Journal of the Korean Nuclear Society
Volume 31, Number 1, pp.49~57, February 1999

Prediction of Safety Critical Software Operational Reliability from Test Reliability Using Testing Environment Factors

Hoan Sung Jung
Korea Atomic Energy Research Institute
P.O.Box 105, Yusong-Gu, Taejon,305-600, Korea

Poong Hyun Seong
Korea Advanced Institute of Science and Technology
373-1, Gusong-Dong, Yusong-Gu, Taejon,305-701, Korea

(Received May 25, 1997)

Abstract

It has been a critical issue to predict the safety critical software reliability in nuclear engineering area. For many years, many researches have focused on the quantification of software reliability and there have been many models developed to quantify software reliability. Most software reliability models estimate the reliability with the failure data collected during the test assuming that the test environments well represent the operation profile. User' s interest is however on the operational reliability rather than on the test reliability. The experiences show that the operational reliability is higher than the test reliability. With the assumption that the difference in reliability results from the change of environment, from testing to operation, testing environment factors comprising the aging factor and the coverage factor are developed in this paper and used to predict the ultimate operational reliability with the failure data in testing phase. It is by incorporating test environments applied beyond the operational profile into testing environment factors. The application results show that the proposed method can estimate the operational reliability accurately.

1. Introduction

With the rapid increase of applications customers consider the software reliability as an important attribute of the computer system. It has been a critical issue to predict the safety critical software reliability in nuclear engineering area. Since the cost and the time required to develop software systems are being increased more than the hardware systems, increasing interest is given to the quality of the software systems. There are many experiences that the failures of the

Key Words : software, reliability, test, environment, prediction, safety-critical

computer systems have resulted from the faults in the software[1]. Since the software is an intellectual product created by human activity the failures are inevitable. The effects of failures are large and often critical. One can not imagine the impact of faults in the software systems used in safety-critical applications such as nuclear reactor protection, aircraft control, chemical process control, and military weapon control. There are many attempts to estimate the software reliability during development[2]. The model developed in a project show reasonable results for that specific project but it could not have consistency with other projects. Recently it is said that the quantification of reliability for safety-critical software is infeasible using statistical methods[3-6].

Many approaches measuring software quality were based on attempting to count faults or defects found in a program. These approaches are developer-oriented. What is usually counted are either failures or faults remaining or found. However, even if these are correctly counted, they are not good indicators of the quality. If the test environments are the same as the use environments, the test reliability must be equivalent with the operational reliability. The experiences show, however, that there is difference in reliability figure between at the test phase and at the operation phase[7]. If we can measure the discrepancy between these two environments quantitatively, the reliability at the operational phase could be predicted by that at the test phase.

In this paper, the testing environment factors that quantify the change of environment between the testing and the operation are developed in order to predict the operational reliability more accurately. Using these environment factors, the operational reliability of two example software are estimated and compared for verification with the existing failure data collected at the test phase. It is another merit of the proposed testing environment factors that these can be used with any software reliability prediction model without changing of basic models in order to incorporate the testing environments.

In this paper the mean time to next failure (MTTF) is used as a reliability measure. It permits one to analyze in common terms the effect on system quality of both software and hardware, both of which are present in any computer system.

2. Modeling for Operational Reliability Estimation

2.1. Selection of Reliability Growth Models

In this study the operational reliability is predicted based on the test reliability fitted with test data. The approach used is not dependent on the models fitted. There are a lot of models developed to estimate the test reliability. But no established model could predict the reliability accurately over the various projects. The models that estimate the reliability in figure of failure rate or fault count using failure counts only would be models to calculate the test reliability. The reliability is represented by mean time between failure (MTBF) and its inverse is the failure rate . However, in this paper, the MTBF equals with the mean time to next failure (MTTF) because the time domain is used in the execution time to ignore the time to repair.

