International Advance Journal of Engineering, Science and Management (IAJESM)
ISSN -2393-8048, July-December 2022, Submitted in September 2022, iajesm2014@gmail.com

STUDY ON SOFTWARE MAINTAINABILITY SIMULATOR

Anju, Research Scholar, Department of Computer Science, Monad University, Hapur, Uttar Pradesh (India)
anjupanwar2793@gmail.com

Abstract

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. The simulator is developed in this chapter to compute n-step e steady state stationary transition probabilities for various state of the software under maintenance. The one step transition probabilities for five initial states of deterioration of the software under maintenance. The transition probabilities are chosen according to Markovian property i.e. the sum of the probabilities of going from one state to all other state is one. The operating efficiency of the software is supposed to be 0.95, 0.87, 0.79, 0.75 and 0.70. The steady state transition probabilities for each state denoted by 0,1,2,3 and 4 are shown. This simulator is executed for a maximum value of n=100 or till the system reaches a steady state while calculating n-step probabilities successively.

Keywords: Software, Simulators, Quality, Maintance

Introduction:

Software is developed, maintained, and used by people in a wide variety of situations. Students create software in their classes, enthusiasts become members of open-source development teams, and professionals develop software for diverse business fields from finance to aerospace. All these individual groups will have to address quality problems that arise in the software they are working with. This chapter will provide definitions for terminology and discuss the source of software errors and the choice of different software engineering practices depending on an organization's sector of business. Every profession has a body of knowledge made up of generally accepted principles. In order to obtain more specific knowledge about a profession, one must either: (a) have completed a recognized curriculum or (b) have experience in the domain. For most software engineers, software quality knowledge and expertise is acquired in a hands-on fashion in various organizations. The Guide to the Software Engineering Body of Knowledge constitutes the first international consensus developed on the fundamental knowledge required by all software engineers.

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 [IEE1990]. 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 [JAR1990].

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 state 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. Here an attempt has been made to develop a simulator to compute n–step transition probabilities successfully for software under maintenance until it reaches steady state. This process is very much depicted by Markov analysis [GIL2004].

The purpose of software maintenance is to assure the quality of performance of the respective software. But design errors, undiscovered faults and installing new applications can cause the software degradation [RIK1999]. There are two aspects of maintainability: serviceability (the probability of returning the item to normal service) and repair ability (the probability of repairing the actual or impending fault). Generally, software maintainability is

termed as repair ability. 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. et al. [RAJ1994] 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 [PFL1998] describes maintainability as the probability that a maintenance activity can be carried out within a stated time interval, it ranges from 0 to Rikard Land [RIK1999] investigates how the maintainability of a piece of software changes as time passes and it is being maintained by performing measurements on industrial systems. Niessink F. [NIE2001] discussed the perspectives of improving software maintenance and described software maintenance process improvement from two perspectives: measurement-based improvement and maturity-based improvement.

Y. Kataoka et al. [YKA2002] discussed program refactoring as a technique to enhance the maintainability of a program. A quantitative method was proposed to measure the maintainability enhancement effect of program refactoring. Coupling metrics were used to evaluate the refactoring effect. By comparing the coupling before and after the refactoring, the degree of maintainability enhancement was evaluated. The results showed that the 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 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 $t_0 < t_1 1 < t_2 <\ldots\ldots\ldots< t_n$ 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 :

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

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

Markov process is a sequence of 'n' experiments in which each experiments has 'n' possible outcomes x_1, x_2,……,xn. 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.

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

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

Mathematically, the probability 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 with elements $\mathbf{p_{ij}}$

Such that $\sum_{j=0} p_{ij} = 1$; 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

$$p_{rs}^{(n)} = P(X_{i+n} = s | X_i = r) = P(X_n = s | X_0 = r)$$

$$p_{rs}^{(n)} \geq 0 \text{ for all states r and s; } n=1, 2,\ldots$$

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

The above equation assumes that there are N+ 1 possible states. 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 states, 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 equations

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

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

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

PROPOSED MODEL

The proposed model assumes that 'maintainability' of the software means 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.

