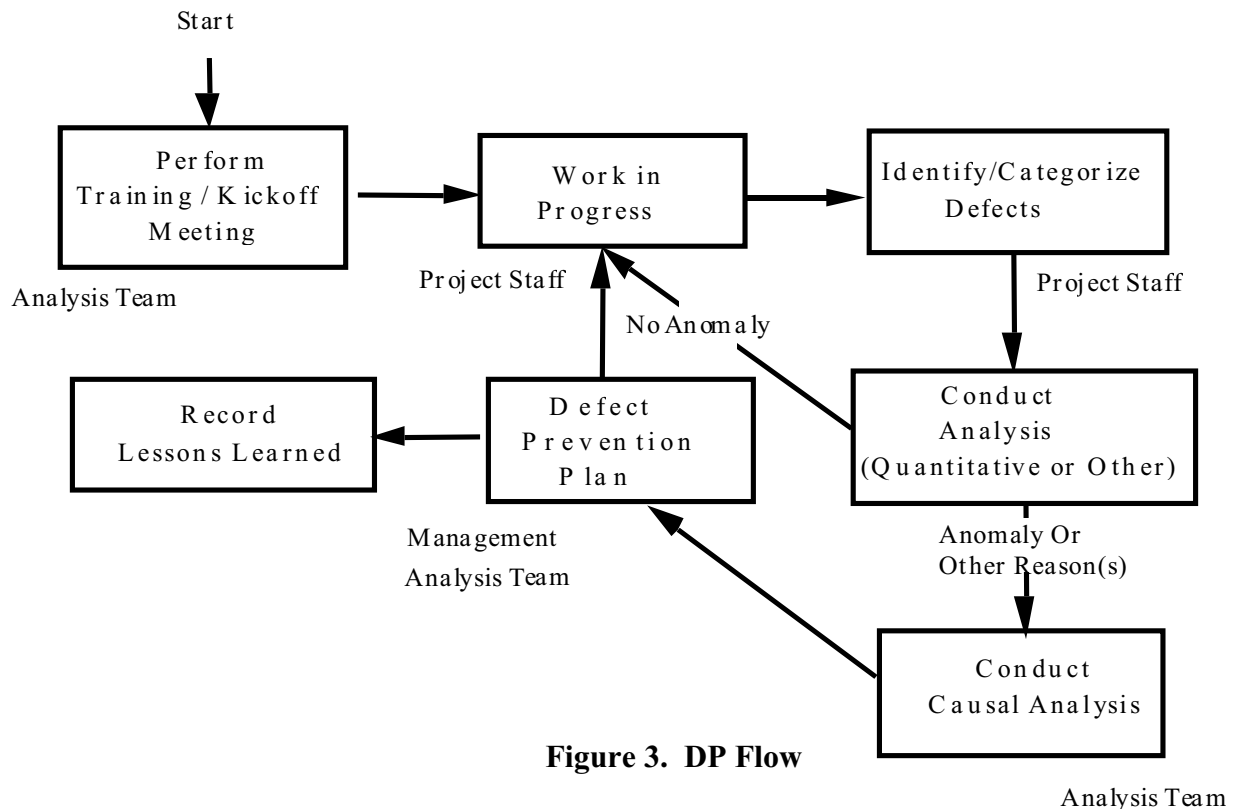


Figure 3. DP Flow

Defects can occur during any life cycle activity against any and all entities. How often do we see requirements that are without problems or schedules that are adequate or management that is sound? Defect Prevention activities are conducted on any defects that warrant prevention.

Defect prevention techniques can be applied to a variety of items:

- Project Plans
- Project Schedules
- Standards
- Processes
- Procedures
- Project Resources
- Requirements
- Documentation
- Quality Goals
- Design
- Code
- Interfaces
- Test Plans
- Test Procedures
- Technologies
- Training
- Management
- Engineering

Level 4 Leads to Level 5

Figure 4 shows how data collection, analysis and management from Level 4 activities leads to the activities at Level 5 of Defect Prevention, Technology Change Management, and Process Change Management.[5]