2.2. Description of Testing Environments

Software reliability is defined as the probability of failure-free operation of a computer program in a specified period of time under specified conditions of operation. The specified environment refers to the operational profile or set of probabilities associated with the set of possible input states. Therefore, if a program is executed in a different environment, the reliability can be expected to change. The changed reliability is best analyzed in terms of failure intensities or its inverse, MTTF. It would be theoretically possible to analyze two operational profiles on a microstructure level, looking at probabilities of occurrence and failure intensities for each input state. Then the effect of a change in operational profile would be computed in a detailed fashion. However this would require an enormous amount of data collection and computation. This is the idea that changes between environments can be figured by special measures. The change results from the fact that input states of the operational phase are not equivalent to those of the test phase. As far as the test environment is not the same as the operation profile exactly, the reliability estimated from the testing executed during test phase will be defined as test reliability only.

Reliability represents a user-oriented view of software quality. Customers or users are interested in the operational reliability rather than in the test reliability. The historical results of some programs show that the failure intensity during use is lower than that during test phase. The failure intensity of the program during use phase is decreased by about one order of magnitude of the failure intensity at the end of the test[7].

After the integration of the software modules, the integration testing is executed. Functional and system tests are followed after the completion of integration test to validate the program correctness. Functional test is the process attempting to find discrepancies between the program and its external specification. An external specification is a precise description of the program' s behavior from the user' s point of view. Functional testing is normally a black-box-oriented activity. To perform a functional test, the specification is analyzed to devise a set of test cases. The equivalence partitioning, boundary-value analysis methods are used to generate test cases. Because the purpose of the functional test is to expose errors, there are a large number of invalid and unexpected input conditions. System test is the process attempting to demonstrate how the program does not meet its objectives. Because the purpose of the system test is to compare the program with its objectives, there are no formal methods to design test cases. The reason for this is that objectives state what a program must do, and how well the program must do it, but do not state the representation of the program' s function. Therefore, in order to show that the program is inconsistent with each sentence in the objective' s statement, a number of tests are performed. The input states in these tests occur seldom in normal operational environment.

A number of categories of tests applicable during the system test and its objectives are as follows[8]:

Volume testing : to show that the program can not handle the volume data specified in its objectives.

Stress testing : to subject the program to peak volume of data at instance to verify its capability Performance testing: to verify the performances such as response time, throughput rates under certain workload and configuration conditions *Storage testing* : to demonstrate the storage objectives of the program

Configuration testing : to demonstrate all the possible configurations may be occurred

Compatibility/Conversion testing : to demonstrate that the compatibility objectives have not been met with, and that the conversion procedures do not work from, the existing system

Installability testing : to test the complicated procedures for installing the system

Reliability testing : to demonstrate that the program meets its reliability objectives Recovery testing: to demonstrate that a program recovers from programming errors, hardware failures, and data errors

Serviceability testing : to test serviceability or maintainability characteristics of a program

Documentation testing : to demonstrate the accuracy of the user documentation Procedure testing: to test any human procedures required by programs

2.3. Development of Testing Environment Factors

The differences in environment between the testing phase and the operational phase are characterized by the input states and execution time (or number of runs). In this study, the differences are defined as *Testing Environment Factors, F_{TE}*. The operational reliability can be predicted from the estimated test reliability using testing environment factors.

In addition to the assumptions made in the models for the test reliability, the following assumptions are made to define the testing environment factors:

- Despite that the tests are performed based on the operational profile, the failure rate of the software during test is higher than during operation.
- The cumulative number of failures is proportional to the test coverage.
- The cumulative number of failures is dependent of the number of program execution (or execution time).

The first assumption, higher operational reliability, is a description of real world in a statistical sense. As shown in the software reliability data book published by Musa[7], in general, the failure rate during operation period is lower than that during test period. The second assumption that the failure rate is proportional to the test coverage holds when the errors are homogeneously distributed and test cases are randomly selected from input space. The last assumption that the failure rate is dependent on the number of program execution time means that the software reliability grows continuously during testing.

Based on the above assumptions, the following lemma can be set up:

The failure rate of the software during operation is lower than that during test due to testing environment factors, F_{TE}.

