SPC for Software Reliability: Imperfect Software Debugging Model

Dr. Satya Prasad Ravi[1], N.Supriya[2] and G.Krishna Mohan[3]

**[1] Associate Professor, Dept. of Computer Science & Engg., Acharya Nagrjuna University
Nagarjuna Nagar, Guntur, Andhrapradesh, India**

**[2] Assistant Professor, Dept. of Computer Science, Adikavi Nannaya university
Rajahmedndry-533105, Andhrapradesh,India**

**[3] Reader, Dept. of Computer Science, P.B.Siddhartha college
Vijayawada, Andhrapradesh, India**

Abstract

Software reliability process can be monitored efficiently by using Statistical Process Control (SPC). It assists the software development team to identify failures and actions to be taken during software failure process and hence, assures better software reliability. In this paper, we consider a software reliability growth model of Non-Homogenous Poisson Process (NHPP) based, that incorporates imperfect debugging problem. The proposed model utilizes the failure data collected from software development projects to analyze the software reliability. The maximum likelihood approach is derived to estimate the unknown point estimators of the model. We investigate the model and demonstrate its applicability in the software reliability engineering field.

Keywords: *Statistical Process Control, Software reliability, mean value function, imperfect debugging, Probability limits, Control Charts.*

1. Introduction

Software reliability assessment is important to evaluate and predict the reliability and performance of software system, since it is the main attribute of software. To identify and eliminate human errors in software development process and also to improve software reliability, the Statistical Process Control concepts and methods are the best choice. SPC concepts and methods are used to monitor the performance of a software process over time in order to verify that the process remains in the state of statistical control. It helps in finding assignable causes, long term improvements in the software process. Software quality and reliability can be achieved by eliminating the causes or improving the software process or its operating procedures [6].

The most popular technique for maintaining process control is control charting. The control chart is one of the seven tools for quality control. Software process control is used to secure the quality of the final product which will conform to predefined standards. In any process, regardless of how carefully it is maintained, a certain amount of natural variability will always exist. A process is said to be statistically "in-control" when it operates with only chance causes of variation. On the other hand, when assignable causes are present, then we say that the process is statistically "out-of-control."

The control charts can be classified into several categories, as per several distinct criteria. Depending on the number of quality characteristics under investigation, charts can be divided into univariate control charts and multivariate control charts. Furthermore, the quality characteristic of interest may be a continuous random variable or alternatively a discrete attribute. Control charts should be capable to create an alarm when a shift in the level of one or more parameters of the underlying distribution or a non-random behavior occurs. Normally, such a situation will be reflected in the control chart by points plotted outside the control limits or by the presence of specific patterns. The most common non-random patterns are cycles, trends, mixtures and stratification [7]. For a process to be in control the control chart should not have any trend or nonrandom pattern.

SPC is a powerful tool to optimize the amount of information needed for use in making management decisions. SPC provides real time analysis to establish controllable process baselines; learn, set, and dynamically improves process capabilities; and focus business areas

which need improvement. The early detection of software failures will improve the software reliability. The selection of proper SPC charts is essential to effective statistical process control implementation and use. The SPC chart selection is based on data, situation and need [4].
The control limits can then be utilized to monitor the failure times of components. After each failure, the time can be plotted on the chart. If the plotted point falls between the calculated control limits, it indicates that the process is in the state of statistical control and no action is warranted. If the point falls above the UCL, it indicates that the process average, or the failure occurrence rate, may have decreased which results in an increase in the time between failures. This is an important indication of possible process improvement. If this happens, the management should look for possible causes for this improvement and if the causes are discovered then action should be taken to maintain them. If the plotted point falls below the LCL, It indicates that the process average, or the failure occurrence rate, may have increased which results in a decrease in the failure time. This means that process may have deteriorated and thus actions should be taken to identify and causes may be removed. It can be noted here that the parameter a, b should normally be estimated with the data from the failure process.
The control limits for the chart are defined in such a manner that the process is considered to be out of control when the time to observe exactly one failure is less than LCL or greater than UCL. Our aim is to monitor the failure process and detect any change of the intensity parameter. When the process is normal, there is a chance for this to happen and it is commonly known as false alarm. The traditional false alarm probability is to set to be 0.27% although any other false alarm probability can be used. The actual acceptable false alarm probability should in fact depend on the actual product or process [9].

