Journal of the Korean Nuclear Society
Volume 29, Number 6, pp433~443, December 1997

A Dependability Modeling of Software Under Memory Faults for Digital System in Nuclear Power Plants

Jong Gyun Choi and Poong Hyun Seong
Korea Advanced Institute and Science Technology
373-1 Kusong-dong, Yusong-gu, Taejon 305-701, Korea

(Received March 16, 1996)

Abstract

In this work, an analytic approach to the dependability of software in the operational phase is suggested with special attention to the hardware fault effects on the software behavior : The hardware faults considered are memory faults and the dependability measure in question is the reliability. The model is based on the simple reliability theory and the graph theory which represents the software with graph composed of nodes and arcs. Through proper transformation, the graph can be reduced to a simple two-node graph and the software reliability is derived from this graph.

Using this model, we predict the reliability of an application software in the digital system (ILS) in the nuclear power plant and show the sensitivity of the software reliability to the major physical parameters which affect the software failure in the normal operation phase. We also found that the effects of the hardware faults on the software failure should be considered for predicting the software dependability accurately in operation phase, especially for the software which is executed frequently. This modeling method is particularly attractive for the medium size programs such as the microprocessor-based nuclear safety logic program.

1. Introduction

Computers today form integral parts of large systems where processing and control are the primary demands. They can also be the mainstay of systems, such as airline reservation systems and nuclear power plants which are required to be highly reliable. Designing the reliable software and accurately predicting the software reliability are, therefore, the most important issues that the computer designers and developers face. During the last two decades, the assessment of the software failure phenomena and the prediction of software dependability have become key issues in digital system design and have drawn the attention of many researchers because the production and maintenance cost of the software has been rapidly increasing. Most of these researches, however, have focused on the development and test phases of software rather than on the operational phase : Models[1-3] are mainly aimed at predicting the future reliability from the failure data accumulated in the past. Some software reliability models consider program in the operational phase[4-7], but these models are less than ideal[8-10] because they analyze the system software as

an entity separated from the system hardware, making it difficult to analyze the interaction of the two in the integrated system.

In recent years, various experiments and studies[11-13] have shown that the hardware faults can affect the software dependability and vice versa. These interactions may be very important factors to predict the dependability of systems that are used in life-critical applications such as flight control system as well as those used in safety critical applications such as nuclear power plant monitoring and control systems. Previous models for dependability prediction of software, in short, have not only mainly concentrated on the development phase focusing on the reliability growth of single component systems but also can not explain these effects of interactions between the software and hardware.

An analytic approach to the dependability evaluation of software in the operational phase is suggested in this work with special attention to the physical fault effects on the software dependability. The physical faults considered are memory faults and the dependability measure in question is the reliability. The model is based on the simple reliability theory and the graph theory with the path decomposition micro-model. The model represents an application software with a graph consisting of nodes and arcs that probabilistically determine the flow from node to node. Through proper transformation of nodes and arcs, the graph can be reduced to a simple two-node graph and the software failure probability is derived from this graph.

Using these models, we predict the reliability of an application software in a digital system, Interposing Logic System (ILS), in the nuclear power plant and show the sensitivity of the software reliability to major physical parameters which affect the software failure in the normal operation phase. It is found that the effects of the hardware faults on the software failure should be properly considered for predicting the software dependability accurately in operation phase.

2. Models

An application of the microprocessors in the nuclear power plant is for the nuclear safety logic. The microprocessors are applied to the nuclear safety logic in Younggwang(YGN) nuclear power plant units 3 and 4 in Korea. The microprocessors which are of high reliability and of wide experience in the market are used. The software

A. Memory Fault and Software Fault

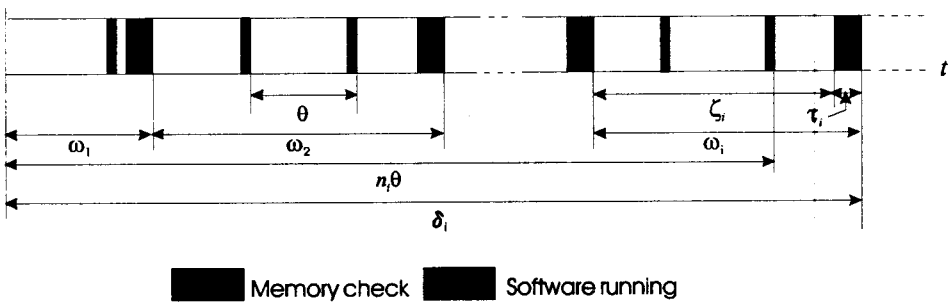

Fig. 1. Fault Recovery Mechanism Between Memory Checking Period.

stored in erasable programmable read only memory(EPROM) is executed when it is required to actuate some specific components in nuclear power plants. The memory has no error correcting circuitry but is supported by the memory test routine to detect memory error periodically. The time between any two successive memory faults is assumed to be exponentially distributed with parameter λ. Fault location in memory is assumed to be random. It is further assumed that, when the software code stored in faulted memory location is executed, the software failure occurs.

