http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (6): 1227-1232, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.1227.1232

Analytical Modeling of Periodically Inspected Software Rejuvenation Policy

[1]Meng Haining, [1]Hei Xinhong and [2]Liu Jianjun
[1]School of Computer Science and Engineering, Xi'an Technology University, Xi'an 710048, China
[2]Aeronautics Computing Technique Research Institute, Xi'an 710068, China

Abstract: As a proactive and preventive software fault tolerant technique, software rejuvenation is a main method for counteracting software aging. In this study, a software rejuvenation model based on periodical inspection policy is set up. Firstly, by analyzing runtime state and failure feature of software system, the functions of system unavailability and cost rate are given and optimal system inspection interval and software rejuvenation interval are selected via minimizing system unavailability and cost rate. Then boundary conditions of cost rate and optimal inspection interval are deduced. Finally, quantitative analysis and numeric experiment result show that selecting optimal inspection interval and software rejuvenation interval can greatly reduce system cost and improve system availability and reliability. In addition, the numeric experiment result further validate that the software rejuvenation model with periodical inspection policy has higher system availability than the general software rejuvenation model in the case of the failure rate following Weibull and exponential distribution.

Key words: Software rejuvenation, software aging, periodical inspection, cost rate, unavailability

INTRODUCTION

Researches on software reliability indicate the existence of software aging phenomenon in long-running software system, in which the system performance degrades gradually with time and even sudden downtime appears (Avritzer and Weyuker, 1997). Software aging has been observed in software system such as operating system, web server, Java virtual machine and so on (Vaidyanathan and Trivedi, 1999; Grottke *et al*., 2006; Cotroneo *et al*., 2007). The causes of software aging can be mainly divided into two categories. One is system instantaneous collapse caused by deadly system abnormalities and the other is system performance degradation and memory capacity decline caused by resources exhaustion.

In order to attack software aging and avoid the serious losses caused by sudden system failures, a proactive and preventive fault-tolerant technique called software rejuvenation is introduced by Huang *et al*. (1995). It involves taking measures to periodically roll back the running software in order to avoid the occurrence of greater failures in the future.

The current study on software rejuvenation technology mainly has two kinds of policies: the model-based policy and the measurement-based policy. In the model-based policy, the software rejuvenation model is set up based on the assumption of some distributions for system failures to determine the optimal rejuvenation interval. Analytical modeling has been used to address this issue in several research studies. Huang *et al*. (1995) introduced the continuous Markov process to build two-phase software rejuvenation model by taking into consideration the downtime costs. Garg *et al*. (1998) presented a software rejuvenation model of a transaction processing system and obtained the optimal rejuvenation interval by minimizing the probability of transaction loss and maximizing the system availability. Xie *et al*. (2005) took the semi-Markov process into account for building a model to estimate software rejuvenation schedule by maximizing the system availability. Avritzer *et al*. (2010) introduced a software rejuvenation model in mission critical systems that are subjected to worm infection via tracking both the state of the mission and the customer affecting metric. Dohi *et al*. (2012) considered a two step failure process model with periodic rejuvenation and derived the optimal rejuvenation policy maximizing the system reliability. In general, the model-based software rejuvenation policy has advantage in the scenario of less change of runtime state yet it has poor flexibility and is hard to make decision in time.

In the measurement-based policy (Grottke *et al*., 2006; Cotroneo *et al*., 2007; Matias *et al*., 2010; Kourai and Chiba, 2011), the runtime state and performance parameters of software system are monitored and detected continuously and the software rejuvenation interval is

Corresponding Author: Meng Haining, School of Computer Science and Engineering, Xi'an Technology University, Xi'an 710048, China

estimated based on the collection and statistical analysis of system data. This kind of policy has high flexibility in real-time decision-making and is suitable for large change of operation scenario. However, it needs frequently monitor the system with more system resources exhaustion and the computation is relatively complex and expensive.

For the reason that system failures occurred randomly, it probably takes a period of time for system failures to be discovered. In this case, the periodical inspection mechanism is introduced to discover system failures. However, the frequent inspection of running software system incurs some overhead and brings greater system losses, it is important to determine when and how often software rejuvenation and system inspection should be initiated.