2. Literature Survey

This section presents the theory that underlies a distribution and maximum likelihood estimation for complete data. If ‘t’ is a continuous random variable with pdf: $f(t;\theta_1,\theta_2,\ldots,\theta_k)$. where $\theta_1,\theta_2,\ldots,\theta_k$ are k unknown constant parameters which need to be estimated, and cdf: $F(t)$. Where, The mathematical relationship between the pdf and cdf is given by: $f(t) = \frac{d(F(t))}{dt}$. Let ‘a’ denote the expected number of faults that would be detected given infinite testing time in case of finite failure NHPP models. Then, the mean value function of the finite failure NHPP models can be written as: $m(t) = aF(t)$, where F(t) is a cumulative distribution function. The failure intensity function $\lambda(t)$ in case of the finite failure NHPP models is given by: $\lambda(t) = aF'(t)$.

2.1 NHPP model

The Non-Homogenous Poisson Process (NHPP) based software reliability growth models (SRGMs) are proved quite successful in practical software reliability engineering [3]. The main issue in the NHPP model is to determine an appropriate mean value function to denote the expected number of failures experienced up to a certain time point. Model parameters can be estimated by using Maximum Likelihood Estimate (MLE).
Various NHPP SRGMs have been built upon various assumptions. Many of the SRGMs assume that each time a failure occurs, the fault that caused it can be immediately removed and no new faults are introduced. Which is usually called perfect debugging. Imperfect debugging models have proposed a relaxation of the above assumption [8].
Let $\{N(t), t \geq 0\}$ be the cumulative number of software failures by time ‘t’. m(t) is the mean value function, representing the expected number of software failures by time ‘t’. $\lambda(t)$ is the failure intensity function, which is proportional to the residual fault content. Thus $m(t) = a(1-e^{-bt})$ and $\lambda(t) = \frac{dm(t)}{dt} = b(a - m(t))$.
Where ‘a’ denotes the initial number of faults contained in a program and ‘b’ represents the fault detection rate. In software reliability, the initial number of faults and the fault detection rate are always unknown. The maximum likelihood technique can be used to evaluate the unknown parameters. In NHPP SRGM $\lambda(t)$ can be expressed in a more general way as $\lambda(t) = \frac{dm(t)}{dt} = b(t)[a(t) - m(t)]$.
Where $a(t)$ is the time-dependent fault content function which includes the initial and introduced faults in the program and $b(t)$ is the time-dependent fault detection rate. A constant $a(t)$ implies the perfect debugging assumption, i.e no new faults are introduced during the debugging process. If we assume that, when the detected faults are removed, then there is a possibility to introduce new faults with a constant rate ‘ β ’. Then the mean value function is [2] given as $m(t) = \frac{a}{1-\beta}\left[1 - e^{-(1-\beta)bt}\right]$.

2.2 ML (Maximum Likelihood) Parameter Estimation

The idea behind maximum likelihood parameter estimation is to determine the parameters that maximize the probability (likelihood) of the sample data. The method of maximum likelihood is considered to be more robust (with some exceptions) and yields estimators with good statistical properties. In other words, MLE methods are versatile and apply to many models and to different types of data. Although the methodology for maximum likelihood estimation is simple, the implementation is mathematically intense. Using today's computer power, however, mathematical complexity is not a big obstacle. If we conduct an experiment and obtain N independent observations, $t_1, t_2, \ldots, t_N$. Then the likelihood function is given by[1] the following product:

$$L(t_1, t_2, \ldots, t_N \mid \theta_1, \theta_2, \ldots, \theta_k) = L = \prod_{i=1}^{N} f(t_i; \theta_1, \theta_2, \ldots, \theta_k)$$

