An Efficient Defect Estimation Method for Software Defect Curves [1]

Chenggang Bai
Department of Automatic Control, Beijing University of Aeronautics and Astronautics
Beijing 100083, China
bcg@buaa.edu.cn

Kai-Yuan Cai
Department of Automatic Control, Beijing University of Aeronautics and Astronautics
Beijing 100083, China
kycai@buaa.edu.cn

T.Y. Chen
School of Information Technology, Swinburne University of Technology
Hawthorn 3122, Australia
tchen@it.swin.edu.au

Abstract

Software defect curves describe the behavior of the estimate of the number of remaining software defects as software testing proceeds. They are of two possible patterns: single-trapezoidal-like curves or multiple-trapezoidal-like curves. In this paper we present some necessary and/or sufficient conditions for software defect curves of the Goel-Okumoto NHPP model. These conditions can be used to predict the effect of the detection and removal of a software defect on the variations of the estimates of the number of remaining defects. A field software reliability dataset is used to justify the trapezoidal shape of software defect curves and our theoretical analyses. The results presented in this paper may provide useful feedback information for assessing software testing progress and have potentials in the emerging area of software cybernetics that explores the interplay between software and control.

1. Introduction

Software defects play a key role in software reliability study. Therefore, it is very important to study the properties of software defects. As pointed out by Yourdon [1], "Defects are not the only measure of quality, of course; but they are the most visible indicator of quality throughout a project". The problem has been studied by many researchers [2-4]. Many early studies of defect occurrence suggested that it followed a Rayleigh curve [5, 6], roughly proportional to project staffing. The underlying assumption is that the more effort expended, the more mistakes are made. McConnell [7] has discussed the relationship between defect rate and development time. In his observations, projects that achieve the lowest defect rates also achieve the shortest schedules. Since software testing is the immediate phase prior to the release of software, it will be most interesting to know more about the relationship between the number of remaining software defects and testing process. Although there have been some experimental curves to depict the relationship between the number of remaining software defects and testing process, a thorough and rigorous analysis has not yet been conducted. In our previous investigation [8], we have presented the notion of software defect curve to depict such a relationship. This paper is a follow-up of our previous investigation.

Software defect curves describe the behavior of the estimate of the number of remaining software defects as software testing proceeds. Figure 1 shows a typical pattern of software defect curves under a time-homogenous test profile. In Figure 1, n denotes the number of software defects detected and removed, and $\hat{M}_n$ denotes the estimate of the number of defects remaining in the software after n defects are detected and removed. In the early testing phase, as more and more software defects are detected and removed, the estimate of the number of remaining defects tends to increase. In the middle testing phase, the estimate of the remaining defects remains steady. In the late testing phase, the estimate of the number of remaining defects tends to decrease.

If the test profile is time-nonhomogenous, changing from one to another, then the expected trend of software defect curves demonstrates a multiple-trapezoidal-like curve as shown in Figure 2.

The trapezoidal shape of software defect curves has been recently proposed [8]. It was motivated mainly by the question: as software testing proceeds and software reliability tends to grow, whether or not the estimate of the number of remaining software defects tends to

[1] Supported by the National Natural Science Foundation of China (60233020), the Aviation Science Foundation of China (01F51025) and the “863” Program of China (2001AA113192).

decrease. This question was triggered by the empirical observation that, under certain circumstances, the estimate of the initial number of software defects tends to increase as software testing proceeds[2].

