Simulator for Software Maintainability

P. K. Suri[1], Bharat Bhushan[2]

[1]Department of Computer Science & Applications, Kurukshetra University, Kurukshetra (Haryana) India,
[2]Department of Computer Science & Applications, Guru Nanak Khalsa College, Yamuna Nagar (Haryana), India

Abstract
According to IEEE Standard Glossary of Software Engineering Terminology: maintainability is the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. [1]. Maintainability can also be defined as the probability that a specified maintenance action on a specified item can be successfully performed (putting the item into a specified state) within a specified time interval by personnel of specified characteristics using specified tools and procedures [2].
Software under maintenance consists of finite number of states. The states have a specific operating efficiency. The maintenance process can bring the software from one sate to another within a specific time slot allotted to the software maintenance engineers. The software fails or reaches its maximum efficiency depends upon the nature of maintenance problems. In this paper an attempt has been made to develop a simulator to compute n–step transition probabilities successively until the software reaches steady state. This process is very much depicted by Markov analysis [3]. The software simulation tool designed here will be helpful for the software project managers in judging the maintenance efforts of the software.

Keywords:
Software Maintenance, *Markov process, Operating Efficiency, Transition probabilities,Software complexity*

INTRODUCTION

The purpose of software maintenance is to assure the quality of performance of the respective software. But design errors, undiscovered faults, & installing new applications can cause the software degradation [4]. There are two aspects of maintainability: serviceability (the probability of returning the item to normal service) and reparability (the probability of repairing the actual or impending fault). Whenever we talk about software maintainability, our discussions are usually about reparability. In software engineering, the main emphasis of maintenance is change or the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.
Rajiv D. Banker j Gordon B. Davis j Sandra A. Slaughter[5] estimated the impact of development activities in a more practical time frame. They developed a two-stage model in which software complexity is a key intermediate variable that links design and development decisions to their downstream effects on software maintenance. They analyzed the data collected from various software enhancement projects and software applications in a large IBM COBOL environment. Results indicated that the use of a code generator in development is associated with increased software complexity and software enhancement project effort. The use of packaged software is associated with decreased software complexity and software enhancement effort.

Pfleeger [6] describes maintainability as the probability that a maintenance activity can be carried out within a stated time interval, it ranges from 0 to 1.

Rikard Land [4] investigates how the maintainability of a piece of software changes as time passes and it is being maintained by performing measurements on industrial systems.

F. Niessink [7] discussed the perspectives on improving software maintenance. He described process improvement of software maintenance from two perspectives: measurement-based improvement and maturity-based improvement.

Y. Kataoka, T. Imai .H. Andou T. Fukaya [8] discussed program refactoring as a technique to enhance the maintainability of a program. They proposed a quantitative evaluation method to measure the maintainability enhancement effect of program refactoring. They used coupling metrics to evaluate the refactoring effect. By comparing the coupling before and after the refactoring, they evaluated the degree of maintainability enhancement. Their results showed that their method was really effective to quantify the refactoring effect.

The software to be maintained may be considered to be in a number of states of deterioration .The maintenance (repair) work of the software is inspected after a regular interval of time say weekly and is classified as being in one of the states.. Each state is considered as functionally

Manuscript received November 5, 2007
Manuscript revised November 20, 2007

independent. The evaluation is carried out using Markov analysis which looks at a sequence of states and analyses the tendency of one state to be followed by another, after each repair the software restored to a state having 'increased' operating efficiency. Using this analysis one can generate a new sequence of random but related states which look similar to the original. This Markov process is stochastic in nature which has the property that the probability of transition from a given state to any future state depends only on the present state and not on the manner in which it was reached