Likely hood function by using λ(t) is:

$$L = \prod_{i=1}^{n} \lambda(t_i)$$

The logarithmic likelihood function is given by:

$$\Lambda = \ln L = \sum_{i=1}^{N} \ln f(t_i; \theta_1, \theta_2, \ldots, \theta_k)$$

$$\text{Log L} = \log\left(\prod_{i=1}^{n} \lambda(t_i)\right)$$

which can be written as $\sum_{i=1}^{n} \log[\lambda(t_i)] - m(t_n)$

The maximum likelihood estimators (MLE) of $\theta_1, \theta_2, \ldots, \theta_k$ are obtained by maximizing L or Λ, where Λ is ln L . By maximizing Λ, which is much easier to work with than L, the maximum likelihood estimators (MLE) of $\theta_1, \theta_2, \ldots, \theta_k$ are the simultaneous solutions of k equations such that: $\frac{\partial(\Lambda)}{\partial\theta_j} = 0$, j=1,2,…,k

The parameters 'a' and 'b' are estimated using iterative Newton Raphson Method, which is given as

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}$$

3. Illustrating the MLE Method

3.1 parameter estimation

To estimate 'a' and 'b' , for a sample of n units, first obtain the likelihood function: assuming $\beta = 0.05$.

$$L = \prod_{i=1}^{N} abe^{-(1-\beta)bt}$$

Take the natural logarithm on both sides, The Log Likelihood function is given as:

$$\text{Log L} = \log\left[\prod_{i=1}^{n} \lambda(t_i)\right] = \log\left[\prod_{i=1}^{n} abe^{-(1-\beta)bt}\right]$$

$$= \sum_{i=1}^{n} \log(abe^{-(1-\beta)bt}) - \frac{a}{1-\beta}\left[1 - e^{-(1-\beta)bt}\right]$$

Taking the Partial derivative w.r.t 'a' and equating to '0'. (i.e $\frac{\partial \log L}{\partial a} = 0$) $\quad a = \frac{n(1-\beta)}{\left(1 - e^{-(1-\beta)bt_n}\right)}$

Taking the Partial derivative w.r.t 'b' and equating to '0'. (i.e $g(b) = \frac{\partial \log L}{\partial b} = 0$)

$$g(b) = \frac{n}{b} - \left[\frac{n(1-\beta)t_n}{e^{(1-\beta)bt_n} - 1}\right] - (1-\beta)\sum_{i=1}^{n} t_i = 0$$

Taking the partial derivative again w.r.t 'b' and equating to '0'. (i.e $g'(b) = \frac{\partial^2 \log L}{\partial b^2} = 0$)

$$g'(b) = \frac{n(1-\beta)^2 t_n^2 e^{(1-\beta)bt_n}}{\left[e^{(1-\beta)bt_n} - 1\right]^2} - \frac{n}{b^2} = 0$$

The parameter 'b' is estimated by iterative Newton Raphson Method using $b_{n+1} = b_n - \frac{g(b_n)}{g'(b_n)}$. which is substituted in finding 'a'.

3.2 Distribution of Time between failures

Based on the inter failure data given in Table 1, we compute the software failures process through Mean Value Control chart. We used cumulative time between failures data for software reliability monitoring using Exponential distribution.

Table:1 Time between failures of a software

Failure Number	Time between failure(h)	Failure Number	Time between failure(h)
1	30.02	**16**	15.53
2	1.44	**17**	25.72
3	22.47	**18**	2.79
4	1.36	**19**	1.92
5	3.43	**20**	4.13
6	13.2	**21**	70.47
7	5.15	**22**	17.07
8	3.83	**23**	3.99
9	21	**24**	176.06
10	12.97	**25**	81.07
11	0.47	**26**	2.27
12	6.23	**27**	15.63
13	3.39	**28**	120.78
14	9.11	**29**	30.81
15	2.18	**30**	34.19

Assuming an acceptable probability of false alarm of 0.27%, the control limits can be obtained as [5]:

$$T_U = \frac{1}{1-\beta}\left[1 - e^{-(1-\beta)bt}\right] = 0.99865$$

$$T_C = \frac{1}{1-\beta}\left[1 - e^{-(1-\beta)bt}\right] = 0.5$$

$$T_L = \frac{1}{1-\beta}\left[1 - e^{-(1-\beta)bt}\right] = 0.00135$$

'$\hat{a}$' and '$\hat{b}$' are Maximum Likely hood Estimates (MLEs) of parameters and the values can be computed using iterative method for the given cumulative time between failures data shown in table 1. Using 'a' and 'b' values we can compute $m(t)$.

