ISSN:2320-0790

An International Journal of Advanced Computer Technology

MODULAR TEST PLAN FOR INFERENCE ON SOFTWARE MAINTENANCE BASED ON MARKOV MODEL

Mahtab Alam
Department of Computer Science, Noida International University, Greater Noida

Abstract: In software industry, the high cost of maintenance of large-scale software put emphasis on the need to manage the maintainability in earlier phases of software system. This paper considers the assessing of the maintenance problem of a software system that can be decomposed into a finite number of modules. It uses a Markov model for transfer of control from one module to another module in order to develop the system for maintainability by detecting and fault module and then repair this module to operate. An operational test procedure is considered in which only the fault modules are maintained and the system is considered reliable if and only if no faults are observed during testing. The minimum number of test required of each module is determined such that the probability of accepting a system whose maintainability falls belong a specified value.

Key Words: Software Maintenance, Software Testing, Operation Profile, Exponential repair time, Steady state.

I. INTRODUCTION

Object oriented design and programming is the dominant development paradigm for software system today. With the growing complexity and size of object-oriented systems, the ability to reason about quality attributes based on automatically computable measures has become increasingly important [3]. Maintainability of software is the degree, to which it can be understood, corrected, adapted and/or enhanced [1]. To maintain a system with less effort and minimum cost is the primary objective of any engineering discipline. Maintainability of software is not an easy task. Data gathered over the past few decades have indicated that software developers can spend as much as 75 percent of their total budget on software maintenance [2]. In other words, software maintenance is the most costly phase of the software life cycle. Over the last several years, characterizing, measuring and evaluating software maintainability have become crucial activities when a large computerized system is on progress. In this particular paper, we addressed one specific aspect namely, Operational testing for software maintainability, where the objective is to find an estimate of the software maintainability actually achieved. For this type of testing system is subjected to the some statistical distribution of inputs that one can expect it to encounter in operation. The procedure is based on Markov model, consists of conducting test only on the modules that comprise the system and in order to draw inferences on the system maintainability. It uses the mathematical model that relates the system reliability to the other component and the operational profile.

In general this paper focuses on operational testing at the module level. It uses Markov model of the transfer of control from one module to another module in order to detect and locate the faulty module and erect them.

In the following sections, we first present a brief discussion of software testing and over need some of the statistical models developed for this purpose that are relevant to the work contained here. This is followed by the test plan result in a three state of software in Markov chain. At last the brief conclusion of this paper is summarized.

II. SOFTWARE TESTING

A software system is a collection of programs and system files such that the system files are accessed and altered by the programmer [4]. Each elements in this collection is called as modules- for example, a module might be a program, a subprogram or a file. The performance of the entire system depends on that of each module and their inter relationship on which they are functioning.

Testing take place almost every stages throughout the entire life cycle of any software system. However, there are different type of test associated with different phases, such as design based testing, functional based testing, operational testing, debug testing, failure testing, and load testing. Testing of the entire system is compulsory; it

stands to reason that a sound and well planned program of testing at the module level has the potential to reduce the effort involved in system level testing. Modular testing allow for more flexibility since it can be done at different times and locations. The key point which must be kept in mind is that the test plans for the individual modules must be system based and designed in such a way that statistically valid inferences may be drawn about the overall system from the results of module test.

Markov chain model describes the switching between the modules within a software system.[5] In this paper, we consider the failure caused by the "interfaces" between any two modules, that is, those failures that are introduced when modules are put together in a larger system, and investigate the effects of shared use on the maintainability of the modular software. For this purpose, we model the distribution of user demands at the module level by a continuous time Markov chain which is referred to as the operational profile. This work is related to the classical statistical concepts related to the testing of hypothesis in a three state software process in which the transitions form a given state to another state can take place at any instant of time.

To approach the problem of drawing statistically valid inferences about a system based on tests of its constituent's modules, one requires a mathematical model that expresses software system availability in terms of the module reliability. Here, we developed a model for software availability and we follow the approach of system based component testing.

III. SOFTWARE MAINTAINABILITY

Object oriented design and programming is the dominant paradigm for the development of any software in recent scenario. With the growing complexity and size of object oriented systems, the ability to reason about quality attributes based on automatically computable measures has become increasingly important. Several software quality attributes such as functionality, various individuals and standardized bodies have defined usability, portability, reliability, availability and maintainability. e.g. Maintainability is a special interesting quality attribute as it has been recognized that a software maintenance activities accounts for the 70% cost in today's software development [6]. However, maintainability is very difficult to estimate. Maintainability is a set of activity performed when software undergoes modification to code and associate documents due to problem or need for improvement. Maintainability of software can be viewed in a fashion similar to that of hardware or any general system. It may be defined as a function of time; it is the probability of failure free operation for some specific mission time under specific condition. Alternatively, it may be viewed from the perspective of general use on a variety of different inputs; in this case, it is the probability that it will correctly process randomly chosen inputs. The precise nature of the application on which software system will be used is not known in advance, we must quantify software use by using a suitable distribution or operational profile. The operational profile and the program structure allow us to develop a statistical distribution of inputs for each individual module and in the modular testing procedure being considered here. Associated with each type of input to the system, there is a specific path of modules over which control is transfer by the system. The transfer of control between modules takes place according to Markov chain. The probability p_{ij} that control transfers from one module i to another module j is independent of how module i was entered. It is supposed that there are n modules where modules i represent the initial state (i.e. it could be main program called by the user) and it is also assumed that when the program successfully completed, control is transferred to a terminal modules S. (it could be the operating system) with probability P_{is} of being entered from state I. Note that