In Section A, the memory faults which cause the software failure are described and modeled. In Sections B and C, the methodology for prediction of the software reliability is introduced. The sensitivity of the software reliability to the major physical parameters has been estimated in Section D.

2.1. Memory Fault and Software Fault

As shown in the Figure 1, the memory test routine checks the memory faults periodically. The software on the ith execution state can fail by memory faults which occur between the previous memory checking time before the software enters ith execution state and the time at which software enters $i+1$th idle state. The probability of the memory faults occurrence which affects the ith software execution can be obtained as follows :

For $n_i\theta \le \delta_i < (n_i+1)\theta$,

$$l_i = \frac{S}{M}\cdot\int_0^{\delta_i-n_i\theta}\lambda e^{-\lambda t}dt = -\frac{S}{M}\cdot e^{-\lambda t}\Big|_0^{\delta_i-n_i\theta} = \frac{S}{M}\cdot(1-e^{-\lambda(\delta_i-n_i\theta)}), \quad (1)$$

where n_i is the largest integer given that $n_i\theta$(is not greater than δ_i. Then, the probability that the fault occurs in one machine instruction can be obtained as the following :

$$q_h(M,\lambda,\delta_i,\theta) = 1-p_h = l_i\frac{1}{S} = \frac{1}{M}(1-e^{-\lambda(\delta_i-n_i\theta)}). \quad (2)$$

If $\omega_1 \approx \omega_2 \approx \cdots \approx \omega_i \approx \cdots \approx \omega$,

$$p_h = 1-\frac{1}{M}(1-e^{-\lambda(i\omega-n_i\theta)}). \quad (3)$$

As shown in the Equation (3), the value of p_h depends on the parameters M, μ, ω, and θ. The software failure by memory faults depends on the above four parameters. The parameter μ is obtained from Mil-Hdbk-217D. This standard handbook presents various data for micro-electronic devices.

p_s can be evaluated from the method in [14-15] by testing programs of the same characteristics and the size developed by the same programming team. Then we can get an upper s-confidence interval on p_s.

2.2. Software Failure Probability

The probabilistic program control flow graphs are useful for the estimation of the execution time of the software and they also can be used for the analysis of the software failure[16]. In the program graph construction, we can obtain a graph by using an arc for one instruction and labeling the arc with the execution characteristic value of that instruction, e.g., instruction execution time. Nodes are used for the branch and the joint points for the control flow in the software.

Let p(*i, j*) be the probability of entering the path from node *i* to node *j*. The joint probability of entering and executing successfully the path (*i, j*) is given by

$$p_{i,j} = p(i \to j \mid i,j)\cdot p(i,j), \quad (4)$$

where $p(i \to j \mid i, j)$ is the probability of successfully executing the path (*i, j*) given that the

program has entered the path (*i j*) and is given by

$$p(i \rightarrow j | i,j) = p_h^n \cdot p_s^m, \quad (5)$$

where n is the number of machine instruction in path (*i, j*) and m the number of software instructions in path (*i, j*). That is, in order for the software to be executed, the software instructions need to be transformed into the machine instructions which are stored in memory. When one software instruction is transformed to machine instruction(s), it can be transformed to several machine instructions and consequently occupy several memory locations. Therefore, in order for one instruction to be executed successfully, the memory locations of transformed machine instructions must be free of physical faults and simultaneously, there must not be software faults. p_h in every machine instruction is considered to be equal because it is assumed that the location of the memory faults occurrence is random. We know neither which software instruction has faults caused by software design or coding error nor which fault in instructions causes the failure of software. It is therefore assumed that every software instruction has the same failure probability by software faults. Then p_s in every instruction is set to be equal.