These limits are converted to $m(t_U)$, $m(t_C)$ and $m(t_L)$ form. They are used to find weather the software process is in control or not by placing the points in Mean value chart shown in Figure 1. A point below the control limit $m(t_L)$ indicates an alarming signal. A point above the control limit $m(t_U)$ indicates better quality. If the points are falling within the control limits it indicates the software process is in stable condition [6]. The values of control limits are as follows.

$$m(t_U) = 31.69529$$

$$m(t_C) = 15.86907$$

$$m(t_L) = 0.042846$$

Table:2 successive differences of cumulative mean values

No	Cum Failures	m(t)	Successive differences	No	Cum failures	m(t)	Successive differences
1	30.02	2.9599655	0.135198384	16	151.78	12.5083594	1.596922361
2	31.46	3.09516389	2.033545566	17	177.5	14.1052818	0.165718931
3	53.93	5.12870945	0.118607302	18	180.29	14.2710007	0.113215983
4	55.29	5.24731676	0.296929927	19	182.21	14.3842167	0.241267574
5	58.72	5.54424668	1.113786929	20	186.34	14.6254843	3.675758395
6	71.92	6.65803361	0.422372776	21	256.81	18.3012427	0.776290028
7	77.07	7.08040639	0.309784217	22	273.88	19.0775327	0.175623745
8	80.9	7.3901906	1.634899415	23	277.87	19.2531564	5.940013828
9	101.9	9.02509002	0.958006905	24	453.93	25.1931703	1.820658724
10	114.87	9.98309692	0.033999962	25	535	27.013829	0.044702692
11	115.34	10.0170969	0.446045517	26	537.27	27.0585317	0.299430566
12	121.57	10.4631424	0.239128039	27	552.9	27.3579622	1.884814636
13	124.96	10.7022704	0.630337689	28	673.68	29.2427769	0.378342482
14	134.07	11.3326081	0.148225045	29	704.49	29.6211194	0.379762003
15	136.25	11.4808332	1.027526243	30	738.68	30.0008814	

Figure 1 is obtained by placing the differences between cumulative failure data shown in Table 2 on y axis, failure number on x axis and the values of control limits are placed on Mean Value chart. The Mean Value chart shows that the 10th failure data has fallen below $m(t_L)$ which indicates the failure process. It is significantly early detection of failures using Mean Value Chart. The software quality is determined by detecting failures at an early stage. The remaining failure data are shown in Figure 1 are in stable. No failure data fall outside the $m(t_U)$. It does not indicate any alarm signal.