$$p_{is} + \sum p_{ij} = 1$$

This, probability defined for a completely reliable system. Since we assume a system that is presumably not so, it is assumed that each module in the system has some fault and that module I has reliability r_i i.e. p (system fails any time control enters module i) = $(1 - r_i)$. An additional state F is defined known as failure of program and since each module is not 100% reliable, therefore, this state can be entered from any modules. It is also note that unlike the transient states (probability that the control will not return to this state) 1,2,3,…..,n, the state S and F are absorbing states (iff $P_{ij} = 1$) and represent successful completion of the program and failure. The Markov chain has thus n+2 states and transient matrix p where $Sij = r_i P_{ij}$ for i = 1,2,3…n and j = 1,2,3,…n and $S_{iF} = 1-r_i$ for i = 1,2,…n and $P_{FF} = P_{SS} = 1$ with all other $P_{ij} = 0$.

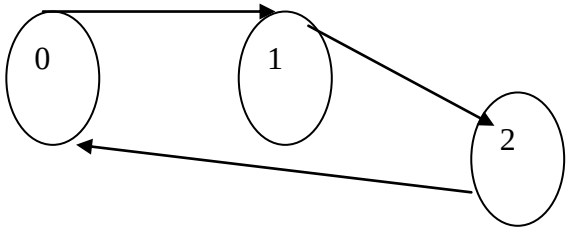

Figure-1 : State Transformation Phase

From the **Figure-1**, the absorbing state S_5 there is no outgoing edge. Then $P_{5j} = 0$ for all j. but the assumption of Markov chain required that $\Sigma P_{ij} = 1$ for each I and j. to avoid this problem we imagine a dummy self-loop. The transition matrix P corresponding to this system is given below:

$$P = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ S \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & S & F \end{array} \\ \begin{pmatrix} 0 & .6 & .4 & 0 & 0 & 1\text{-}r1 \\ 0 & 0 & .6 & 0 & .4 & 1\text{-}r2 \\ 0 & .2 & 0 & .4 & .4 & 1\text{-}r3 \\ 0 & 0 & .6 & 0 & .4 & 1\text{-}r4 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

The transition probability matrix of such a chain may be partitioned so that
P = | …Q/O…….| …..C/1…|

Where Q is a an (n-1)X(n-1) sub stochastic matrix (with at least one row sum less than 1), C is a column vector and O is a row vector of (n-1) row. Now the k^{th} stop transition probability matrix P^k has the form

P^k = |...Q^k/O… |…C^k/1…|

Where C_k is a column vector whose elements will be no further use and hence need not be computed.

The (i,j) entry of matrix Q^k denotes the probability of arriving in state S_j after exactly k stops starting from state S_i. It can be shown that ΣQ^k (k = 0 to i) converges as I approaches infinity. This implies that the inverse matrix $(1-Q)^{-1}$ called the fundamental matrix M exists and is given by

$M = (1-Q)^{-1} = 1 + Q + Q^2 + \ldots\ldots\ldots + = \Sigma Q^k$ (k = 0 to infinity). The fundamental matrix M is rich source of information on the Markov chain, and is very useful to detect the fault. The three state model of component failure-repair assumed that the failure and repair time distributions are both exponential. Assume now that the repair process can be broken down into two phases. First, fault detection and location and second, fault correction. These two phases have exponential distribution with mean $1/\mu_1$ and $1/\mu_2$ respectively. Consider a module with failure rate λ. Upon failure, it is repaired with an exponential repair time distribution of parameterμ. The steady state availability is the steady state probability that the system in condition of use. We can transform the given system in **Figure-2**. Define the three states as according:

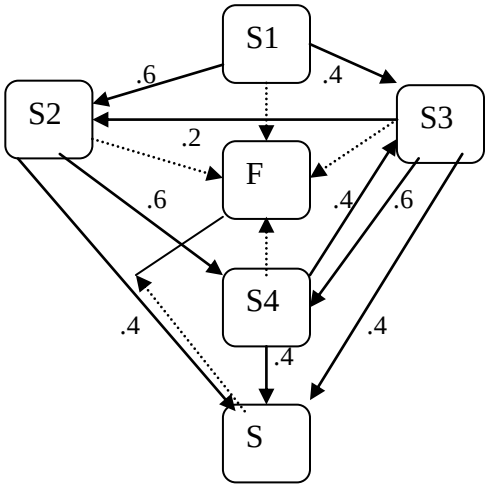

Figure: 2 Modular Software Systems with Transition Probabilities

0: The components are in working state.