p_h depends on several parameters including time as described in section A, whereas p_s depends on software quality. For program loops with L known, the looping probability is $L/(L+1)$ and the probability of not-looping is $1/(L+1)$. Probability $p_{i,j}$ and time $\tau_{i,j}$ are assigned to each oriented arc of the graph, where node *i* is the origin of a directed arc and node *j* the destination. Following the software reliability prediction method in [14], three elementary transformations are defined - series, parallel and loop as shown in Figure 2. In these transformations, it is assumed that entering and executing successfully different paths are mutually s-independent events. The series transformation applies to a pair of arcs in series where the terminal node of one arc is the origin node of the other arc. The pair of arcs and the node between them can be replaced by a single arc, provided no other arcs terminate or originate at the interior nodes.

In the series reduction, arcs (*i, k*) and (*k, j*) are replaced by a new arc (*i, j*) as in the Figure 2. The probability and execution time for this new arc are as follows :

$$p_{i,j} = p_{i,k} \cdot p_{k,j}, \quad \tau_{i,j} = \tau_{i,k} + \tau_{k,j}. \quad (6)$$

The parallel transformation is applied to a pair of arcs in parallel, that is, a pair of arcs which have the same origin node and the same terminal node. The pair of arcs is replaced by a single new arc. The probability and execution time for this new arc are as follows:

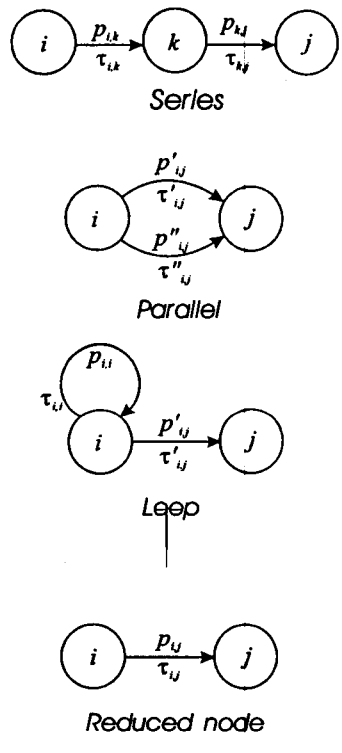

Fig 2. Node Transformation

$$p_{i,j} = p'_{i,j} + p''_{i,j}, \quad \tau_{i,j} = \frac{p'_{i,j}\cdot\tau'_{i,j} + p''_{i,j}\cdot\tau''_{i,j}}{p'_{i,j} + p''_{i,j}}. \quad (7)$$

The loop transformation removes an arc which has the same node for both its origin and terminal nodes. The new probability and execution time assigned to the remaining arc are as follows :

$$p_{i,j} = \frac{p'_{i,j}}{1 - p_{i,i}}, \quad \tau_{i,j} = \tau'_{i,j} + \frac{p_{i,j}\cdot\tau_{i,i}}{1 - p_{i,i}}. \quad (8)$$

With the above transformation methods we can get a simple two-node graph as shown in Figure 3. The nodes are numbered from 1 to N. Then we have the software failure probability as the following :

$$f(p_h, p_s) = 1 - p_{1,N}, \quad (9)$$

where $f(p_h, p_s)$ is the failure probability of the software.

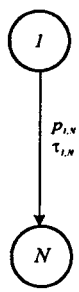

Fig. 3. 2-Node Graph Model

2.3. Cumulative Software Failure robability

As shown in Figure 4, the software is either in idle state or in execution state. The software is idle during the time ζ_i between the i-1th and ith execution state. The ith execution of the software begins after this idle time ζ_i and continues for the execution time τ_i. Because τ_i is very small compared with the idle time ζ_i, it is reasonable to assume that $\tau \approx \tau_1 \approx \cdots \approx \tau_i \approx \cdots \approx 0$. Since the first software execution begins after idle time ζ_1, the failure probability of software before time δ_1 is zero as shown in the following :

$$P\{Failure\ Time\} \equiv F(t) = 0, \text{ for } 0 \le t < \delta_1 \quad (10)$$

For $\delta_1 \le t < \delta_2$, the software is executed once before time t. The failure probability is, therefore, given by

$$F(t) = f_1 \quad (11)$$

Similarly, the following equations are obtained :

$$F(t) = f_1 + (1-f_1)f_2, \text{ for } \delta_2 \le t < \delta_3 \quad (12)$$

$$F(t) = f_1 + (1-f_1)f_2 + (1-f_1)(1-f_2)f_3, \text{ for } \le t < \delta_4 \quad (13)$$