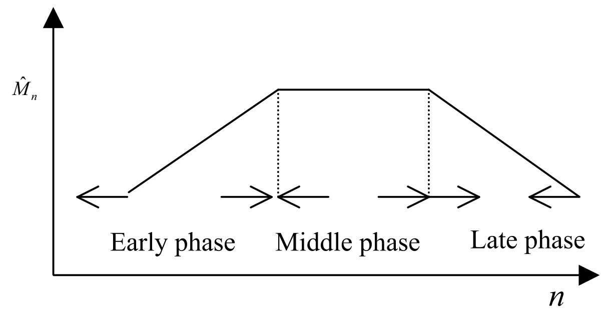

Figure 1 **Expected trend of software defect curves under a single test profile**

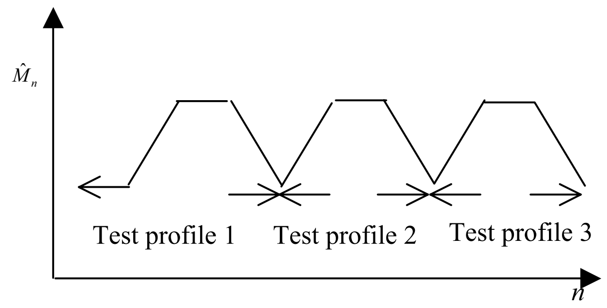

Figure 2 **Expected trend of software defect curves under multiple test profiles**

Obviously, software defect curves help to answer the questions. A considerable amount of theoretical analyses and simulation studies have been conducted to justify the trapezoidal shape of software defect curve and its potential applications [8]. Software defect curves may help to detect changes in software test profile, identify stages of software testing, improve software testing strategies, and so on. In order to do so, it is necessary to analyze the actual trend of a software defect curve. In particular, we need to characterize the relationship between $\hat{M}_n$ and $\hat{M}_{n+1}$ (refer to Figure 1). In this paper we present some necessary and/or sufficient conditions for the relationship. These conditions are easily verifiable. These results also supplement our previous theoretical analyses of software defect curves [8].

The rest of the paper is organized as follows. Section 2 reviews the Goel-Okumoto NHPP model of software reliability. Section 3 presents necessary and/or sufficient conditions for software defect curves under the Goel-Okumoto NHPP model. In Section 4 presents a software defect curve based on Musa's software reliability data. Concluding remarks are presented in Section 5.

2. The Goel-Okumoto NHPP model

The Goel-Okumoto NHPP model is one of the most important models in software reliability engineering. It is based on the following assumptions [9].

(1). Software is tested under anticipated operating environment.

(2). For any set of finite time instants $t_1, t_2, \quad, t_n$, the numbers of software failures $\eta_1, \eta_2, \quad, \eta_n$ observed in the time intervals $(0,t_1], (t_1,t_2], \quad, (t_{n-1},t_n]$ respectively, are independent.

(3). Every software defect has an equal chance of being detected.

(4). The accumulative number of software failure observed up to time *t*, $N(t)$, follows a Poisson distribution with the mean value $m(t)$ such that the number of software failures observed in the interval $(t, t+\Delta t)$ is proportional to the interval length and to the number of remaining software defects at time *t*.

(5). $m(t)$ is a bounded and non-decreasing function with

$m(t)$=0, as $t \to 0$, and $m(t) \to a$, as $t \to \infty$

where a is the total number of software failures eventually observed.

With these assumptions, we have

$$m(t) = a\,\{1 - \exp(-bt)\}$$

In this way

$$\Pr\{N(t_i) - N(t_{i-1}) = \eta_i\} = \frac{[m(t_i) - m(t_{i-1})]^{\eta_i} \exp\{m(t_{i-1}) - m(t_i)\}}{\eta_i!}$$

Suppose that there is an one-to-one correspondence between failure and defect, and each failure-causing defect is removed immediately upon the detection of a failure, without introducing new defects. Then, the number of software failures is equal to the number of software defects. Thus the Goel-Okumoto NHPP model can be used to estimate the number of software defect [2]. Originally in the Goel-Okumoto NHPP model $\{t_1, t_2, \quad, t_n\}$ can be any observation time instants given a priori and the model deals with $\{\eta_1, \eta_2, \quad, \eta_n\}$ which are classified as the type II data[2] [10]. It should be noted that Goel-Okumoto NHPP model deals with both

[2] Type I data mean that software reliability data are represented in terms of the time intervals between successive software failures. Type II data mean that software reliability data are represented in terms of the numbers of failures observed in successive time intervals.

type I and type II data. Since type I data is more accurate than type II data, we use type I data in this study.