The operational reliability can be predicted by multiplying the factor F_{TE} that quantifies the environment change to the estimated test reliability as following:

$$MTTF_{use} = MTTF_{test} \times F_{TE} \qquad (1)$$

The testing environment factors is the combination of two factors, aging factor α and coverage factor V, and defined in two aspects, the number of execution and the test coverage. It is denoted as

$$F_{TE} = \alpha + V \qquad (2)$$

The first factor, aging factor (α) is a measure of relative number of execution times of the more probable functions in the invalid input space in comparison with the ordinary operation of the program. During the test, more invalid and unexpected input states compared to the normal operation are applied to the program. Since these invalid or unexpected inputs are effective in detecting errors, it can be said that the same portion of failures is detected by these inputs. And these inputs are seldom executed in the operational phases. The valid input distribution is not changed in both test and operational phase. The aging factor is the ratio of execution time required in the operational phase to execution time required in the test phase to cover the invalid input space of the program.

If τ_k be the execution time for the run corresponding to an input state k, and this input state occurs with probability p_k during operation. Then [9]

$$\alpha = \frac{\sum_{k=1}^{S_I} p_k \tau_k}{p_{\min} \sum_{k=1}^{S_I} \tau_k} \tag{3}$$

where

$\sum_{k=1}^{S_I} p_k = 1$,

τ_k = the execution time for input state k,

p_k = the probability of occurrence of input state k,

$p_{\min}$ = the probability of input state that occurs during operational phase,

S_I = total number of invalid input states.

If all runs are of equal length, the aging factor is simplified to

$$\alpha = \frac{1}{S_I p_{\min}} \tag{4}$$

In order to represent a wide range of the operation profile, we assume the probability p_k has the form $p_k = ak^b$, $b \leq 0$ and the input states are ranked. Then,

$$a = \frac{1}{\sum_{k=1}^{S_I} k^b} \tag{5}$$

and input state S_I will occur with probability $p_{\min}$. For input state $k = S_I$, by substitution,

$$p_{\min} = \frac{S_I^b}{\sum_{k=1}^{S_I} k^b} \tag{6}$$

If we assume that the probability of selection is inversely proportional to rank order, the $b = -1$ and

$$\alpha = \sum_{k=1}^{S_I} \frac{1}{k} \tag{7}$$

The summation of series can be written in terms of *Psi* (digamma) function[10]:

$$\alpha = \Psi(S_I+1) - \Psi(1) \tag{8}$$

where $\Psi(1) = -0.5772$.

Musa has done similar approach using testing compression factor, but he counts all the input states[7].

The second factor, coverage factor (V), is the measure of the degree of how much test case covers the system test categories beyond the operational profile (function test). By assuming that the system test cases are randomly selected from the system test input space, the probability that a test category exercises *k* new features which have not been exercised after covering *l* categories more than once is given by the Hyper Geometric Distribution[11-12].

$$prob(k|l) = \frac{{}_{L-l}C_k \times {}_{l}C_{p-k}}{{}_{L}C_p} \tag{9}$$

where

L = the number of categories would be tested in a program,

C = the symbol of Combination in the mathematics(aCb= b!/{a!(b-a)!}),

p = the average number of categories covered by a test category during the system test.

The expected number of categories newly exercised by the *i* th test category is given by

$$v_i = \sum_{j=1}^{p} j \times prob\{j|l_{i-1}\} \tag{10}$$

where l_{i-1} is the number of categories covered before running the i th test category.

The l_i is recursively estimated using v_j as follows:

$$\overline{l_i} = \sum_{j=1}^{i} v_j,\ v_1=p,\ \text{and}\ \overline{l_1} = p \tag{11}$$

Therefore, we get

$$v_i = p \times \frac{L-\overline{l_{i-1}}}{L} = \frac{p}{L}[L-\overline{l_{i-1}}] \tag{12}$$

Since $\overline{l_i}$ can be regarded as an integration of v_i, we can formulate a continuous approximation function as:

$$\frac{d}{dx} l(x) = \frac{p}{L}[L-l(x)] \tag{13}$$

where x is the number of test categories executed. By solving this equation, we get

$$l(x) = L(1-e^{-\frac{p}{L}x}) \tag{14}$$

where $l(x)$ is the number of categories covered by the system testing and the value of it is the coverage factor, V in the equation (2).