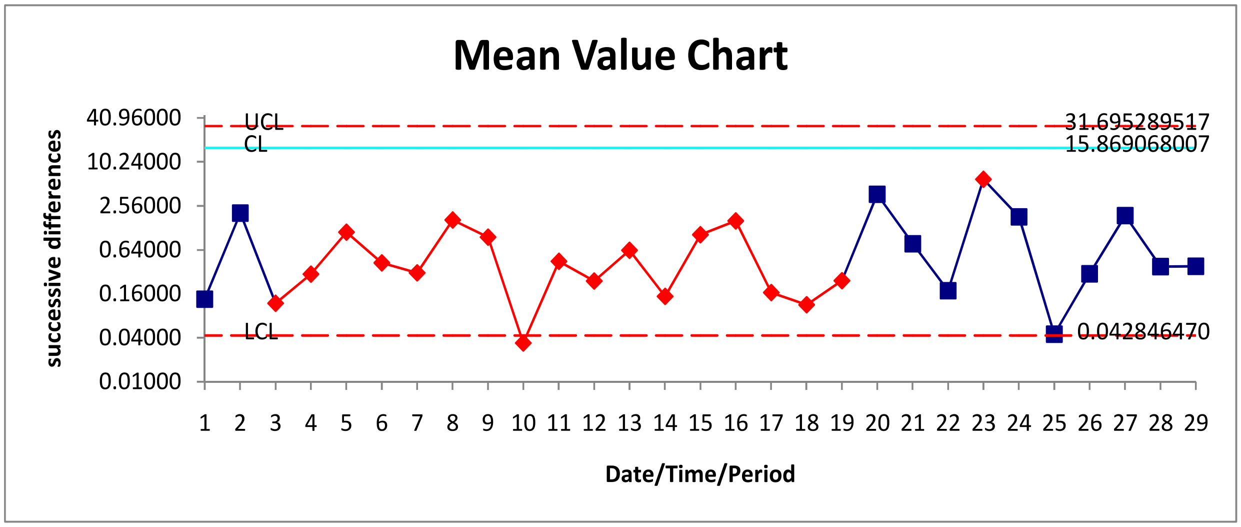

Figure: 1 Mean Value Chart

4. Conclusion

The given 30 inter failure times are plotted through the estimated mean value function against the failure serial order. The parameter estimation is carried out by Newton Raphson Iterative method for Exponential model. The graphs have shown out of control signals i.e below the LCL. Hence we conclude that our method of estimation and the control chart are giving a +ve recommendation for their use in finding out preferable control process or desirable out of control signal. By observing the Mean value Control chart we identified that the failure situation is detected at 10th point of Table-2 for the corresponding $m(t)$, which is below $m(t_L)$. It indicates that the failure process is detected at an early stage compared with Xie et a1, (2002) control chart [5], which detects the failure at 23rd point for the inter failure data above the UCL. Hence our proposed Mean Value Chart detects out of control situation at an earlier instant than the situation in time control chart. The early detection of software failure will improve the software Reliability. When the time between failures is less than LCL, it is likely that there are assignable causes leading to significant process deterioration and it should be investigated. On the other hand, when the time between failures has exceeded the UCL, there are probably reasons that have lead to significant improvement.

References

[1] Hoang Pham, "Handbook Of Reliability Engineering", Springer: Apr 2003, edition1.
[2] Huan-Jyh Shyur "A stochastic software reliability model with imperfect-debugging and change-point"- The journal of systems and software 66 (2003) 135-141.
[3] J.D. Musa., A. Iannino., K. Okumoto., 1987. "Software Reliability: Measurement Prediction Application". McGraw-Hill, New York.
[4] J. F. MacGregor and T. Kourti "Statistical process control of multivariate processes".
[5] M Xie, T.N Goh, P.Ranjan "Some effective control chart procedures for reliability monitoring" -Reliability engineering and System Safety 77 143 -150¸ 2002.
[6] M. Kimura, S. Yamada, S. Osaki."Statistical Software reliability prediction and its applicability based on mean time between failures".
[7] M.V.Koutras, S.Bersimis, P.E.Maravelakis "Statistical process control using shewart control charts with supplementary Runs rules" Springer Science + Business media 9:207-224, 2007.
[8] M.Ohba "Software Reliability Analysis Models" IBM Journal Research Development Vol.29,No. 4, pp. 428-442, July 1984.
[9] Swapna S. Gokhale and Kishore S.Trivedi, 1998. "Log-Logistic Software Reliability Growth Model". The 3rd IEEE International Symposium on High-Assurance Systems Engineering. IEEE Computer Society.

Author's profile:

First Author
Dr. R. Satya Prasad received Ph.D. degree in Computer Science in the faculty of Engineering in 2007 from Acharya Nagarjuna University, Andhra Pradesh. He received gold medal from Acharya Nagarjuna University for his out standing performance in Masters Degree. He is currently working as Associate Professor and H.O.D, in the Department of Computer Science & Engineering, Acharya Nagarjuna University. His current research is focused on Software Engineering. He has published several papers in National & International Journals.

Second Author
Mrs. N.Supriya Working as Assistant professor in the department of Computer Science, Adikavi Nannaya university, Rajahmedndry. She is at present pursuing Ph.D at Acharya Nagarjuna University. Her research interests lies in Softwrare Engineering and Data Mining.

Third Author

Mr. G. Krishna Mohan is working as a Reader in the Department of Computer Science, P.B.Siddhartha College, Vijayawada. He obtained his M.C.A degree from Acharya Nagarjuna University in 2000, M.Tech from JNTU, Kakinada, M.Phil from Madurai Kamaraj University and pursuing Ph.D at Acharya Nagarjuna University. His research interests lies in Data Mining and Software Engineering.