Here we still treat t_i as the time instant of the *i*th software failure. Let the software failure process be represented in Figure 3, where $\{X_i : i = 1, \quad, n\}$ is a series of time intervals between successive software failures with $X_i = T_i - T_{i-1}$.

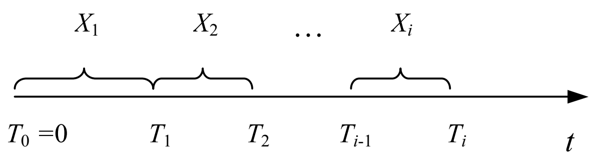

Figure 3 **Software failure process**

Suppose $X_1, X_2, \quad, X_n$ are independent and t_i is realization of T_i. Then the joint density distribution function of $T_1, T_2, \quad, T_n$ can be determined by[3]

$$L(t_1, t_2, \quad, t_n) = (ab)^n \exp\{-b\sum_{i=1}^{n} t_i\} \exp\{-a(1 - e^{-bt_n}\}$$

Hence, the maximum likelihood estimates of *a* and *b* are determined by the following equations

$$\hat{a}_n = \frac{n}{1 - e^{-\hat{b}_n t_n}}$$
(1)

$$\frac{n}{\hat{b}_n} = \hat{a}_n t_n e^{-\hat{b}_n t_n} + \sum_{i=1}^{n} t_i$$
(2)

Here we are more interested in $\hat{M}_n$ than $\hat{a}_n$, with

$$\hat{M}_n = \hat{a}_n - n$$
(3)

Let $z = \hat{b}_n t_n$ and $P_n = \frac{\sum_{i=1}^{n} t_i}{n t_n}$, the followings are after equations (1)-(3)

$$\hat{M}_n = \frac{n}{e^z - 1}$$
(4)

$$P_n = \frac{1}{z} - \frac{1}{e^z - 1} \qquad (5)$$

Let $z(w)$ be the inverse function of $w = \frac{1}{z} - \frac{1}{e^z - 1}$, and $h(w) = \frac{1}{e^{z(w)} - 1}$. In [8], we have proved the following proposition.

Proposition 1 $\hat{M}_{n+1} \sim \hat{M}_n$ iff $\bullet n+1 \bullet h(P_{n+1}) \sim n\, h(P_n)$, where $\sim$ can be $>$, $=$, or $<$. **Q.E.D.**

In the next section, we will present another necessary and sufficient condition for software defect curves, which is easier to be verified.

3. Necessary and sufficient conditions

From equations (4) and (5), we note

$$P_n = \frac{1}{\ln(1 + \frac{n}{\hat{M}_n})} - \frac{\hat{M}_n}{n}$$

Thus, we have $\lim_{\hat{M}_n \to 0} P_n = 0$

Let $y = f(x) = \begin{cases} \frac{1}{\ln(1 + \frac{1}{x})} - x & x > 0 \\ 0 & x = 0 \end{cases}$

Lemma 1 $y = f(x)$ is continuous and strictly increasing on $[0, \infty)$.

Proof It is obvious that $y = f(x)$ is continuous on $[0, \infty)$.

Furthermore, for $x \in (0, \infty)$, $y = f(x)$ is equivalent to

$$\begin{cases} x = \frac{1}{e^{\xi} - 1} \\ y = \frac{1}{\xi} - \frac{1}{e^{\xi} - 1} = \frac{1}{\xi} - x \end{cases}$$
(6)

So, $\frac{dy}{dx} = -\frac{1}{\xi^2}\frac{d\xi}{dx} - 1 = \frac{1}{\xi^2} \cdot \frac{(e^{\xi} - 1)^2}{e^{\xi}} - 1$

It is easy to prove that $\frac{1}{\xi^2} \cdot \frac{(e^{\xi} - 1)^2}{e^{\xi}} - 1 > 0$, for $\xi > 0$.