3. Application

The data sets compiled by Musa[7] provide a convenient measure of assessing performance of models for software reliability. Among them the two data sets that comprise system testing data and operational data for system #1 and system #2 are used for applying the models developed in this study. The system #1 and the system #2 are real-time command and control systems and perform complex logic and control with some command interpretation. The systems were written in assembly language and had total 21700 and 27700 instructions, respectively. The data were reported as time intervals between failures in execution time in unit of second. Musa said test cases were selected not only from a complete set of use input sets but also from extreme conditions. One to three display consoles with push button and light pen and repertories of

Table 1. Software Failure Data

Phase	Execution time [hour]	Failures	
		System #1	System #2
Test phase	1	27	10
	2	16	6
	3	11	4
	4	10	5
	5	11	2
	6	7	1
	7	2	1
	8	5	1
	9	3	0
	10	1	1
	11	4	3
	12	7	7
	13	2	1
	14	5	0
	15	5	0
	16	2	0
	17	0	0
	18	5	4
	19	1	0
	20	1	1
	21	2	1
	22	1	0
	23	2	1
	24	1	1
	25	1	1
	26	use	0
	27	-	1
	28	-	0
	29	-	1
	30	-	0
	31	-	1
	32	-	0
	33	-	0
Use phase	0-29.1	2	-
	0-62.7	-	2

displays were used in system testing. The raw failure data are grouped into numbers of failure per hour of execution time to have independence of data in this study. Table 1 shows the grouped data for two systems.

The two models, exponential and logarithmic, are applied to estimate the test reliability. For the exponential and logarithmic models, the parameters are estimated using maximum likelihood method. To select a specific model from two models for the specific system, the sums of squared residuals of failure rates of last 5 hours are compared. From the comparison of the results, the exponential model and the logarithmic model are selected for system #1 and system #2, respectively.

The fitted models for cumulative failures based on the data are as follows:

$$\text{system \#1}: \mu(t) = 142.3(1-e^{-0.1246t}) \quad (15)$$

$$\text{system \#2}: \mu(t) = 16.13\ln(1+0.829t) \quad (16)$$

From the fitted model, the mean time to next failure, MTTF, at the end of the system test is calculated as follows:

system #1 : $MTTF_{system\ \#1test@fitted} = 1.27$ [hours]
system #2 : $MTTF_{system\ \#1test@fitted} = 2.117$ [hours]

The aging factor, the first factor of testing environment factors, is calculated based on the number of test cases estimated from the size of the program. It is assumed that the typical number of test cases is 1(one) per 10 lines of code[11]. If we assume the portion of invalid test cases is about 50% of the input space, the numbers of invalid test cases (S_i) for the system#1 and system#2 are about 1085 and 1385, respectively.

Using the equations developed in the previous section, the aging factors are calculated as follows:

system #1 : $\alpha = 7.56$
system #2 : $\alpha = 7.81$

The coverage factor (V), the second factor of the testing environment factor, is obtained from the number of feature tests applied during system test. It is assumed from the description of system test environment that such tests as stress, recovery, procedure, storage, and security tests are applied during the system test. The test environment for both systems is assumed to be the same. The coverage factor, V, is 4.1 for both systems with the values, L =12, p = 1, x = 5, if we assume that one kind of system test covers only one category of the program feature and 5 kinds of tests from the aforementioned 12 tests have been conducted during the system test. The testing environment factors for the systems are $F_{TEsys\#1} = 11.66$ and $F_{TEsys\#2} = 11.91$.

For the purpose of comparison, the actual values of operational reliability (MTTF) are obtained by averaging the failure intervals measured at the use phase.

The actual MTTF at the operational phase are as follows:

$MTTF_{sys\ \#1use@measured} = 14.6$ [hours]
$MTTF_{sys\ \#2use@measured} = 31.4$ [hours]