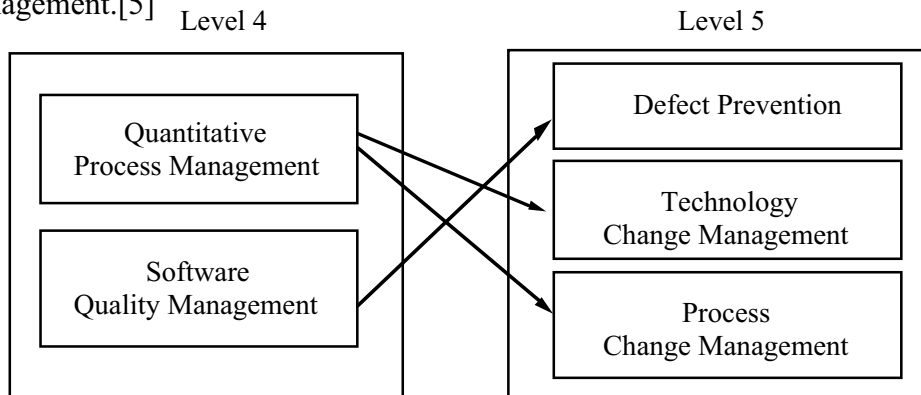


Figure 4. Level 4 and Level 5 Paths of Influence

Quantitative Process Management, which focuses on the process, leads to making process and technology improvements while Software Quality Management, which focuses on quality, leads to preventing defects.

LEVEL 4 GOALS AND PLANS

The CMM 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.[1] 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 that this paper is based on had the following key requirements:

- Timing - subject search response in less than 2.8 seconds 98% of time
- Availability - 99.86% 7 days, 24 hours (7/24)

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

Goals

The quality goals are:

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

Plans

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

Monitoring of critical computer resources:

- General purpose million instructions per second (MIPS)
- Disc storage read inputs/outputs per second (IOPS) per volume
- Write IOPS per volume
- Operational availability
- Peak response time
- Server loading

QUANTITATIVE ANALYSIS EXAMPLES

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

Example 1

Table 1 shows raw data collected at requirements peer reviews:

- Sample – series of peer reviews
- SRSs – System Requirements Specifications (requirements documents)
- No. Rqmts – number of requirements reviewed for each sample
- Defects – number of defects detected for each sample
- Defects/100 Rqmts – defects normalized to 100 requirements for each sample

Table 1. Requirements Peer Review Defects

Sample	SRSs	No. Rqmts	Defects	Defects/100 Rqmts
1. UTL	1	152	5	3.28
2. APP	1	37	4	10.81
3. HMI	1	350	101	28.85
4. MSP	1	421	13	3.8
5. EKM	1	370	25	6.75
6. CMS	1	844	60	7.10
Totals	6	2174	208	

The formulas for constructing the control chart follow.[3] The control chart used is a U-chart.

- Defects/100 Rqmts = Number of Defects * 100/Rqmts reviewed per sample (calculated for each sample). These are plotted as Plot.
- CL = total number of defects/total number of requirements reviewed * 100
- a(1) = requirements reviewed/100 (calculated for each sample)
- UCL = CL+3(SQRT(CL/a(1))) (calculated for each sample)
- LCL = CL-3(SQRT(CL/a(1))) (calculated for each sample)

The calculations are shown in Table 2. Whenever the LCL is negative, it is set to zero.

- PLOT – defects per 100 requirements is the plot on the chart
- CL – center line, an average
- a(1) – a variable calculated for each sample
- UCL–upper control limit calculated for each sample
- LCL– lower control limit calculated for each sample

Table 2. Calculations for Requirements Peer Review Defects

Sample	PLOT	CL	UCL	LCL	a(1)
1. UTL	3.28	9.56	17.09	2.04	1.52
2. APP	10.81	9.56	24.82	0	0.37
3. HMI	28.85	9.56	14.52	4.60	3.5
4. MSP	3.08	9.56	14.09	5.04	4.21
5. EKM	6.75	9.56	14.39	4.74	3.7
6. CMS	7.10	9.56	12.76	6.37	8.4

The control chart is shown in Figure 5.