Therefore $y = f(x)$ is strictly increasing for $x \in (0, \infty)$.

Finally, $f(0) = 0$, and $y = f(x) > 0$ for $x \in (0, \infty)$ because $\ln(1 + \frac{1}{x}) < \frac{1}{x}$. Hence, we can conclude that $y = f(x)$ is strictly increasing on $x \in [0, \infty)$. **Q.E.D.**

Corollary 1 $x = k(y) = f^{-1}(y)$ is continuous and strictly increasing on $[0, \frac{1}{2})$.

[3] The joint density distribution function can be obtained from most books, e.g. [11], about non-homogeneous Poisson process.

Proof Note $f(0)=0$ and $\lim\limits_{x\to\infty}\frac{1}{\ln(1+\frac{1}{x})}-x=\frac{1}{2}$.

Therefore, we have $y\in[0,\frac{1}{2})$, and $x=k(y)$ is continuous and strictly increasing on $[0,\frac{1}{2})$ as a result of Lemma 1. **Q.E.D.**

Corollary 1 $x=k(y)=f^{-1}(y)$ is continuous and strictly increasing on $[0,\frac{1}{2})$.

Proof Note $f(0)=0$ and $\lim\limits_{x\to\infty}\frac{1}{\ln(1+\frac{1}{x})}-x=\frac{1}{2}$.

Therefore, we have $y\in[0,\frac{1}{2})$, and $x=k(y)$ is continuous and strictly increasing on $[0,\frac{1}{2})$ as a result of Lemma 1. **Q.E.D.**

Lemma 2 For $x\in(0,\infty)$, $\frac{\mathrm{d}^2y}{\mathrm{d}x^2}<0$.

Proof Refer to equation (6).

$$\frac{\mathrm{d}^2y}{\mathrm{d}x^2}=(\frac{\mathrm{d}^2y}{\mathrm{d}\xi^2}-\frac{\mathrm{d}y}{\mathrm{d}\xi}.\frac{\mathrm{d}^2x}{\mathrm{d}\xi^2}.\frac{\mathrm{d}\xi}{\mathrm{d}x}).(\frac{\mathrm{d}\xi}{\mathrm{d}x})^2$$
$$=\frac{2e^{\xi}-\xi e^{\xi}-\xi-2}{\xi^3(e^{\xi}-1)}(\frac{\mathrm{d}\xi}{\mathrm{d}x})^2$$

It is easy to verify that $2e^{\xi}-\xi e^{\xi}-\xi-2<0$ for $\xi>0$. This completes the proof. **Q.E.D.**

Lemma 3 $k(y)$ is a strictly convex function over $[0,\frac{1}{2})$.

Proof First,

$$\frac{\mathrm{d}^2x}{\mathrm{d}y^2}=\frac{\mathrm{d}}{\mathrm{d}y}(\frac{\mathrm{d}x}{\mathrm{d}y})=-\frac{\mathrm{d}x}{\mathrm{d}y}.\frac{\mathrm{d}^2y}{\mathrm{d}x^2}/(\frac{\mathrm{d}y}{\mathrm{d}x})^2>0$$

Hence, $k(y)$ a strictly convex function over $[0,\frac{1}{2})$.

Furthermore, we have

$$k^{/}(0)=\lim_{y\to0}\frac{k(y)-k(0)}{y-0}=\lim_{y\to0}\frac{x}{y}=\lim_{y\to0}\frac{\mathrm{d}x}{\mathrm{d}y}=\lim_{\xi\to\infty}\frac{\mathrm{d}x}{\mathrm{d}y}$$
$$=\lim_{\xi\to\infty}\frac{\xi^2e^{\xi}}{(e^{\xi}-1)^2-\xi^2e^{\xi}}=0$$
$$k^{//}(0)=\lim_{y\to0}\frac{\frac{\mathrm{d}x}{\mathrm{d}y}-k^{/}(0)}{y-0}=0$$

This implies that $k(y)$ is a convex function defined on $[0,\frac{1}{2})$, that is, $\forall y\in[0,\frac{1}{2})$.