In this study, from the mathematic modeling point of view, a software rejuvenation model is built with periodical inspection policy and probability statistics method. In order to improve software reliability, the function of system unavailability and cost rate is given and optimal system inspection interval and software rejuvenation interval are derived. Finally, the numerical results are shown to validate the proposed model.

MODELLING AND ANALYSIS OF PERIODICALLY INSPECTED SOFTWARE REJUVENATION POLICY

Most previous studies which have focused on model-based software rejuvenation policy put forward different algorithms or models to derive the optimal rejuvenation interval. As shown in Fig. 1, the system begins operation with the best performance value of P_{max}. When the system operates at the time δ', system performance reduces to the value of P_{min} and thereby software rejuvenation is performed to bring the system back to the initial health state. However, due to the random occurrence of system failures, system performance could have declined to the minimum at time δ'' ($\delta''<\delta'$) and eventually the system crashed during software rejuvenation interval δ'. In this case the reactive maintenance method such as system recovery is taken action to recover the system to its initial state.

Basic idea of periodically inspected software rejuvenation policy: For the reason that system failures occurred randomly, it is hard to discover them immediately. Thus it is necessary to adopt the periodically inspected software rejuvenation policy, in which the system inspection is performed at intervals δ and the error log is used to record the information of whether the system failures occur or not.

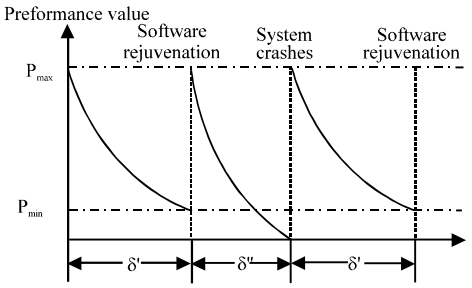

Fig. 1: Periodical rejuvenation model of software system, δ' is the software rejuvenation interval, δ'' is the time when system crashes, P_{max} is the best performance value, P_{min} is the worst performance value

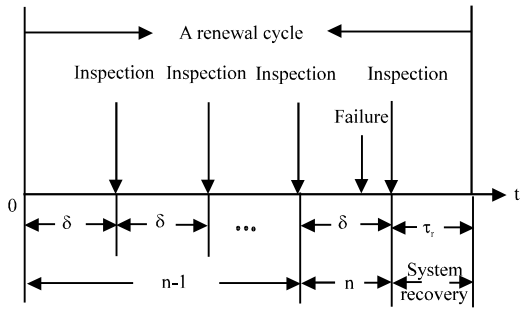

Fig. 2: Failures are detected after the inspection for k($k\le n$) times, δ is the inspection interval, τ_r is the execution time to recover from system failures, n is the system inspection times

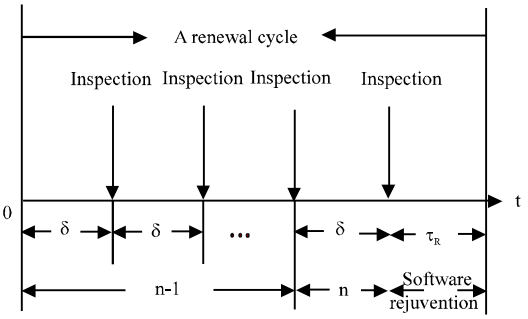

Fig. 3: No failure is detected after the inspection for n times, δ is the inspection interval, τ_R is the execution time to perform software rejuvenation, n is the system inspection times

The system completes a renewal cycle in two cases. In one case as shown in Fig. 2, the system failures are discovered during k ($k\le n$) inspection intervals and then the system recovery is carried out to bring the system to initial state. In another case as shown in Fig. 3, no failure

is detected after the n inspection intervals and then the software rejuvenation is performed to bring the system to initial state.

