Simulation Based Effort Optimization for Software Testing

Er. Shweta Sharma[1], Mr.Ravi Jaiswal[2], Dr.Rajesh Garg[3]
[1]M.Tech Scholar, [2]Assistant Proffesor, [3]Lecturer
[1, 2] Ganpati Institute of Technology & Management,
[3]Seth Jai Parkash Polytechnic Damla, Yamuna Nagar-135001, Haryana (INDIA)

Abstract- This paper describes the use of simulation concept in testing of software modules in a software project. Simulation is used to observe the dynamic behavior of model of a real or imaginary system. Indeed, by simulating the complex system we are able to understand its behavior at low cost. The Program Evaluation and Review Technique is a network model that allows for randomness in weights of software modules. In this paper, activities and nodes for preparing network diagram are taken for software and then simulation is applied to identify the critical and near critical activities, so that testing process is optimized and less efforts for software testing.

Keywords- *Modules, Criticality Index, Simulation, Testing, Software.*

I. INTRODUCTION

Testing is an essential activity in software engineering. Software Testing is a necessary part of the development or implementation of any new software installation or up gradation. The goal of testing is to find errors and the good test case is the one that has maximum probability of finding errors. Testing amounts to observing the execution of a software system to validate whether it behaves as intended and identify potential malfunctions [1]. Testing is the only phase that consumes most of the time and efforts in Software Development Life Cycle. To test and evaluate a system, testers need to be very experienced and dedicated. So a technique to reduce time, cost and efforts is essential to apply and it is Simulation. Simulation is the representation of a real life system by another system, which depicts the important characteristics of the real system and allows experimentation on it. Simulation is used to observe the dynamic behavior of model of a real or imaginary system. This paper presents how the modules are represented through the network diagram, and various events are interconnected through each other. In this way most optimistic, pessimistic and likely time for every module is analyzed for calculation of critical modules in project. Systematic arrangement of all modules and applying specific distribution, we identify the critical and near critical modules in software that need to be paid more attention. Simulation very powerful technique helps in this direction. This makes testing software modules very easy and efficient. But otherwise, lot of time and efforts of software engineers are consumed in testing of complex projects. They are written in programming language, such as C++ or Java. Simulation allow the developer to capture basic algorithmic functionality at the same time as they focus attention on topology, timing, criticality of modules , overall scalability and other properties characteristics of distribution [7].

II. NETWORK REPRESENTATION FOR TESTING OF MODULES IN SOFTWARE PROJECT

The PERT is used here for simulating software testing process. For this, a network diagram is being prepared in which activity is a module and nodes are the events after completion of one or more activities. The network having loops or cycles is always reduced to a cyclic graph and that is easily interconnected through various nodes.[2] In order to simulate the module network, one has to avoid loops; otherwise it is not possible to achieve desired results. In this paper, there are thirty modules in software project that are considered here for testing of modules. In all there are sixteen nodes/ events which are milestones for the software to be simulated. These nodes are interconnected through different modules in network diagram and each module has its weight depending upon their use. The weights are estimated randomly and generated using beta distribution. It has been assumed that weight assigned to each activity follow a Uniform distribution. A series of random weights is generated using Box Muller transformation [3].

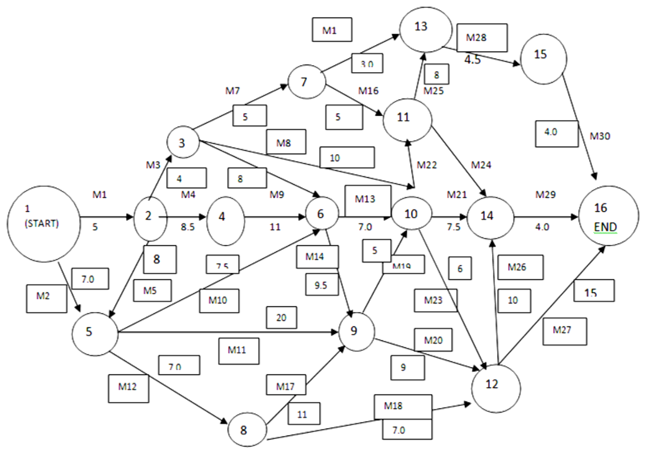

Fig.1. Network representation of software modules for finding critical modules in software.