Generally, for $\delta_i \le t < \delta_{i+1}$ and $i \ge 2$, we have Cumulative Software Failure Probability as the

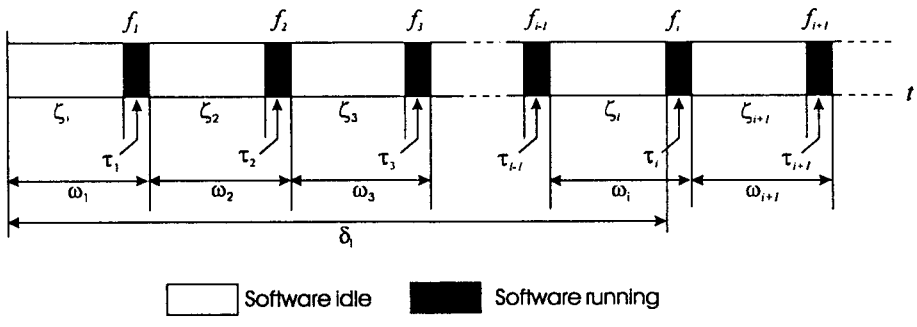

Fig. 4. Relationship Between the Various Time Interval

Table 1. Information of an Application Software

Hardware		Application	
CPU	8085	SIZE	72 Byte
MEMORY	64k EPROM	EXECUTION TIME	379 Clock time
CLOCK FREQ.	1 Mhz	CHECKING PERIOD	5 Minutes

Table 2. Mean Time to Failure of the Software

Chip Quality	α	
	0.5 hr	500 hr
Normal (10^{-1}/1000 hr)	1.57×10^{3} yr	1.57×10^{6} yr
High (10^{-5}/1000 hr)	1.57×10^{7} yr	1.57×10^{10} yr

following :

$$F(t) = f_1 + \sum_{k=1}^{i-1} f_{k+1} \cdot \prod_{j=1}^{j=k} (1 - f_j). \tag{14}$$

From equation (14), the reliability and mean time to failure can be obtained as follows :

$$R(t) = 1 - F(t), \tag{15}$$

$$T_m = \int_0^\infty R(t)dt. \tag{16}$$

2.4. Sensitivity of Reliability to the Physical Parameters

The sensitivity of the reliability to a parameter V can be derived by differentiating the reliability function by variable V. For $\delta_i \le t < \delta_{i+1}$, the sensitivity is as follows :

$$S_V = \frac{\partial R}{\partial V} = -\frac{\partial F}{\partial V}$$

$$= -\left(\frac{\partial f_1}{\partial V} + \sum_{k=1}^{i-1} \frac{\partial f_{k+1}}{\partial V} \cdot \prod_{j=1}^{k} (1 - f_j) \right)$$

$$- \sum_{k=1}^{i-1} f_{k+1} \cdot \sum_{l=1}^{k} \left(- \frac{\partial f_l}{\partial V} \cdot \prod_{\substack{j=1 \\ j \neq l}}^{k} (1 - f_j) \right). \tag{17}$$

For the sample software shown in Appendix, $\partial f / \partial p_h$ is given by

$$\frac{\partial f}{\partial p_h} = -\frac{1}{4} p_s^{58} p_h^{69} (73 p_s^3 p_h^3 + 72 p_s^2 p_h^2 + 71 p_s p_h + 70), \tag{18}$$

and

$$\frac{\partial p_h}{\partial M} = \frac{1}{M^2} (1 - e^{-\lambda(\delta_j - n_j\theta)}), \tag{19}$$

$$\frac{\partial p_h}{\partial \theta} = \frac{1}{M} \lambda n_j e^{-\lambda(\delta_j - n_j\theta)}, \tag{20}$$

$$\frac{\partial p_h}{\partial \lambda} = \frac{1}{M} (\delta_j - n_j) e^{-\lambda(\delta_j - n_j\theta)}. \tag{21}$$

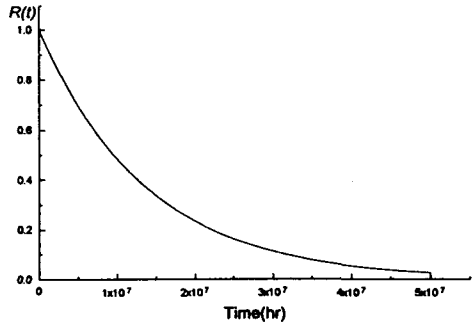

Fig. 5. Reliability of Software in the Normal Quality memory.