1: The components are in detection-location state.

2: The components are in final repair state.

The steady state availability can be computed by first writing down the balance equations as follows:

$\lambda p_o = \mu_2 p_2, \quad \mu_1 p_1 = \lambda p_o, \mu_2 p_2 = \mu_1 p_1$

This yields the relation

$p_2 = (\mu_1/\mu_2)p_1 = (\mu_1/\mu_2)(\lambda/\mu_1)p_o = (\lambda/\mu_2)p_o$

Now since $p1 + p2 + p3 = 1$, we have $p_o = 1/[1+ (\lambda/\mu_1) + (\lambda/\mu_2)]$

The result can be extended to the case of the k-phase hypo exponential repair time distribution with probability $\mu 1, \mu 2, \ldots\ldots \mu k$ with the result

$A = p_o = = 1/[1+ (\lambda/\mu_1) + (\lambda/\mu_2) + \ldots\ldots\ldots + (\lambda/\mu_k)]$

If we denote the average repair time by $1/\mu$, then we have

$1/\mu = \Sigma 1/\mu_I$ (where $1 \leq i \leq k$)

It should be mentioned that this method of computing the availability expression as given in equation (1) can be quite inconvenient without the use of software that can be perform the requisite symbolic linear algebra.

IV. AN OPERATIONAL TEST PLAN

We can compute the availability of software by maintaining the fault module using the approach described in the preceding section. We now develop a procedure for drawing inferences on system availability by test on its

individual module. Consider a parallel redundant system with N components, each with a constant failure rate λ. The system is not available for use when ever all N components have failed and are waiting for repairs. We wish to compare the following designs of repair facility.

a) Each component has its own repair facility with repair rate μ. Then the vailability of an individual component is given by.

$$= 1/(1 + \lambda/\mu) = 1/(1 + p)^N$$

and the availability of the system is computed as

$$A_1 = 1 - (p/(1+p)$$

In this scheme, no machine has to wait for a repair facility.

b) We can economize the repair facility and share a single repair facility of rate μ among all N machines. In this case the system is down onle when all the components are undergoing repair. We compute the steady state availability by

$$A_{11} = 1 - p_N$$

$$= p^N N!/ \Sigma pk[N!/(N-k)!],$$

$$(\text{where } 1 \leq i \leq k)$$

c) We may speed up the rate of the reapir facility N/μ, while retaining a single repair facility.

$$A_{111} = [1 - (\lambda/N\mu)^N N!/ \Sigma(\lambda/N\mu)^k\{N!/(N - k)!\}]$$

Table – 1 shows that the value A1, A11, and A111 for various values of N assuming that λ/μ = 0.1

Table : 1**: Availabilities for parallel redundant system**.

N (Number of Components)	Individual repair Facility A1	Single repair facility of rate μ A11	Single repair facility of rate Nμ A111
1	0.909091	0.909091	0.909091
2	0.991736	0.983607	0.995475
3	0.999249	0.995608	0.999799

This table depicts that for each case the maintainability of a system fails below a small fraction α which is less than 1.

V. CONCLUSION

Using a particular Markov model for software availability, we have provided a minimum number of tests required for the different modules, such that there is no more than a small pre specified probability of accepting a software system whose availability is less than a specific value. It is worth mentioning that the number of tests is based on the assumption that the system will not work even if a single failure occurs in the module test, the procedure is likely to produce results that are quite stringent in character.

VI. REFERENCES

[1] "*Software Engineering, A Practical Approach*", 4th Edition, Mc. Graw Hill, 1997.

[2] Dimitris Stavrinoudis, 1999, "*Relation Between Software Metrics and Maintainability*," Proceedings of the FESMA (Federation of European Software Measurement Association), Amsterdom, International Conference, pp. 465-470

[3] Melis Dagpinar, Jens H. Johnke, 2003, "*Predicting Maintainability with Object Oriented Metrics – An Empirical Comparison",* Proceedings of the 10th working conference on Reverse Engineering,

[4] Rajgopal Jayant, Mainak Mazumdar, 2002, "*Modular Operational Test plans for Inferences on Software Reliability Based on Markov Model*", IEEE Transaction on Software Engineering, Vol. 28, No.-4, pp 358-363

[5] Trivedi K.S. 1982, "*Probability, Statistics with Reliability, Queing and Computer Science application*", PHI Publication, Englewood Cliffs, NJ1982

[6] Alam Mahtab, Abdul Wahid, R.A. Khan, 2005, "*Reducing Software Mintenance Cost and Effort – A Metric Prospective*", Proceedings of RAFIT05 (Recent Advanced and Future Trends in IT), National Conference, Patiala, pp.198-200.