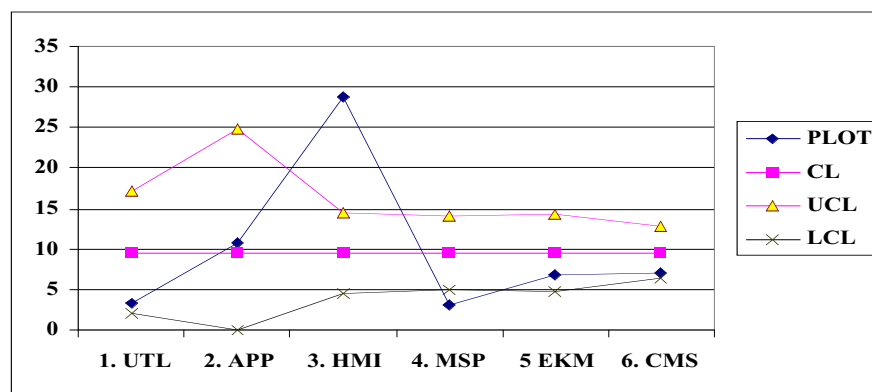


Figure 5. Control Chart for Requirements Peer Review Defects

An anomaly occurred in the third sample. Causal analysis revealed that data for that sample were for human machine interface specifications, all others were applications. Control charts

require similar data for similar processes, i.e., apples to apples analogy. The HMI sample was removed and the data charted again as shown in Figure 6.

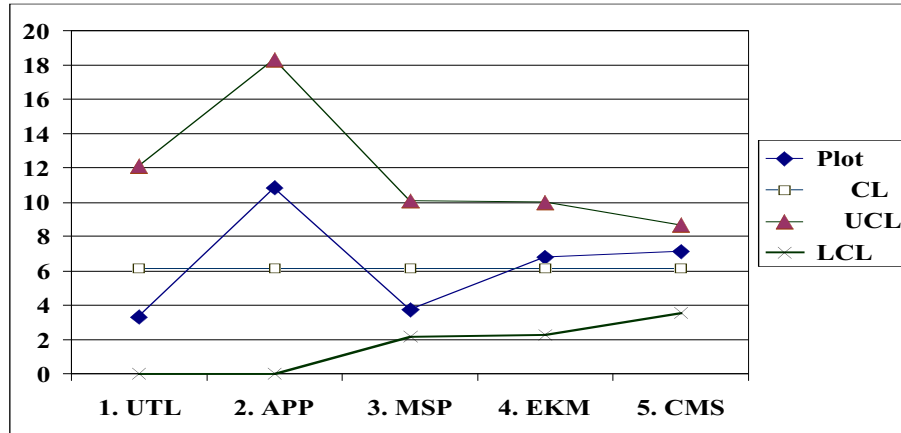


Figure 6. Control Chart for Requirements Peer Review Defects without HMI

The process is now under statistical process control. The root cause is that data gathered from dissimilar activities cannot be used on the same statistical process on control charts. The process for defining and implementing human machine interfaces is different for that of defining and implementing applications code as are the teams and methodologies. The defect prevention is against the process of collecting data for SPC control charts.

Example 2

Table 3 shows raw data collected during code peer reviews.

Table 3. Code Peer Review Defects

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	10434	425	

The calculations are shown in Table 4.

Table 4. 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

The control chart is shown in Figure 7.

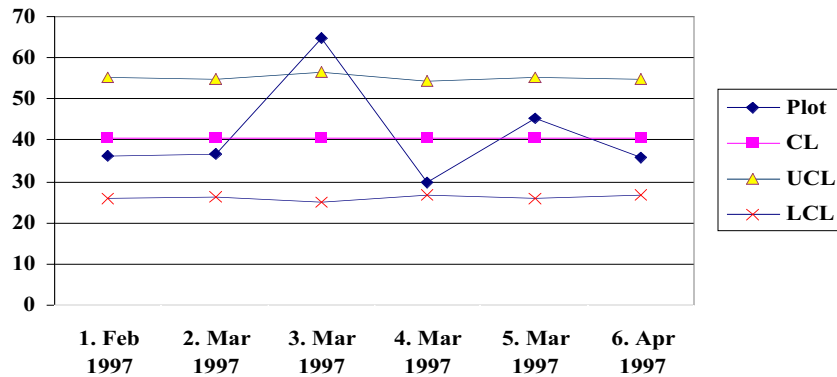


Figure 7. Control Chart for Code Peer Review Defects

The process is out of statistical process control in the third event. Causal analysis revealed that 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 and the defect prevention is to provide training whenever a new process or technology is introduced.