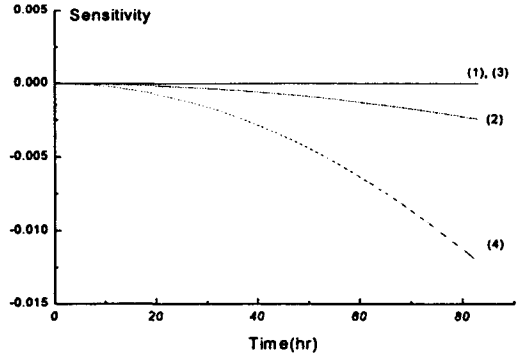

Fig. 6. The Sensitivity of the Reliability to the Parameters.

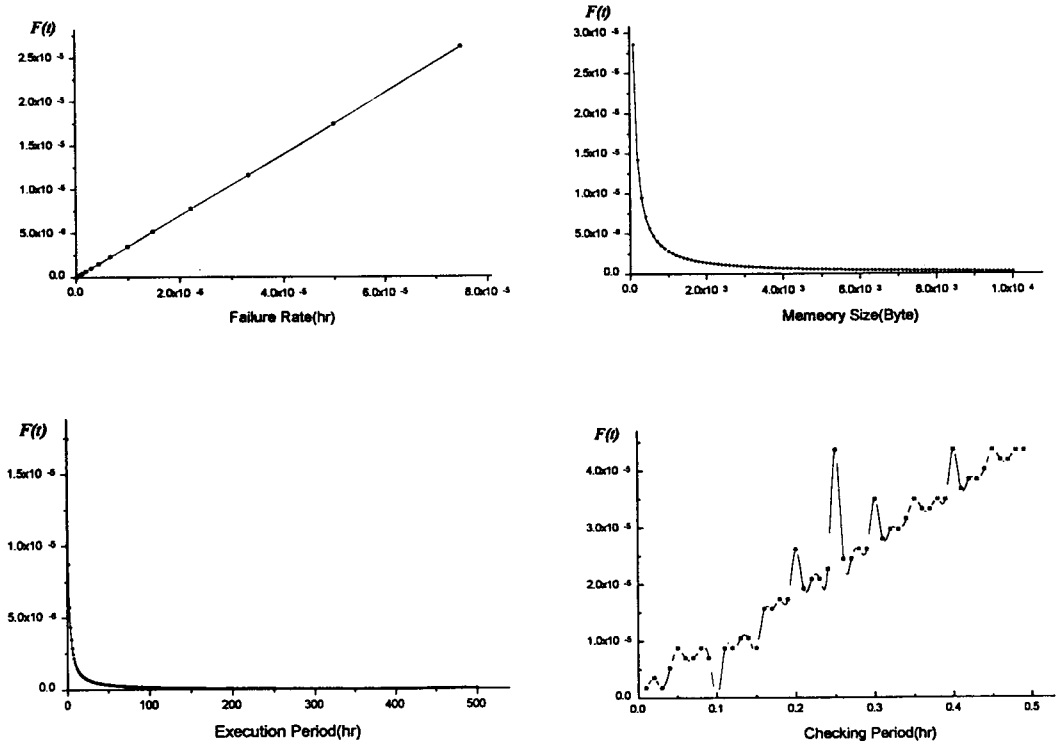

Fig. 7. CSFP of the Software to Some Physical Parameters.

3. Model Application

The target ILS software is a part of AFS-1000 system developed by Forney International Cooperation and installed in YGN nuclear power units 3 and 4. It is written in the Intel 8085 assembly language using top-down modular design techniques. Reliability of ILS software is predicted only considering the software failure by memory faults.

As shown in Table 1, the memory in the system is the Erasable Programmable Read Only Memory(EPROM) and the capacity of the memory is 64k bites. The clock frequency is 1Mhz. The sample program which is shown in Appendix is a part of the executive program that consists of the various subroutines which generate the logic to perform miscellaneous functions (ANDs, ORs, Counters, etc.). To apply the derived model to the program, we used physical parameters and assumptions as follows : M = 8192 bytes, θ = 5 minutes, $\omega \gg \tau$, p_s = 1 (i.e., the program has no software errors). It is assumed that $p(i, j)$ is equal for each program branch.