$\lambda\in(0,1)$, there holds

$k((1-\lambda)0+\lambda y)\le(1-\lambda)k(0)+\lambda k(y)$

or $k(\lambda y)\le\lambda k(y)$

In order to show that $k(y)$ is a strictly convex function defined on $[0,\frac{1}{2})$, we need to show $k(\lambda y)\ne\lambda k(y)$ for $\forall y\in[0,\frac{1}{2})$, $\lambda\in(0,1)$. This is to be done by contradiction. Suppose $\exists y\in[0,\frac{1}{2})$, $\lambda_0\in(0,1)$, such that $k(\lambda_0y_0)=\lambda_0k(y_0)$

Let $x_0=k(y_0)$ •or $\lambda_0x_0=k(\lambda_0y_0)$. Then

$$y_0=\frac{1}{\ln(1+\frac{1}{x_0})}-x_0$$
$$\lambda_0y_0=\frac{1}{\ln(1+\frac{1}{\lambda_0x_0})}-\lambda_0x_0$$

However, we also have

$$\lambda_0y_0=\lambda_0(\frac{1}{\ln(1+\frac{1}{x_0})}-x_0)$$

Thus, $(1+\frac{1}{x_0})^{\frac{1}{\lambda_0}}=1+\frac{1}{\lambda_0x_0}$

On the other hand, we have

$$(1+\frac{1}{x_0})^{\frac{1}{\lambda_0}}>1+\frac{1}{\lambda_0x_0},\forall x_0\in(0,\frac{1}{2}),\ \lambda_0\in(0,1)$$

This leads to a contradiction. **Q.E.D.**

Theorem 1 $\hat{M}_{n+1}\sim\hat{M}_n$ iff $(n+1)k(P_{n+1})\sim nk(P_n)$, where ~ can be >, <, or =.

Proof The proof follows after $P_n=f(\frac{\hat{M}_n}{n})$ and that $k(.)$ is strictly increasing function. **Q.E.D.**

Theorem 2 If $P_{n+1}>P_n$, then $\hat{M}_{n+1}>\hat{M}_n$.

Proof $P_{n+1}>P_n$ implies $k(P_{n+1})>k(P_n)$, which in turn implies $(n+1)k(P_{n+1})>nk(P_n)$. **Q.E.D.**

Theorem 3 If $(n+1)P_{n+1}<nP_n$, then $\hat{M}_{n+1}<\hat{M}_n$

Proof Note $k(.)$ is a strictly increasing and convex function, and $k(0)=0$. Hence:

$$k(P_{n+1})<k(\frac{n}{n+1}P_n)=k(\frac{n}{n+1}P_n+\frac{1}{n+1}0)$$

$$< \frac{n}{n+1} k(P_n) + \frac{1}{n+1} k(0) = \frac{n}{n+1} k(P_n)$$

This completes the proof as a result of Theorem 1. **Q.E.D.**

4. Example of software defect curves

In this example, we use the software reliability dataset of Musa [12]. $\hat{M}_n$ ($1 \le n \le 136$)can be computed from Equations (1)-(3). Figure 4 depicts the trajectory of $\hat{M}_n$. Disregarding the portion of $\hat{M}_n$ for $1 \le n \le 27$, which is over fluctuating and unreliable as anticipated at the very early stage of software testing, Figure 4 matches the pattern of Figure 2. For $28 \le n \le 95$, the behavior of $\hat{M}_n$ is displayed in Figure 5, which also matches the pattern of Figure 1.