If $t_0 < t_1 < t_2 <\ldots\ldots\ldots< t_n$ represents the points in time scale then the family of random variables $\{X(t_n)\}$ is said to be a Markov process provided it holds the Markovian property :

$$P\{X(t_n) = x_n \mid X(t_{n-1}) = x_{n-1}, X(t_0) = x_0\} = P\{X(t_n) = x_n \mid X(t_{n-1}) = x_{n-1}\}$$

$$\forall \quad X(t_0), X(t_1),\ldots,X(t_n)$$

Markov process is a sequence of n experiments in which each experiments has n possible outcomes x_1, x2,……,x_n. Each individual outcome is called a state and probability (that a particular outcome occurs) depends only on the probability of the outcome of the preceding experiment. The simplest of the Markov processes is discrete and constant over time. It is used when the sequence of experiment is completely described in terms of its states (possible outcomes). There is a finite set of states numbered 0,1, 2, 3, ….n and this process can be only in one state at a prescribed time. The system is said to be discrete in time if it is examined at regular intervals eg. Daily, weekly, monthly or yearly.

Transition Probability

The probability of moving from one state to another or remaining in the same state during a single time period is called transition probability.
Mathematically, the probability

$$P\, x_{n-1,}\ x_n = P\{X(t_n)= x_n \mid X(t_{n-1})= x_{n-1}\}$$

is called the transition probability. This represents the conditional probability of the system which is now in state x_n at time t_n provided that it was previously in state x_{n-1} at time $t_{n-1.}$ This probability is known as transition probability because it describes the system during the time interval $(t_{n-1,}\ t_n)$. Since each time a new result or outcome occurs, the process is said to have stepped or incremented one step. Each step represents a time period or any other condition which would result in another possible outcome. The symbol n is used to indicate the number of steps or increments.

The transition probability can be arranged in a square matrix form denoted by

$$P = \begin{pmatrix} p_{00} & p_{01} & p_{02}\ldots\ldots\ldots\ldots\ldots & p_{1n} \\ p_{10} & p_{11} & p_{12}\ldots\ldots\ldots\ldots\ldots & p_{1n} \\ & . & . & . \\ & . & . & . \\ & . & . & . \\ & . & . & . \\ P_{n0} & p_{n1} & p_{n2}\ldots\ldots\ldots\ldots\ldots & p_{nn} \end{pmatrix}$$

Such that $\sum_{J=0}^{n} p_{ij} = 1$; i=0, 1, 2, 3…..n and $0 \leq p_{ij} \leq 1$

n-step stationary transition probabilities

The n-step stationary transition probabilities are defined to be

$p_{rs}^{(n)} = P(X_{i+n} = s|X_i = r) = P(X_n = s|X_0 = r)$
$p_{rs}^{(n)} \geq 0$ for all states r and s ; n=1, 2, ……

$$\sum_{s=0}^{n} p_{rs}^{(n)} = 1 \quad \text{for all states r; n=1, 2,….}$$

The above equation assumes there is N+ 1 possible state. Note that if the system is currently in state r, it must be in some state n steps from now. Thus

$$\sum_{s=0}^{n} p_{rs}^{(n)} = 1$$

In general, the n-step stationary transition probabilities can be calculated as follows:

$$p_{rs}^{(n)} = \sum_{j=0}^{n} p_{rj^*}\, p_{js}^{(n-1)}$$

Where the possible states are 1, 2,…..N. That is, the probability of going from state r to state s in n steps is the probability of going from state r to state j in one step, times the probability of going from state j to state s in n-1 steps, summed over all j=0, 1, 2,……, N.

Steady state stationary transition probabilities

Suppose a given system has N+1 state, 0, 1, 2,….., N. if for some value of n,

$p_{rs}^{(n)} > 0$ for r = 0, 1, 2,….., N
s = 0, 1, 2,….., N
and if

$p_{rr} > 0$ for r = 0, 1, 2,….., N

then

$\lim_{n\to\infty} p_{rs}^{(n)} = a_s$ for s = 0, 1, 2,….., N

The quantity a_s is the steady state stationary transition probability of being in state s after a large number of steps. That is to say, if every state can eventually be reached from every other state (possibly in a large number of steps), and if the system can be in any given state on two consecutive steps, then the probability of being in any given state after a large number of steps is a constant. This constant is called the steady state probability for the given state.
The N+1 steady state probabilities satisfy the N+2 linear steady state equation

$$a_s = \sum_{r=0}^{N} a_r * p_{rs} \text{ for } s=0, 1, 2,\ldots.., N$$

$$\sum_{s=0}^{N} a_s = 1$$

Thus, if we form a system of N+1 linear equations in N+ 1 unknown by using any N of the equations in the above equation the solution of the system will be the N+1 steady state probabilities.