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, one can determine n-step stationary transition probabilities until the software reaches steady state.

The simulator developed in this chapter computes the n-step probabilities successively until the system reaches steady state or until n = 100, which ever occurs first. If steady state is not reached, a message stating such is printed. The simulator is developed using high level programming language.

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.

DESCRIPTION OF ALGORITHM: SIM_SOFT_MAINT

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
MAT (I, J)	: Probabilities of being in state J after I steps.

Algorithm SIM_SOFT_MAINT for n-step probabilities using Markov Analysis

1. [INPUT]

(a) [Number of states for software maintenance]

Read NS

(b) [Probabilities of being in state I initially]

[Compute the probabilities (operating efficiency) of each state of deterioration initially operating efficiency using Box-Muller transformation by (with the help of random numbers generation), computing of their mean and standard deviation and normalizing the function These probabilities are denoted by PROB(I)), I=1 to NS] or

For I= 1 to NS

Read PROB (I)

End For

(c) [One step stationary transition probabilities]

```
    For I=1 to NS
      For J = 1 to NS
        Read P (I, J)
      End for
    End for
```

2. [Calculate n step stationary transition probabilities for N = 1, 2, 3, …..]

```
    For R =1  to NS
      For S = 1 to NS
        PN[R, S) = 0
        For J= 1 to  NP
        PN (R, S)=PN (R,S)+P(R,J)*P (J,S)
       End  for
      End for
    End for
```

3. [Compute steady state transition probability]

```
    For  J=1 to NS
      TEMP(J)=0
     For I=1 to NS
       TEMP (J)=TEMP(J)+PROB(I)*PN (I,J)
     End for
    End for
```

4. [Write probabilities of being in state j after i steps in the form of matrix MAT using TEMP (J)]
5. [write results]

```
   For I=1 toNS
     For J= 1 to NS
       Write MAT(I,J)
     End for
    End for
```

6. Stop

RESULTS & DISCUSSION

The simulator is developed in this chapter to compute n-step e steady state stationary transition probabilities for various state of the software under maintenance. The one step transition probabilities for five initial states of deterioration of the software under maintenance have been shown in table 1. The transition probabilities are chosen according to Markovian property i.e. the sum of the probabilities of going from one state to all other state is one.

The operating efficiency of the software is supposed to be 0.95, 0.87, 0.79, 0.75 and 0.70. The steady state transition probabilities for each state denoted by 0,1,2,3 and 4 are shown in the table 2 in the form of results.

This simulator is executed for a maximum value of n=100 or till the system reaches a steady state while calculating n-step probabilities successively.

TABLE 1: Transition Probabilities Matrix

From State	0	1	2	3	4
0	0.55	0.40	0.03	0.02	0
1	0	0.50	0.46	0.03	0.01
2	0	0	0.44	0.50	0.06
3	0	0	0	0.68	0.32
4	0	1.0	0	0	0

TABLE 2: Steady State Transition Probabilities

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

CONCLUSION:

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 increase. 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 referring 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 16. This state is the terminal stage where the user has to adapt the strategy of either invests in new alternate software or goes for an improved version of the same. The software simulation tool designed here will be helpful for the software project managers in judging the maintenance efforts of the software. 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.

Reference:

Aannestad, B., Hooper, J., "The Future of Groupware in the Interactive Workplace", HRMagazine, Vol. 12, Issue 11, November 1997, pp. 37-41.

Abdrabou A, Zhuang W (2006) A position-based QoS routing scheme for UWB mobile ad hoc networks. IEEE J. Select. Areas Commun. 24:850-856.

Agarwal, H., Demillo, R. A. and Spafford, E.H. Debugging with Dynamic Slicing and Backtracking, Software Practice and Experience, 23, pp. 589-616, 1993.