Example 3

Table 5 shows raw data collected at code peer reviews over a period of months:

- Units – units of code reviewed for each sample
- SLOC – number of source lines of code (SLOC) review for each sample
- Defects/KSLOC – defects normalized to 1000 lines of code for each sample

Table 5. Code Peer Review Defects

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	3582	72	

Table 6 shows the calculations.

Table 6. Calculations for Code Peer Review Defects

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

The control chart is shown in Figure 8.

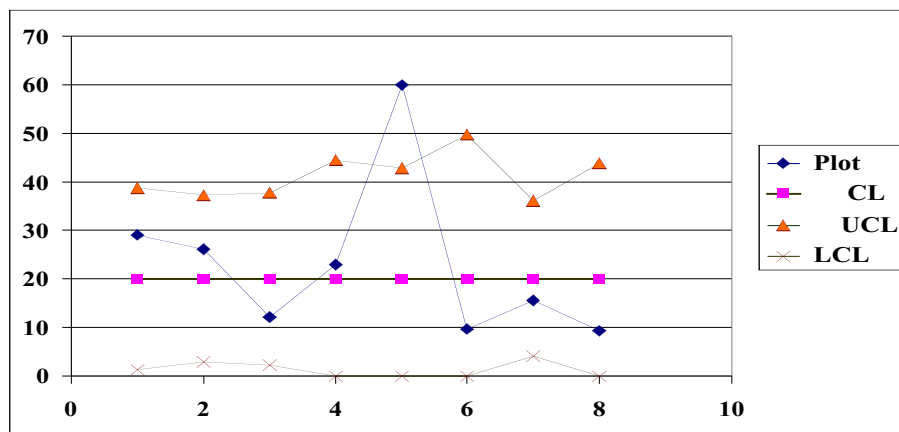


Figure 8. Control Chart for Code Peer Review Defects

An anomaly occurred in the fifth sample. Causal analysis revealed that data for that sample were for database code, all others were applications code. Again, control charts require similar data for similar processes. The database sample was removed and the data charted again as shown in

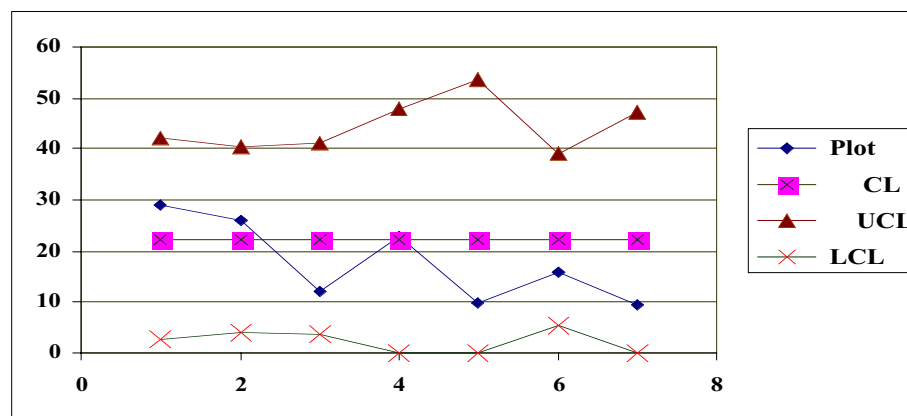


Figure 9. Control Chart without Database Defects

The process is now under statistical process control. Again, 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. Again, the defect prevention is against the process of collecting data for SPC control charts.

Example 4

During integration thread tests, 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 7.

Table 7. Thread Test’s Defects

Samples	Test Plan	Test Data	Logic	Interface	Standards	Design	Requirements
1			6	1		1	
2							
3		1	4				
Totals	0	1	10	1	0	1	0

Bar charts were used in Figure 10 to show defects discovered during integration thread tests.