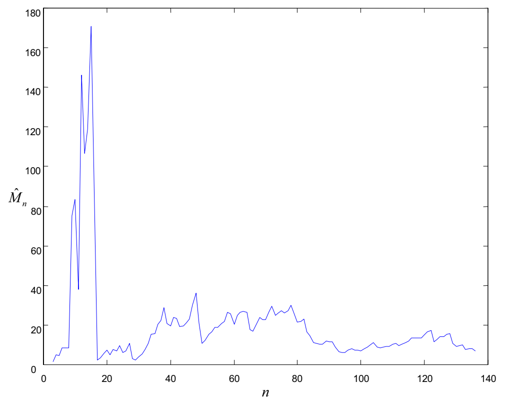

Figure 4 **Trajectory of** $\hat{M}_n$ **for the Musa Data**

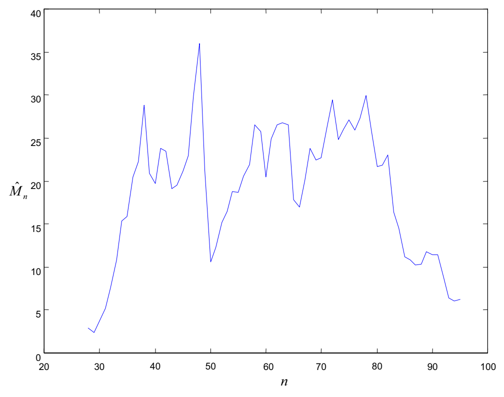

Figure 5 **Trajectory of** $\hat{M}_n$ **for the Musa Data from** $n = 28$ **to** $n = 95$

The example software defect curves in Figures 4 and 5 demonstrate:

(1) The trapezoidal shape of software defect curves complies with empirical observations.

(2) Although the expected trend of a software defect curve may demonstrate a single-trapezoidal-like curve or a multiple-trapezoidal-like curve, fluctuations are associated with the software defect curve throughout the testing process.

(3) For the trapezoidal-like software defect curve, if $\hat{M}_n$ tends to increase, then the software testing is in its early phase with respect to the corresponding test profile; if $\hat{M}_n$ tends to fluctuating around certain values, then the software testing is in its middle phase with respect to the corresponding test profile; if $\hat{M}_n$ tends to decrease, then the software testing is in its late stage with respect to the corresponding test profile.

Once we know whether $\hat{M}_n$ tends to increase or decrease, we can judge the phase of software testing. As shown in Equations (1)–(3), $\hat{M}_n$ can be calculated only after we know $\hat{b}_n$. However, Equation (2) is an implicit function of $\hat{b}_n$. Therefore, it is very difficult to calculate $\hat{M}_n$ on-line during the process of software testing. In order to solve the problem, we have developed Theorems 1-3 to provide other more easily computable conditions. These conditions involve P_n which could be computed directly from the data collected during the process of software testing. In view of Theorems 1-3, it is natural to ask: will P_n be as effective as $\hat{M}_n$ in judging the stage of software testing. Figures 6 and 7 show the relationship between P_n and *n*.