The developed simulator identifies the critical and near critical activities. The description of modules is given as follows: - Source module pertains module M1 having weight 5. Sink (ending) module is M29 having weight 4.0. Similarly, there are other modules like M3, M4, M5 …………M30 with varying weights. Ending modules pertains M12, M14, M15, and M16 with weight 15, 10, 4, 4 respectively. Table 1 shows the interconnection of activities and nodes. For example, module M2 has Starting Node 1 and Finishing Node 5. [4][7] Similarly activity M25 has Starting Node 11 and Finishing Node 13 so on.

TABLE 1. START AND FINISH FOR MODULES OF FIGURE 1.

MODULES (I)	S[I]	F[I]
1 (M1)	1	2
2 (M2)	1	5
3 (M3)	2	3
4 (M4)	2	4
5 (M5)	2	5
6 (M6)	3	6
7 (M7)	3	7
8 (M8)	3	10
9 (M9)	4	6
10 (M10)	5	6
11 (M11)	5	9
12 (M12)	5	8
13 (M13)	6	10
14 (M14)	6	9
15 (M15)	7	13
16 (M16)	7	11
17 (M17)	8	9
18 (M18)	8	12
19 (M19)	9	10
20 (M20)	9	12
21 (M21)	10	14
22 (M22)	10	11
23 (M23)	11	13
24 (M24)	11	14
25 (M25)	12	10
26 (M26)	12	14
27 (M27)	12	16
28 (M28)	13	15
29 (M29)	14	16
30 (M30)	15	16

III. ALGORITHM

```
Algorithm 1: TEST_SIM (N, M, I ,S[I], F[I], MUE[I], SIGMA[I])
Step-1: Read N, M.
Step-2: Read S[I], F[I], MUE[I] and SIGMA[I] for I = 1,2...N.
Step-3: Set ERROR←0.001, LRUN←1000, FREQ [I] ←0, CRIT [I] ←0 for I= 1, N
        Set RUN←1
Step-4: Repeat for I←1 to N
        Generate weight samples W [I].
        [End of loop.]
Step-5: Perform forward pass.
[EFW(i)= ESW(i) + W(i) ∀ i = 1,2...N

ENW(j)= max{EFW[all activities terminating in j]}

                              ∀ j = 1,2...M

ESW(i)= ENW(S(i))        ∀ i = 1,2...N]

Step-6: Traverse the network for backward pass

        [LSW(i)= LFW(i)- W(i)  ∀ i = 1,2...N

        LNW(j)= min{LSW[all activities originating in j]}
                              ∀ j = 1,2...M
        LFW[every activity terminating in node j]=LNW(j)
                              ∀ j = 1,2...M]

Step-7: Mark the critical activities and update freq [i].
        [IF LSW(i)- ESW(i)<= ERROR Then mark i as critical activity and
        Update Freq[i] = Freq[i] +1]

Step-8: Set RUN←RUN + 1                    [Update]

Step-9: IF RUN <= LRUN, THEN
            GoTo step (4)
        ELSE
        Calculate CRITICAL_INDEX for all activities and show results
        (End of IF block)

Step-10: Exit.
```

IV. TOOL AND PLATFORM USED FOR SIMULATION

This simulator is designed using C++ language under Windows operating system on an Intel core i5 Compatible machine. C++ is real time simulation language that has its many applications in real world problems. The model explained in this research paper is stochastic and dynamic in nature. The next –event discrete simulation model has been used for conducting simulation experiment. Monte Carlo Simulation is implemented on the system to get good results and outputs.