Proposed model

The proposed model assumes that by 'maintainability' of the software we mean a quantitative characteristic called 'operating efficiency' , which from user point of view is maximum in the beginning and deteriorates progressively with the passage of time in view of ever increasing user expectations that evolve constantly over time .

The operating efficiency of the software at specific interval of time is computed using Bux Muller transformation .
Software under consideration for maintenance must be in one and only one state of deterioration at specific point of time. The software that is currently in state r must be in some state n steps from now. Under fairly general conditions, if the one-step stationary transition probabilities are available, we can determine. n-step stationary transition probabilities until the software reaches steady state.

Assumptions

- The software to be maintained may be considered in one of the five states of deterioration. Say X_i = { 0, 1, 2, 3, 4 } represents the state of deterioration of the software at the end of i^{th} week.
- The operating efficiency is simulated for each state using Bux Muller transformation. e.g. 95% to 100% for the state=0 and below 70% for state =4 and in-between for other states.
- The one-step stationary transition probabilities may be given or may be determined from the past data.
- n-step transition probabilities are calculated successively until the system reaches steady-state or n = 100 which ever occurs first.
- In the absence of a steady-state a message stating such is printed..

Terms and Notations

N	:	Number of n-step probabilities.
NS	:	Number of states of Deterioration for the software to be maintained.
PROB (X0=I)	:	Probability of being in state I initially (operating efficiency)
P (I, J)	:	One step stationary transition probability
PN (I, J)	:	n steps stationary transition probability
PN (NS, J)	:	steady-state transition probability
MAT (I, J)	:	probabilities of being in state J after I steps.

Algorithm to compute n-step probabilities using Markov analysis

Step 1: Start
Step 2.(a) [Input number of states for software maintenance]
Read NS
(b) [Read the probabilities of being in state I initially]
For I= 1 to NS do
Read PROB {I}
Or
Compute the probabilities (operating efficiency)of each state of deterioration initially operating efficiency using Bux Muller transformation (with the help of random numbers generation, computation of their mean and standard deviation and normalizing the function.)
(c) [Read one step stationary transition probabilities]
For I=1 to NS do
For J = 1 to NS do
Read P (I, J)
Step3. [Calculate n step stationary transition probabilities for n = 1, 2, 3, ….. using the relation]

$$p_{rs}^{(n)} = \sum_{j=0}^{n} p_{rj} * p_{js}^{(n-1)}$$

Step4. [Compute steady state transition probability using the relation]

$$a_s = \sum_{r=0}^{N} a_r * p_{rs} \text{ for } s=0, 1, 2, \ldots, N$$

Step 5. [Compute probabilities of being in state j after I steps.]

Step 6. [Write Results]
Step 7.[Stop]

Input: Read the values of NS and compute operating efficiency say 0.95, 0.87, 0.79, 0.75 and 0.70 as initial state of deterioration
The table 1 acts as input for one step stationary transition probabilities for software maintenance.

To State

From State	0	1	2	3	4
0	0.50	0.45	0.03	0.02	0
1	0	0.56	0.4	0.03	0.01
2	0	0	0.45	0.50	0.05
3	0	0	0	0.60	0.40
4	0	1.0	0	0	0

Table 1

Output:

State	Steady state stationary transition probabilities
0	0
1	0.3173
2	0.2308
3	0.3123
4	0.1396