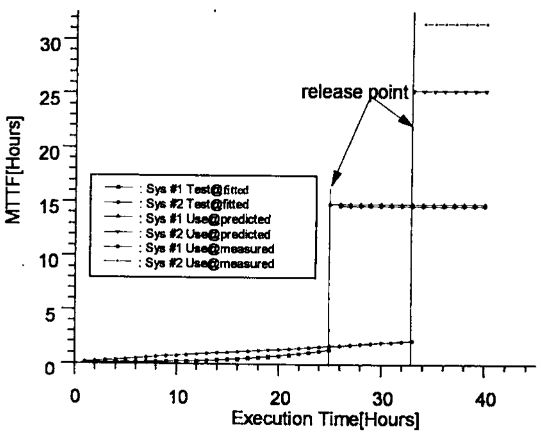

Fig. 1. MTTF Plot

The predicted MTTF of the operation phase with the calculated testing environment factors are as follows:

$$MTTF_{sys\ \#1use@predicted} = 14.8\ [\text{hours}]$$
$$MTTF_{sys\ \#2use@predicted} = 25.21[\text{hours}]$$

Figure 1, MTTF plot, shows these results graphically. The application results are shown to be consistent with the actual data. The predicted MTTF' s at the use periods are close to measured values. Musa proposed similar approach by defining the test compression factor using all the input space. MTTF' s predicted by Musa for the system#1 and the system#2 are 20.4 hours and 43.5 hours respectively. Musa' s approach overestimates the operational reliability in both cases. He says in his text that it is due to the repetition of input states [9].

4. Conclusions

In this paper the testing environment factors consist of aging factor and coverage factor are developed and the operational reliability of two examples is estimated using the proposed testing environment factors and the failure data collected at the test phase. It is verified with two application examples that the predicted values of MTTF are closer to the actual values than the reliability growth model only approach.

Software in safety systems of nuclear power plants is tested with various test cases which reflect not only the normal operational environments but also the extreme environments[13-14]. But the some efforts and effects is not considered in software reliability models explicitly if it does not reveal the failures. And the difference in values of reliability between at the test phase and at the operational phase is not well explained. With this approach we can estimate more reasonably the operational reliability of software systems which are used in safety-critical applications such as nuclear power plants, airplanes, and military facilities using the testing environment factors. It is the another merit of this approach that it estimates the test reliability and the operational reliability simultaneously. Since the method developed in this paper does not depend on any specific software reliability model for estimating the test reliability, we can apply this approach to any other software systems at the end stage of development by quantifying these testing environment factors reasonably.

References

1. S. Yamada et. al., "Software reliability measurement and assessment based on nonhomogeneous process models : a survey", Microelectronics and reliability, Vol.32, No.12, pp.1763-1773, (1992).
2. Y.K. Malaiya and P.K. Srimani, "Software reliability models; theoretical developments, evaluation & applications", IEEE Computer Society Press, (1991).
3. J.A. McDermid, "Issues in developing software for safety critical systems" , Reliability Engineering and System safety, Vol.32, No1&2, pp.1-24, (1991).
4. R.W. Butler and G.B. Finelli, "The infeasibility of quantifying the reliability of life-critical real-time software", IEEE Trans. Software Eng., Vol.19, No.1, pp. 3-12, (1993).
5. D.L. Parnas et. al., "Evaluation of safety-critical software", Communications of ACM, Vol.33, No. 6, pp.636-648, (1990).
6. B. Littlewood and L. Strigini, "Validation of ultrahigh dependability for software based system", Communications of ACM, Vol.36, No. 11, pp.69-80, (1993).

7. J.D. Musa, "Software reliability data", Bell Lab., (1979).
8. G.J. Myer, " The art of software testing", A Willy-Interscience Pub., (1979).
9. J.D. Musa, A. Iannino, and K. Okumoto, "Software reliability; measurement, prediction, application", McGraw-Hill, (1987).
10. H.T. Davis, "The summation of series", The Principia Press of Trinity University, San Antonio, Texas, (1962).
11. P. Piwowarski, M. Obha, and J. Caruso, "Coverage measurement experience during function test", Proc. IEEE Software Engineering, pp. 287-301, (1993).
12. Y. Tohma et.al., "Structural approach to the estimation of number of residual software faults based on the hypergeometric distribution", IEEE Trans. Software Eng., Vol.15, No.3, pp. 345-355, (1989).
13. "Sizewell B reactor protection reliability: Nuclear Electric presents its case", Nuclear Engineering International, pp. 28-33, Mar. (1993).
14. D. Welbourne, " NNC gives high marks to Sizewell B' s primary protection system", Nuclear Engineering International, pp. 34-35, Mar. (1993).