The basic assumptions on the periodically inspected software rejuvenation policy are:

- The value of τ_R and τ_r have been selected on whatever practical basis and $\tau_R<\tau_r$
- After the system recovery or software rejuvenation, software system will return to the new initial state
- The probability function of failure rate is increasing monotonically and has general distribution
- The inspections discover system failures and have no effect on the operation of software system

Periodically inspected software rejuvenation modeling and performance analysis: Based on the periodically inspected software rejuvenation policy above, the average renewal cycle begins with a renewal and ends either with software rejuvenation after time $n\delta$ (with probability $F(n\delta)$) or with system recovery (with probability $F(n\delta)$). Therefore, the average length of a renewal cycle $L(\delta,n)$ is expressed as follows:

$$\begin{aligned} L(\delta,n) &= (n\delta+\tau_R)\bar{F}(n\delta)+\sum_{k\delta-\delta}^{k\delta}(k\delta+\tau_r)dF(t) \\ &= \delta\sum_{k=0}^{n-1}\bar{F}(k\delta)+\tau_R\bar{F}(n\delta)+\tau_rF(n\delta) \end{aligned} \tag{1}$$

where, $F(t)$ is the distribution function of failure rate, $\bar{F}(t)$ is the survive function and $\bar{F}(t) = 1-F(t)$.

The expected uptime for software system in a cycle is:

$$R(\delta,n)=\int_0^{n\delta}tdF(t)+n\delta\cdot\bar{F}(n\delta)=\int_0^{n\delta}\bar{F}(t)dt \tag{2}$$

Software system is down when software recovery or system rejuvenation is started. Thus, the expected downtime per cycle is deduced as:

$$\begin{aligned} &D(\delta,n) \\ &= L(\delta,n)-R(\delta,n) \\ &= \delta\sum_{k=0}^{n-1}\bar{F}(k\delta)+\tau_R\bar{F}(n\delta)+\tau_rF(n\delta)-\int_0^{n\delta}\bar{F}(t)dt \end{aligned} \tag{3}$$

In addition, the expected number of system inspection intervals in a cycle is:

$$I(\delta,n)=\sum_{k=1}^{n}\int_{k\delta-\delta}^{k\delta}kdF(t)+n\bar{F}(n\delta)=\sum_{k=0}^{n-1}\bar{F}(k\delta) \tag{4}$$

It is assumed that c_I is the expected inspection cost and c_f is the expected cost during system downtime which consists of the time of system recovery, software rejuvenation and system failure. Then the average total cost per cycle is:

$$\begin{aligned} &c(\delta,n)=c_I\cdot I(\delta,n)+c_f\cdot D(\delta,n) \\ &=c_I\sum_{k=0}^{n-1}\bar{F}(k\delta)+c_f(\delta\sum_{k=0}^{n-1}\bar{F}(k\delta)+\tau_R\bar{F}(n\delta)+\tau_rF(n\delta)-\int_0^{n\delta}\bar{F}(t)dt) \end{aligned} \tag{5}$$

The cost rate is defined as the ratio of the average total cost per cycle to the average length of a renewal cycle. Then the cost rate function $r(\delta,n)$ is:

$$\begin{aligned} r(\delta,n) &= \frac{c(\delta,n)}{L(\delta,n)} \\ &= \frac{c_I\cdot I(\delta,n)+c_f\cdot D(\delta,n)}{\delta\sum_{k=0}^{n-1}\bar{F}(k\delta)+\tau_R\bar{F}(n\delta)+\tau_rF(n\delta)} \\ &= c_f-\frac{c_f\int_0^{n\delta}\bar{F}(t)dt-c_I\sum_{k=0}^{n-1}\bar{F}(k\delta)}{\delta\sum_{k=0}^{n-1}\bar{F}(k\delta)+\tau_R\bar{F}(n\delta)+\tau_rF(n\delta)} \end{aligned} \tag{6}$$

The system availability is defined as the ratio of the expected uptime per cycle to the average length of a renewal cycle. Then the system availability function is:

$$\begin{aligned} A(\delta,n) &= \frac{R(\delta,n)}{L(\delta,n)} \\ &= \frac{\int_0^{n\delta}\bar{F}(t)dt}{\delta\sum_{k=0}^{n-1}\bar{F}(k\delta)+\tau_R\bar{F}(n\delta)+\tau_rF(n\delta)} \end{aligned} \tag{7}$$

According to formula (7), the system unavailability function is derived as:

$$\begin{aligned} U(\delta,n) &= 1-A(\delta,n)=1-\frac{R(\delta,n)}{L(\delta,n)} \\ &= 1-\frac{\int_0^{n\delta}\bar{F}(t)dt}{\delta\sum_{k=0}^{n-1}\bar{F}(k\delta)+\tau_R\bar{F}(n\delta)+\tau_rF(n\delta)} \end{aligned} \tag{8}$$

A practical task is to minimize the cost rate by proper selection of δ and n. A reasonable approach is to calculate the derivative of the cost rate function $r(\delta,n)$ or the system unavailability function $U(\delta,n)$ and if it satisfies $r'(\delta,n) = 0$ or $U'(\delta,n) = 0$, then the optimal δ is obtained.