Table 2

Discussion and conclusion

It is well known that software maintenance claims a large proportion of organizational resources. Poorly developed software creates a 'performance anxiety' in the user's mind on one hand and adds to risk cost in terms of reduced efficiency on the part of the organization that uses it. On the other hand, poor software design on the part of the developers results in loss of man hours of persons using it when the later fails to achieve desired analysis and result out of its use.
Though it is difficult to quantify the actual maintenance efforts at different point of time of our choice, but its impact is fairly realized over the software life cycle. A precise measure of software maintainability can help better manage the maintenance phase effort.
A gradual 'eye' on upkeeps of the software would reveal that with the passage of time the 'operating efficiency' decreases and the level of maintainability effort increases. The initial state of software's operating efficiency proceeds to a state after passing through 'n' steps where the operating efficiency noose dives to the lowest level refers to as 'steady state' after which there will conceptually be no retardation of software efficiency any further and the concerned software may be branded as 'unfit for use' i.e. no further maintainability is desirable and no effort should be made to modify the software. This is achieved after a large number of steps and as such the transition probabilities remain fairly constant for each

state as shown in the table 2. This state is the terminal stage where upon the user has to adapt the strategy of either invest in a new alternate software or go for an improved version of the same.

References

[1] IEEE, IEEE Standard Glossary of Software Engineering Terminology, report IEEE Std 610.12-1990, IEEE, 1990.

[2]Jarrett Rosenberg **"**Can We Measure Maintainability?", Sun Microsystems, 901 San Antonio Road, Palo Alto, CA 94303

[3] Gillite Billy E.,"Operations Research", Tata Mc Graw Hill Publishing Company Limited, 2004

[4] Rikard Land**, "**Measurements of Software Maintainability**"** Mälardalen University, Department of Computer Engineering, Box 883, SE-721 23 Vasteras, Sweden rikard.land@mdh.se

[5] Rajiv D. Banker j Gordon B. Davis j Sandra A." Slaughter Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study"
School of Management, University of Texas at Dallas, Richardson, Texas 75083

[6] Pfleeger S. L., Software Engineering, Theory and Practice, Prentice-Hall, Inc., 1998

[7] Niessink ,F**. "**Perspectives on improving software maintenance", Proceedings of IEEE International Conference on Software Maintenance, pp. 553-556, 2001

[8] Y. Kataoka, T. Imai .H. Andou T. Fukaya "A Quantitative Evaluation of Maintainability Enhancement by Refactoring" 18th IEEE International Conference on Software Maintenance, pp. 0576, 2002

P.K. Suri received his Ph.D. degree from Faculty of Engineering, Kurukshetra University, Kurukshetra, India and Master's degree from Indian Institute of Technology, Roorkee (formerly known as Roorkee University), India. He is working as Professor in the Department of Computer Science & Applications, Kurukshetra University, Kurukshetra - 136119 (Haryana), India since Oct. 1993. He has earlier worked as Reader, Computer Sc. & Applications, at Bhopal University, Bhopal from 1985-90. He has supervised five Ph.D.'s in Computer Science and thirteen students are working under his supervision. He has more than 100 publications in International / National Journals and Conferences. He is recipient of 'THE GEORGE OOMAN MEMORIAL PRIZE' for the year 1991-92 and a RESEARCH AWARD –"The Certificate of Merit – 2000" for the paper entitled ESMD – An Expert System for Medical Diagnosis from INSTITUTION OF ENGINEERS, INDIA. His teaching and research activities include Simulation and Modeling, SQA, Software Reliability, Software testing & Software Engineering processes, Temporal Databases, Ad hoc Networks, Grid Computing , and Biomechanics.

Bharat Bhushan received the M Sc. (Physics), from Panjab Univ. Chandigarh and M.Sc. (Comp. Sc.), MCA degrees from Guru Jambeshwar University. Respectively. Presently working as Head, Department of Computer Science and Applications, Guru Nanak Khalsa College, Yamuna Nagar (affiliated to Kurukshetra University, Kurukshetra- Haryana, India) and senior most teacher of computer science in Haryana since 1984. He is a member of Board of Studies of Computer Science, Kurukshetra University and member of Advisory Board of educational programme (EDUSAT) launched by Govt. of Haryana to impart online education. His research interest includes Software engineering, Digital electronics, networking and Simulation Experiments.