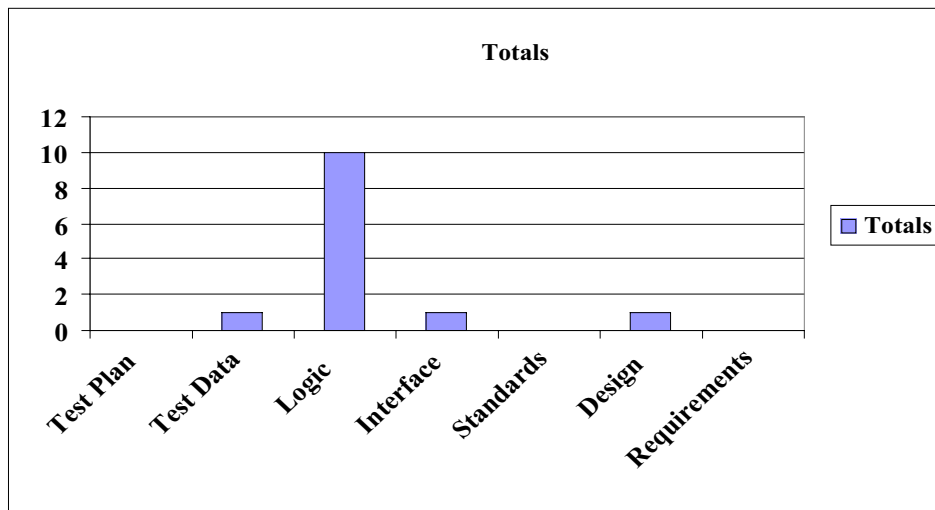


Figure 10. Thread Test Defects Bar Chart

No analysis was required since code logic defects are expected during thread tests.

Example 5

Table 8 shows defects from another series of thread tests.

Table 8. Thread Test's Defects

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						2	
10			6				
11	1	1					
12		10					
13		9	1				
14		6	1	1			
15		5	7				
16		2	1				
Totals	6	102	55	1		2	

Figure 11 plots the defects discovered during integration thread tests.

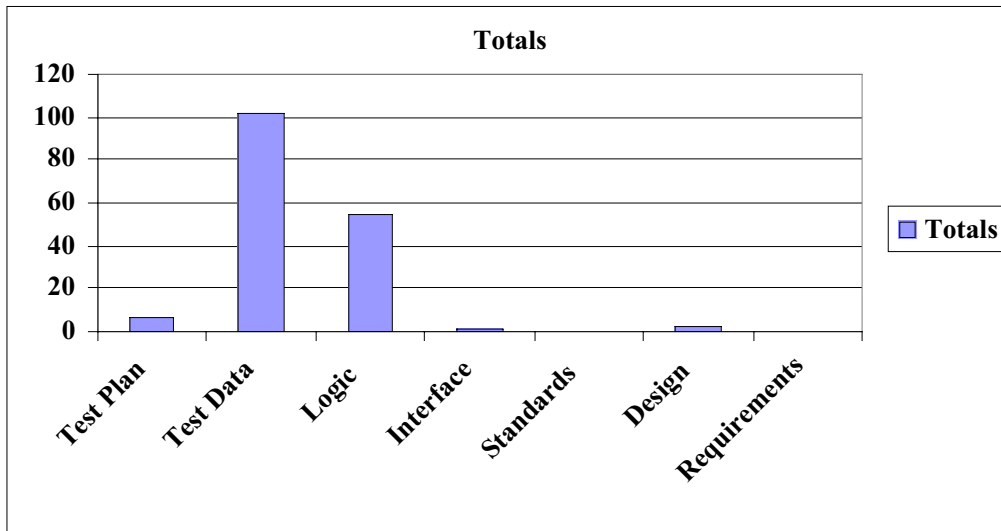


Figure 11. Thread Test Defects Bar Chart

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 6

During preliminary design and prior to acquiring hardware, a simulated performance model was used to monitor critical computer resources. Figure 12 shows some results of monitoring the required million instructions per second (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 which requires remedial action.