TABLE 2.INPUT DATA FOR SIMULATOR

MODULES (I)	S[I]	F[I]	A[I]	M[I]	B[I]	MUE[I]	SIGMA [I]
1 (M1)	1	2	1.68	5.08	8	5.0	1.22
2 (M2)	1	5	1	9.25	4	7.0	0.50
3 (M3)	2	3	0.3	4.42	6	4.0	0.95
4 (M4)	2	4	0.7	9.32	13	8.5	2.05
5 (M5)	2	5	0.10	9.475	10	8.0	1.65
6 (M6)	3	6	0.90	10.75	4.0	8.0	0.75
7 (M7)	3	7	2	5.0	8.0	5.0	1.00
8 (M8)	3	10	0.8	11.30	14.0	10.0	2.20
9 (M9)	4	6	0.10	11.725	19	11.0	3.15
10 (M10)	5	6	0.10	9.475	7.0	7.5	1.15
11 (M11)	5	9	0.88	7.78	10.0	20.0	4.20
12 (M12)	5	8	0.80	23.30	26	7.0	1.52
13 (M13)	6	10	2.0	8.75	20	7.0	0.25
14 (M14)	6	9	0.50	9.875	2.0	9.5	3.00
15 (M15)	7	13	0.60	5.85	6.0	3.0	0.20
16 (M16)	7	11	0.80	3.55	3.0	5.0	0.90
17 (M17)	8	9	0.90	12.275	16.0	11.0	2.75
18 (M18)	8	12	0.90	6.525	15	7.0	2.35
19 (M19)	9	10	0.90	4.775	10.0	5.0	1.75
20 (M20)	9	12	2	8	20	9.0	3.00
21 (M21)	10	14	0.02	6.245	11	7.5	2.25
22 (M22)	10	11	0.10	7.225	7	6.0	1.83
23 (M23)	10	12	0.50	7.775	14	6.0	1.15
24 (M24)	11	14	0.40	7.90	16	8.0	2.25
25 (M25)	11	13	0.50	8.375	14	8.0	2.60
26 (M26)	12	14	0.90	10.775	16	10.0	2.75
27 (M27)	12	16	0.72	17.07	21	15.0	3.38
28 (M28)	13	15	1	4.75	7	4.5	1.00
29 (M29)	14	16	1	3.25	10	4.0	1.50
30 (M30)	15	16	0.7	3.3	10	4.0	1.55

Where, A [I] = optimistic estimate for each module, M [I] = most likely duration for each module, B [I] = most pessimistic estimate for each module.

MUE [I] = (A [I] +4 * M [I] +B [I])/6

SIGMA [I] = ((B [I] - A [I])/6)

The simulator is provided with following fixed input: ACT=30 (Number of modules in software), NODE=16 (Number of events in the particular software project), LRUN=1000 (Number of simulation runs), ERROR= .001. Average (MUE) and variance (SIGMA) for weights of modules are shown in Table 2. User can enter values of ACT, NODE, ERROR and LRUN in the designed simulator. If the parameter ERROR is changed the results are also changed. Criticality indices of activities are the outputs of the simulator and given in the Table 3. First column in the table 3 is the modules and other is the critical index of respective activity. Criticality index is the measure of number of times the corresponding module constitutes a part the critical path out of LRUN times. The value of critical index is 1 for module M1 and near critical modules having value less than 1 are M2 (0.975), M4(0.925), M6(0.91) , M11(0.811), M13(0.95). The values of modules M13 is 0.001, M5(0.606), M9(0.525), M10(0.57), M12(0.416), M14(0.007), M17 (0.211), M20(0.25), M21(0.53), M22 (0.457), M24 (0.499), M25(0.669), M26(0.285), M27(0.41), M28(0.499), M29(0.336), M30(0.499).

V. DISCUSSION ON RESULTS

Hence, the final output is represented in form of bar graph on which x-axis represents the number of modules whereas y-axis shows the criticality index of modules in the testing software. Criticality indices of activities are the outputs of the simulator and given in the Table 3.