Boundary conditions: By analyzing the asymptotic behavior of cost rate function $r(\delta,n)$ for increasing δ and any fixed $n = 1$, two boundary conditions are concluded as follows:

- **Theorem 1:** The cost rate of software system is not more than the expected cost during system downtime, i.e. $r(\delta,n) = c_f$

Proof: From formula (6):

$$r(\delta,n) = c_f - \frac{c_f\int_0^{n\delta}\bar{F}(t)dt - c_I\sum_{k=0}^{n-1}\bar{F}(k\delta)}{\delta\sum_{k=0}^{n-1}\bar{F}(k\delta) + \tau_R\bar{F}(n\delta) + \tau_r F(n\delta)}$$

Consider the mean time to system failure which is given by:

$$t_f = \int_0^{\infty}\bar{F}(t)dt$$

It follows that:

$$\int_0^{n\delta}\bar{F}(t)dt \xrightarrow[\text{large }\delta]{} t_f$$

Consider that $\lim_{t\to\infty}F(t) = 1$ and $\lim_{t\to\infty}\bar{F}(t) = 0$.
Thus:

$$\sum_{k=0}^{n-1}\bar{F}(k\delta) \xrightarrow[\text{large }\delta]{} \bar{F}(0) = 1$$

Formula (6) yields:

$$r(\delta,n) \xrightarrow[\text{large }\delta]{} c_f - \frac{c_f t_f - c_I}{\delta + \tau_r}$$

Furthermore, note that:

$$\lim_{\delta\to\infty}\left(c_f - \frac{c_f t_f - c_I}{\delta + \tau_r}\right) = c_f$$

Hence:

$$\lim_{\delta\to\infty} r(\delta,n) = c_f$$

Therefore, $r(\delta,n) \le c_f$, the assertion is proven.

- **Theorem 2:** The optimal system inspection interval is not less than the ratio of the expected inspection cost to the expected cost during system downtime, i.e.:

$$\delta^* \ge \frac{c_I}{c_f}$$

Proof: According to Theorem 1:

$$r(\delta^*,n) \le c_f$$

Formula (6) yields:

$$c_f\int_0^{n\delta^*}\bar{F}(t)dt \ge c_I\sum_{k=0}^{n-1}\bar{F}(k\delta^*) \tag{9}$$

For the survive function $\bar{F}(t)$ is non-increasing such that:

$$\delta\sum_{k=0}^{n-1}\bar{F}(k\delta) \ge \int_0^{n\delta}\bar{F}(t)dt \quad (\text{For any } \delta \ge 1) \tag{10}$$

Hence, it follows that:

$$\delta\cdot c_f\cdot\sum_{k=0}^{n-1}\bar{F}(k\delta) \ge c_f\cdot\int_0^{n\delta}\bar{F}(t)dt \,(\text{For any } \delta \ge 1) \tag{11}$$

Based on the triangle inequality, Eq. (9-11) yield:

$$\delta^*\cdot c_f\cdot\sum_{k=0}^{n-1}\bar{F}(k\delta^*) \ge c_I\cdot\sum_{k=0}^{n-1}\bar{F}(k\delta^*)$$

That is:

$$\delta^* \ge \frac{c_I}{c_f}$$

Therefore, the assertion is proven.

NUMERICAL RESULTS AND ANALYSIS

Based on the periodically inspected software rejuvenation model above, numerical experiments are performed by taking system unavailability and cost rate as evaluation indicators of system reliability. The model is solved for multiple values of δ and n and the optimal value of δ is determined. Meanwhile, the affects of software rejuvenation and periodical inspection on system reliability are investigated. By minimizing system unavailability and cost rate, the optimal software rejuvenation interval and system inspection interval are obtained.

Numerical values of system parameters are given in Table 1. All the system parameter values are selected by experimental experience for demonstration purposes.