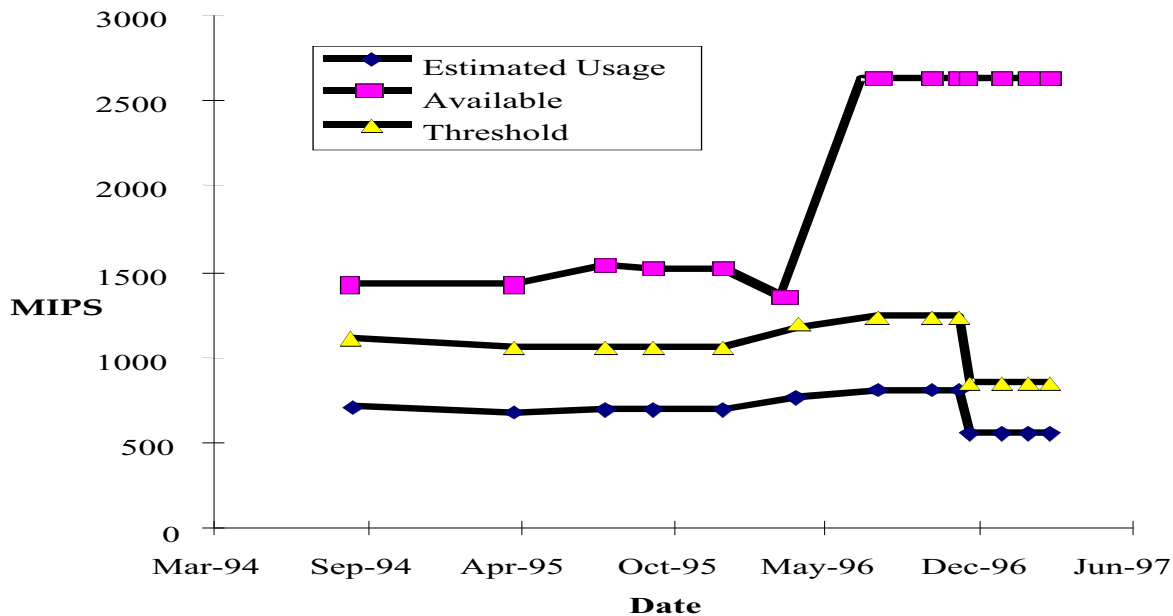


Figure 12. General Purpose MIPS

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.

CONCLUSION

The use of rigorous statistics using statistical process control (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, will improve the productivity of development and the quality of products.

REFERENCES AND SUGGESTED READINGS

1. Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, Charles V. Weber, February 1993, *Capability Maturity Model for Software*, V1.1, Software Engineering Institute
2. John H. Baumert, Mark S. McWhinney, 1992, *Software Measures and the Capability Maturity Model*, Software Engineering Institute
3. William A. Florac, Robert E. Park, Anita D. Carleton, SEI, April, 1997, *Practical Software Measurement: Measuring for Process Management and Improvement*, Software Engineering Institute
4. Thomas Pyzdek, 1984, *An SPC Primer*, Quality America, Inc.
5. Ron Radice , 1997, *Getting to Level 4 in the CMM*, The 1997 SEI Software Engineering Process Group Conference, San Jose, CA
6. Anita Carleton, Mark C. Paulk, 1997, *Statistical Process Control for Software*, The 1997 Software Engineering Symposium, Pittsburgh, PA
7. David S. Chambers & Donald J. Wheeler, 1995, *Understanding Statistical Process Control*, SPC Press
8. *Juran's Quality Control Handbook*, 1988, 4th Edition, McGraw-Hill Book Company
9. Donald J. Wheeler, 1993, *Understanding Variation, The Key to Managing Chaos*, SPC Press
10. Donald J. Wheeler, 1995, *Advanced Topics in Statistical Process Control*, SPC Press
11. W. Edwards Deming, November 1975, *On Probability As a Basis For Action*, The American Statistician, Vol. 29, No.4 (146-152)
12. Gerald J. Hahn & William Q. Meeker, February 1993, *Assumptions for Statistical Inference*, The American Statistician, Vol. 47, No.1 (1-11)
13. Watts S. Humphrey, September 1997, *Managing the Software Process*, SEI Series in Software Engineering, Addison-Wesley Publishing Company
14. William A. Florac, Anita D. Carleton, 1999, *Measuring the Software Process, Statistical Process Control for Software Process Improvement*, Software Engineering Institute