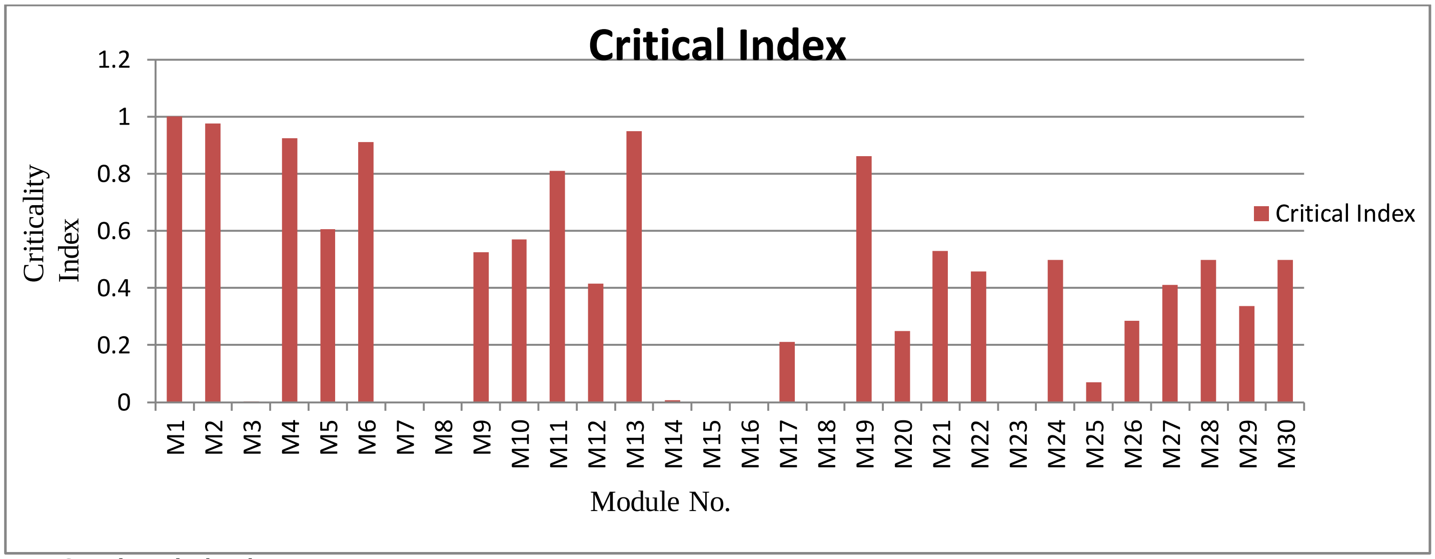

Fig.2: Criticality Index bar chart

On analyzing the output is observed that criticality indices of modules M1, M2, M4, M5, M6, M10, M11, and M13 are more than other modules. It means that these modules should be paid more emphasis and are more error prone as compare to the other ones. The module numbered M1, M2, M4, M5, M6, M10, M11, and M13 should be scheduled more carefully as compared to other modules. Hence it saves lot of time and efforts. Otherwise we have to test all 30 modules in this complex software network, but with the Program Evaluation and Review Technique (PERT) our efforts are really optimized.

TABLE3. CRITICALITY INDEX TABLE

Modules	Critical Modules
M1	1
M2	0.975
M3	0.001
M4	0.925
M5	0.606
M6	0.91
M7	0
M8	0
M9	0.525
M10	0.57
M11	0.811
M12	0.416
M13	0.95
M14	0.007
M15	0
M16	0
M17	0.211
M18	0
M19	0.861
M20	0.25
M21	0.53
M22	0.457
M23	0
M24	0.499
M25	0.069
M26	0.285
M27	0.41
M28	0.499
M29	0.336
M30	0.499

VI. CONCLUSION

Critical modules in software are as obtained as output from simulator. Any failure in them will result in failure of software project. Experts must be employed for scheduling critical modules. In this way, the designed simulator will help in finding error prone modules in the complete software without practical implementation**.** Rather it helps in finding the modules or activities in the software which have more probability of error due to which testing phase consumes about 60 percent of total Software Development Life Cycle (SDLC). Hence, the overall testing cost and efforts are minimized drastically which is the objective of this research. This simulator will be helpful for further testing, development and debugging of complex as well as large software systems.

VII. REFERENCES

[1] Baci, O., “Verification, Validation and Testing”, in Handbook of Simulation, Jerry Banks, ed., John Wiley, New York, 1998

[2] Beizer,B.,“SoftwareTestingTechniques,” 2nd ed., Van Nostrand Reinhold, New York, 1990.

[3] Banks, J., Carson, J.S. and Nelson, B.L., “Discrete-Event System Simulation”, 2nd ed., Prentice-Hall, Upper Saddle River, New Jersey, 1996.

[4] Law, Averill M., “Simulation modeling and Analysis”, 4th ed., McGraw-Hill, New York, 2008.

[5] D.S. Hira, “System Simulation”, 1st ed. S. Chand and Company Ltd, New Delhi, 2001.

[6] Narsingh Deo, “System Simulation with Digital Computer”, Prentice Hall of India, Eighth Edition, 2008.

[7] Matthew J. Rutherford, Antonio Carzaniga, and Alexender L.Wolf, “Simulation-Based Testing of distributed System” University of Colorado, Department of computer science, technical Report CU-CS-1004-06 January 2006.

[8] Matthew J. Rutherford, Antonio Carzaniga, and Alexender L.Wolf, “Simulation-Based Testing of distributed System” University of Colorado, Department of computer science, technical Report CU-CS-1004-06 January 2006.