Figure 4 shows the relationship between system inspection interval and cost rate for different values n and in the case of failure rate following the Weibull distribution F(t), where:

Table 1: System parameter

Parameter	c_f	c_I	τ_R (h)	τ_r(h)
Value	50	2	0.2	0.3

c_f: Expected cost during system downtime, c_I: Expected inspection cost, τ_R: Execution time to perform software rejuvenation, τ_r: Execution time to recover from system failures

$$F(t) = 1 - e^{-(ct+(at)^b)}$$

Here, $c = 3\times10^{-3}$, $\alpha = 2.8\times10^{-3}$ and $b = 4$.

It can be observed from Fig. 4 that for the fixed inspection times n, when the system inspection interval δ is very small which means the frequency of triggering software rejuvenation is very high, the system is almost unavailable and the cost rate is considerable. With the increase of the system inspection interval δ, the cost rate reduces rapidly and goes to the minimum at the point of optimal inspection interval δ^*. And then the inspection interval δ continues to increase, the possibility of occurring failures are steadily rising and the cost rate becomes bigger. In addition, on the whole, the higher the inspection times n is, the lower the cost rate and the optimal inspection interval δ^* are.

The cost rate and system unavailability versus optimal inspection interval with varying inspection times n is shown in Table 2, from which the optimum inspection interval δ^* is selected when the value of the cost rate function $r(\delta,n)$ and the system unavailability function $U(\delta,n)$ reach the minimum. It can be seen From Table 2 that the overall optimal combination which minimize the cost rate and unavailability is $n = 9$ and $\delta^* = 5$ h (i.e. the optimal software rejuvenation interval is 45 h). Correspondingly, the system unavailability $U(\delta^*,n)$ is 0.1641 and the cost rate $r(\delta^*,n)$ is 1.0872. It also can be seen that the reduction of inspection times leads to larger inspection interval and software rejuvenation interval and increasing the cost rate and the system unavailability.

Finally, the differences between our periodically inspected software rejuvenation model and the general software rejuvenation model without periodical inspection policy given by Huang *et al.* (1995) are investigated. Aiming at the latter model, Fig. 5 illustrates the relationship between software rejuvenation interval and system unavailability in cases of failure rate following with the Weibull and exponential distribution. It can be observed that system unavailability decreases with software rejuvenation interval δ' increases, attains a minimum at $\delta' = 15$ h and $\delta' = 21$ h, respectively and then gradually increases. Corresponding, the minimum value of system unavailability is 0.02604 and 0.02025, respectively, which is greater than 0.01641 derived from the periodically inspected software rejuvenation model above. It indicates that our periodically inspected software rejuvenation model has superiority in the aspect of improving system availability compared with the general software rejuvenation model.

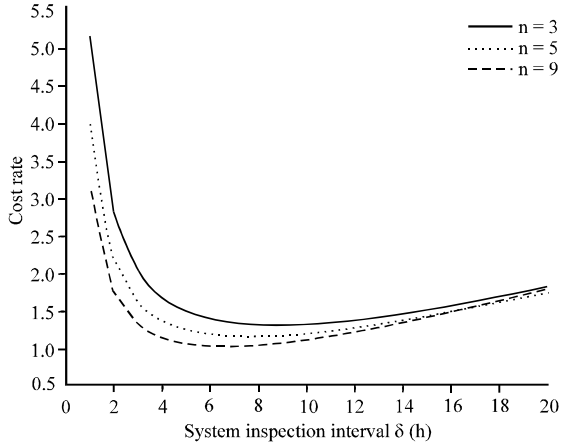

Fig. 4: Cost rate versus system inspection interval, n is the system inspection times

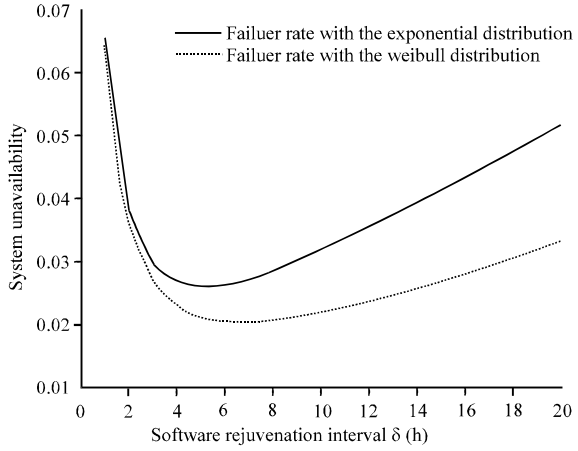

Fig. 5: System unavailability versus software rejuvenation interval

Table 2: Comparison between optimal inspection interval and cost rate/unavailability