As described in Table 2, the MTTF of the software failure by only memory faults is predicted for four cases. When the software is operated with execution period, 0.5 hr, and failure rate, 10^{-4}/hr, the software fails once whenever the memory faults occur about 10^3 times. For the same software with execution period 500 hr, the failure occurs about every 10^6 memory faults..

Figure 5 shows the reliability of the software in normal quality memory and with a checking period, 0.5hr. The result shows that the reliability is exponentially distributed. Figure 6 shows the sensitivity of the reliability to the parameters as time goes on. In Figure 6, the curve (2) indicates the time dependency of the sensitivity of the reliability to the memory fault rate. It shows that the sensitivity increases in negative direction, that is, as time goes on, only the small increase of memory fault rate can affect the software failure significantly. Curves (1) and (3) show the cases for software execution period and for memory size, respectively. Time dependency of the effects of execution period and memory size to the software reliability seems to be negligible. The curve (4) shows that the effect of changing checking period becomes more significant as time goes on. From Figure 6, we know that the checking period is the major factor which affects the reliability of the software significantly.

Figure 7 shows the change of CSFP to some physical parameters at a given time, 1000 hrs. From Figure 7, we know that the increase of the memory fault rate lowers the reliability linearly. As the memory size becomes larger, the probability that the memory fault affects the software is decreased because the memory fault rate is constant, fault location is random, and the portion of the software in memory is decreased. The increase of the memory size, therefore, improves the reliability when the software size and memory fault occurrence rate are fixed. If the time interval between the software executions increases, the number of the program execution in a given time decreases. Therefore, the chance of the software failure is reduced and the reliability increases. The reliability fluctuates and decreases as checking period increases as shown in Figure 7. The fluctuation is due to the change of the interval between last checking time and program runtime. If the program runtime occurs right after memory checking, for example, the reliability is relatively higher, and if it occurs right before the memory checking, the reverse is true.

4. Summary and Conclusion

In this work, we suggested an analytic approach for the dependability evaluation of software in the operation phase with special attention to the physical fault effects on the software dependability. Generally, the safety critical software is required to have the failure rate under 10^{-7}/5000 hr[17]. This is the requirement for the software failure by only software error. As shown in the Table 2, the result shows that the software failure rate by only the memory faults is about the same order of magnitude or is greater than the value of this requirement. Interaction effects, therefore, should be considered for predicting the software dependability accurately in operation phase, especially for the software which has short execution period. In radiation environments such as in nuclear power plant, the memory fault occurrence rate will be higher and the interaction effect must be considered seriously. It was found that the major physical parameters that affect the software reliability are memory size, software execution periods, memory checking period and the memory fault occurrence rate. The major factor which affects the reliability of the software significantly is the memory checking period.

This modeling method is particularly attractive for medium size programs such as software used in digital systems of nuclear power plants. Without modification this modeling methodology can be extended to the software system which consists of several complete modules performing the independent functions. This work is believed to be also useful to obtain optimal value of physical memory parameters when the software is

implemented into the digital system.

Nomenclature

$f(p_h, p_s)$	probability of software failure.
p_h	probability that no memory faults occur in one machine instruction of memory.
q_h	probability that memory faults occur in one machine instruction of memory, 1-p_h.
p_s	probability that there is no software faults in one software instruction.
f_i	software failure probability at ith execution state.
$F(t)$	cumulative software failure probability(CSFP) of the software failure probability function $f(p_h, p_s)$.
$P(i, j)$	probability of entering the path (i,j).
$p(i \rightarrow j \mid i,j)$	probability of successfully executing path (i, j) given that the program has entered the path (i, j)
$p_{i,j}$	joint probability of entering and executing successfully the path (i, j).
δ_i	time at which $_i$ th program execution starts.
l_i	probability of the memory faults occurrence which affects ith software execution.
λ	memory fault interarrival rate.
θ	memory checking period.
ω_i	sum of the $_i$th execution time and the $_i$th idle time, $\tau_i + \zeta_i$.
τ_i	program execution time.
ζ_i	program idle time.
M	number of machine instructions which can be stored in memory.
S	size of software in the number of machine instructions.
T_m	mean time to failure.