Campos, J., Arcuri, A., Fraser, G. and Abreu, R. Continuous Test Generation: Enhancing Continuous Integration with Automated Test Generation, In the Proceedings of Automated Software Engineering (ASE), 2014.

Camuffo, M., Maiocchi, M. & Morselli, M., 1990. Automatic software test generation. Inform. Softw. Technol., pp. 337-346. Carnes, P., 1997. Software reliability in weapon systems. , Proceedings of 8th International Symposium On Software Reliability Engineering, p. 114–115.

Canfora, G., Cimitile, A. and De Lucia, A. Conditioned Program Slicing. Information and Software Technology, 40(11), pp. 595–607, 1998.

Cao, Y., Hu, C. and Li, L. An Approach to Generate Software Test Data for a Specific Path Automatically with Genetic Algorithm, In the Proceedings of ICRMS, Chengdu, pp. 888-892, 2009.

Dufner, D., Kwon, O., Hadidi, R., "WEB-CCAT: A Collaborative Learning Environment For Geographically Distributed Information Technology Students and Working Professionals", Communications of the Association for Information Systems, Vol. 1, Article 12, March 1999, Available [Online]: http://cais.isworld.org/articles/1-12/article.htm [26 November 2000].

Edvardsson, J and Kamkar, M. Analysis of the Constraint Solver in UNA Based Test Data Generation, In the Proceedings of the 9th European software engineering conference

held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, 26(5), pp. 237-245, 2003.
Ehrlich, W. K., Lee, K. & Molisani, R. H., 1990. Applying reliability measurement: A case Study. IEEE Transaction on Software, p. 56–64..
Udell, J., Asthagiri, N., Tuvell, W., Peer-To-Peer: Harnessing the Power of Disruptive Technologies, O'Reill & Associates, 2001.
UNCTAD,World Economy Report(2012)The Software Industry and Developing Country PP-38-42 Review of Literature - II Economic Analysis of Changing Dimensions of IT Sector in India Page 74.
Upadhya, Carol (2007). „Employment, Exclusion & „Merit" in the Indian IT Industry", Economic & Political Weekly, A Sameeksha Trust Publication alsoseehttp://www.epw.org.in ,VolXLI No.36 September9- 15,2006 PP-1863- 1867.
Vaishnav, Rajiv (2011)." Indian Industry2011: Key Driver of growth", The Hindu Survey of Indian Industry, REGD, RN/5734|61 pp.190-192.
Varma, Shweeta (2012). „Looking for that Sunshine", Dataquest, Vol.xxxNo.16 & 17August31-September15, 2012, PP- 104-108.
Vivek V, Sandeep T, Manoj B S, Murthy C S R (2004) A novel out-of-band signaling mechanism for enhanced real time support in tactical ad hoc wireless networks. Proc. IEEE RTAS 56-63.
Wallace, D. & Coleman, C., 2001. Application and Improvement of Software Reliability Models, NASA, Goddard Space Flight Centre(GSFC): Technical Report,Software Assurance Technology Center.
Wang M, Kuo G S (2005) An application-aware QoS routing scheme with improved stability for multimedia applications in mobile ad hoc networks. Proc. IEEE Vehicular Technology Conf. 1901-1905.
Wang, Z. & Wang, J., 2005. Parameter estimation of some NHPP software reliability models with changepoint. Communications in Statistics: Simulation and Computation, Volume 34, p. 121–134..
Wang, Z., Wang, J. & Liang, X., 2007. Non-parametric Estimation for NHPP Software Reliability Models. Journal of Applied Statistics, pp. 107-119.
Whittaker, J. A., 2000. What is software testing? And why is it so hard?. Software, pp. 70-79.
Wood, A., 1996. Predicting software reliability. IEEE Computer, Volume 11, pp. 69 - 77.
Wilson, J., Hoskin, N., Nosek, J., "The Benefits of Collaboration for Student Programmers", 24th SIGCSE technical symposium on Computer Science Education, February 1993, pp. 160-164.