	System inspection times n									
	1	2	3	4	5	6	7	8	9	10
δ^*	13	9	8	7	7	6	6	5	5	5
$r(\delta^*,n)$	1.8728	1.458	1.2895	1.2022	1.1548	1.1198	1.106	1.0935	1.0872	1.0911
$U(\delta^*,n)$	0.04072	0.02926	0.02488	0.02195	0.021	0.018877	0.18592	0.016541	0.016406	0.016473

δ^* is the optimal inspection interval, $r(\delta^*,n)$ is the cost rate, $U(\delta^*,n)$ is the system unavailability

CONCLUSION AND IMPLICATIONS

Software aging is an important potential factor that affects the software reliability. As a proactive and preventive software fault tolerant technique, software rejuvenation is a main method for counteracting software aging. Considered system failures occurred randomly and analyzing the runtime state of software system, a software rejuvenation model is presented by using periodical inspection policy. The optimal inspection interval and optimal software rejuvenation interval are selected via minimizing system unavailability and cost rate. Then boundary condition of cost rate and system inspection interval is deduced. Finally, quantitative analysis and numeric experiment results show that selecting optimal system inspection interval and scheduling optimal software rejuvenation can greatly reduce the average cost and improve the system reliability.

Analytic work in the future may focus on the fine-granularity software rejuvenation model considering of system inspections on part of software system. Dynamically selecting the software rejuvenation policy is another avenue for new development.

ACKNOWLEDGMENTS

The author would like to thank the sponsors of the National Natural Science Foundation of China under Grant No. 61100173, Scientific Research Plan Project of Shaanxi Education Department of China under Grant No. 09JK642, Doctoral Fund No. 116-210912 and Scientific Research Plan Project of Xi'an Technology University under Grant No. 116-210907.

REFERENCES

Avritzer, A. and E.J. Weyuker, 1997. Monitoring smoothly degrading systems for increased dependability. Empirical Software Eng., 2: 59-77.

Avritzer, A., R.G. Cole and E.J. Weyuker, 2010. Methods and opportunities for rejuvenation in aging distributed software systems. J. Syst. Software, 83: 1568-1578.

Cotroneo, D., S. Orlando and S. Russo, 2007. Characterizing aging phenomena of the Java virtual machine. Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems, October 10-12, 2007, Beijing, China, pp: 127-136.

Dohi, T., H. Okamura and K.S. Trivedi, 2012. Optimizing software rejuvenation policies under interval reliability criteria. Proceedings of the 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing, September 4-7, 2012, Fukuoka, Japan, pp: 478-485.

Garg, S., A. Puliafito, M. Telek and K. Trivedi, 1998. Analysis of preventive maintenance in transactions based software systems. IEEE Trans. Comput., 47: 96-107.

Grottke, M., L. Li, K. Vaidyanathan and K.S. Trivedi, 2006. Analysis of software aging in a web server. IEEE Trans. Reliab., 55: 411-420.

Huang, Y., C. Kintala, N. Kolettis and N. Fulton, 1995. Software rejuvenation: Analysis, module and applications. Proceedings of the 25th International Symposium on Fault Tolerant Computing, Jun 27-30, 1995, Pasadena, CA., USA., pp: 381-390.

Kourai, K. and S. Chiba, 2011. Fast software rejuvenation of virtual machine monitors. IEEE Trans. Dependable Secure Comput., 8: 839-851.

Matias, R., P.A. Barbetta, K.S. Trivedi and P.J.F. Filho, 2010. Accelerated degradation tests applied to software aging experiments. IEEE Trans. Reliab., 59: 102-114.

Vaidyanathan, K. and K.S. Trivedi, 1999. A measurement-based model for estimation of software aging in operational software systems. Proceedings of the 10th International Symposium on Software Reliability Engineering, November 1-4,1999, Boca Raton, Florida, USA., pp: 84-93.

Xie, W., Y. Hong and K. Trivedi, 2005. Analysis of a two-level software rejuvenation policy. Reliab. Eng. Syst. Saf., 87: 13-22.