Appendix

```
0000  MAN_AUTO:  LXI   H, FAOF
0003             LXI   D, FGPF
0006             MOV   B, M
0007             INX   H
0008             MOV   A, M
0009             INX   H
000A             MOV   C, A
000B             CMA
000C             ANA   B
000D             MOV   B, A
000E             LDA   FAOF+13D
0011             ORA   C
0012             ANA   M
0013             INX   H
0014             ORA   B
0015             ORA   M
0016             INX   H
0017             ANI   01
0019             JZ    AUTO_MAN1
001C             STAX  D
001D  AUTO_MAN1  MOV   B, M
001E             INX    H
001F             MOV   A, M
0020             INX   H
0021             ANI   01
0023             JZ    AUTO_MAN2
0026             XRA   A
0027             STAX  D
0028  AUTO_MAN2  LDAX  D
0029             INX   D
002A             MOV   C, A
002B             MOV   A, B
002C             CMA
002D             STAX  D
002E             INX   D
002F             MOV   A, M
0030             INX   H
0031             ANA   C
0032             ANA   M
0033             INX   H
0034             STAX  D
0035             INX   D
```

0036	MOV	A, M
0037	INX	H
0038	ANA	C
0039	ANA	M
003A	INX	H
003B	STAX	D
003C	INX	D
003D	MOV	A, M
003E	INX	H
003F	CMA	
0040	STAX	D
0041	INX	D
0042	MOV	A, M
0043	ORA	C
0044	STAX	D
0045	INX	D
0046	MOV	A, C
0047	CMA	
0048	STAX	D

References

1. P. Zeephongsekul, G. Xia and S. Kumar, "Software Reliability Growth Model primary failures generate Secondary faults under imperfect Debugging," IEEE Trans. Rel., Vol. 43, No. 3, pp. 408-413, Sep. (1994)
2. N.F. Schneidewind, "Software reliability model with optimal selection of failure data," IEEE Trans. Software Eng., Vol. 19, No. 11, Nov. (1993)
3. S. Yamada and S. Osaki, "Software reliability growth modeling : models and assumptions," IEEE Trans. Software Eng., Vol. Se-11, pp. 1431-1437, Dec. (1985)
4. N. Lopez-Benitez, "Dependability modeling and analysis of distributed programs," IEEE Trans. Software Eng., Vol. 20, No. 5, May (1994)
5. G.Q. Kemey, "Estimating defects in commercial software during operation use," IEEE Trans. Rel., Vol. 42, No. 1, pp. 107-115, Mar. 1993.
6. N. Karunanithi, P. Whitley and Y.K. Malaiyar, "Prediction of software reliability using connectionist models," IEEE Trans. Software Eng., Vol. 18, No. 7, pp. 563-574. July (1992)
7. J.C. Laprie, "Dependability evaluation of software in operation," IEEE Trans. Software Eng., Vol. SE-10, (1984)
8. B. Littlewood, "Theories of software reliability : How good are they and how can they be improved?" IEEE Trans. Software Eng., Vol. SE-6, pp. 489-500, Sep. (1980)
9. P.A. Deiller, B. Littlewood, D. R. Miller and A. Safer, "On the quality of software reliability prediction," NATO ASI Series, Vol. F3, Electronic Systems Effectiveness and Life Cycle Costing, Springer-Verlag, (1983)
10. B. Littlewood, "How to measure software reliability and how not to ···," IEEE Trans. Rel., Vol. R-28, pp. 103-110, Jun (1979)
11. K.K. Goswami and R. K. Iyer, "Simulation of Software Behivior Under Hardware Faults," Proc. On Fault-Tolerant Computing Systems, pp. 218-227, (1993)
12. J.C. Laprie and K. Kanoun, "X-ware reliability and availability modeling," IEEE Trans. Software Eng., Vol. 18, No. 2, pp. 130-147, Feb. (1992)
13. R.K. Iyer and P. Velardi, "Hardware-Related Software Errors : Measurement and Analysis," IEEE Trans. Software Eng., Vol. SE-11, Feb. (1985)
14. J.L. Roca, "A Method for Microprocessor Software Reliability Prediction," IEEE Trans. Rel., Vol. 37, Apr. (1988)
15. J.W. Duran and J.J. Wiorkowski, "Quantifying software validity by sampling," IEEE Trans. Rel., Vol. R-29, No. 2, pp. 141-144, Jun (1980)
16. M.L. Shooman, "Structured models for

software Reliability prediction," Proc. 2nd International Conf. Software Eng., San Francisco, pp. 268-280, (1976)

17. G. Leveson, "Software Safety : Why, What, and How," ACM Computing Surveys, Vol. 18, No. 2, June (1986)