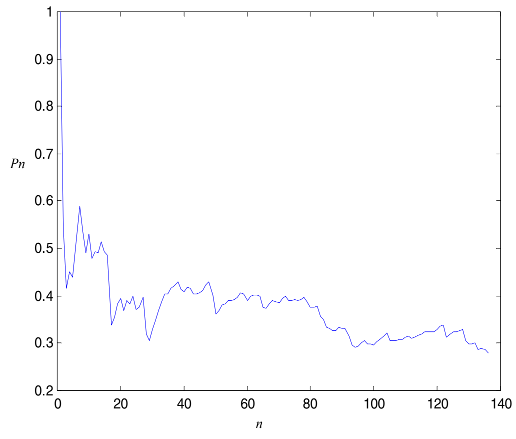

Figure 6 **Trajectory of** P_n **for the Musa Data**

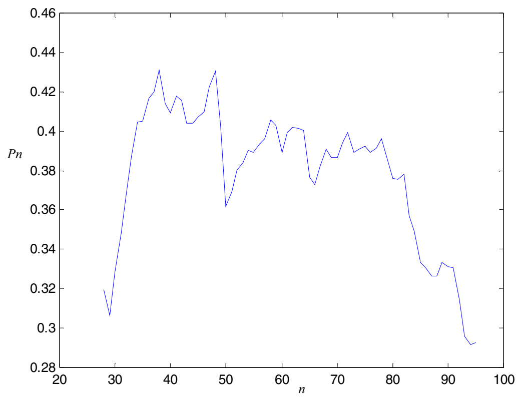

Figure 7 Trajectory of P_n for the Musa Data from $n = 28$ to $n = 95$

Having compared Figures 4 and 6, as well as Figures 5 and 7, we can conclude that the patterns of the trajectories of P_n and $\hat{M}_n$ are very similar. Since it is easier to compute P_n, P_n should be used instead of $\hat{M}_n$ to check the stage of software testing.

5. Conclusion

In this paper, we have reviewed the trapezoidal shape of software defect curves and presented some necessary and/or sufficient conditions for software defect curves of the Goel-Okumoto NHPP model. These results have supplemented the results of our previous theoretical analyses. Our necessary/sufficient conditions provide a faster approach to predict the types of change for the estimated number of remaining software defects, after detecting and removing a software defect. We have also used an example software defect curve generated by a field software reliability dataset to further justify the trapezoidal shape of software defect curves, and to demonstrate the applications of our necessary/sufficient conditions. In addition, these necessary/sufficient conditions may help to assess software testing progress and thus provide useful feedback information for adaptive software testing which is counterpart of adaptive control in software testing and falls into the scope of software cybernetics. Software cybernetics explores the interplay between software and cybernetics [13, 14].

References

[1] E.Yourdon, "Software Metrics," *Application Development Strategies* (*newsletter*), Nov. 1994, pp 16.

[2] K.Y. Cai, *Software Defect and Operational Profile Modeling*, Kluwer Academic Publishers, 1998.

[3] A.M. Neufelder, "How to predict software defect density during proposal phase", *National Aerospace and Electronics Conference*. NAECON 2000, pp 71 -76.

[4] N.E. Fenton and M. Neil, "A critique of software defect prediction models" *IEEE Transactions on Software Engineering* , Vol: 25 ,Issue: 5 , pp675 -689.

[5] D. N. Card, "Managing Software Quality with Defects",In *Proceedings: Computer Software and Applications Conference*, August 2002 , 2002 IEEE Computer Society.

[6] C.Y. Huang, S.Y. Kuo, and I.Y. Chen, "Analysis Of A Software Reliability Growth Model With Logistic Testing-Effort Function", *8th International Symposium on Software Reliability Engineering*, Nov, 1997, pp378-388.

[7] S. McConnell, "Software Quality at Top Speed", *Software Development*, Aug,1996.

[8] C.G. Bai, K.Y. Cai, "Software Defect Curves", submitted for publication, 2002.

[9] A.L. Goel and K. Okumoto, "Time Dependent Error Detection Rate Model for Software Reliability and Other Performance Measure", *IEEE Transactions on Reliability*, Vol. R-28, No.3, 1979, pp206-211.

[10] K.Y. Cai, "Towards a Conceptual Framework of Software Run Reliability Modeling", *Information Sciences*, Vol.126, 2000, pp137-163.

[11] D.L. Syder, *Random Point Processes*, Wiley, New York, 1975.

[12] J. D. Musa, *Software Reliability Data*, Bell Telephone Laboratories Whippany, N.J. 07981, 1979.

[13] K.Y.Cai, "Optimal Software Testing and Adaptive Software Testing in the Context of Software Cybernetics", *Information and Software Technology*, Vol.44, 2002, pp841-855.

[14] K.Y.Cai, T.Y.Chen, T.H.Tse, "Towards Research on Software Cybernetics", *Proc. 7th IEEE International Symposium on High Assurance Systems Engineering*, 2002, pp240-241.