Simulator for process reliability with reuse of component in time bound software projects

Ritu Soni

Department of Computer Science and Application, Guru Nanak Girls College, YUMUNA NAGAR, HARYANA, 135001, (INDIA)
Ph : 01732-309869(R), 9355370656 (m ) ritusoniynr@hotmail.com, ritusoniynr@yahoo.com

Summary. This work describes the use of process simulator for evaluating various reliability parameters for the time bound component based software projects. A software engineering process consists of number of components that are processed sequentially or in parallel and there by representing the SEP model in the form of network consisting of components connected in series and parallel. Components can be constructed from starch or can be reused. Human reliability also affects SDLC. The design phase in these component-based event driven software require special consideration. So, process reliability of the software depends on the reliability of these individual components, human reliability of each component and their structured usage in the process of development. The process simulation is a part of decision support system for assisting the project managers in planning or tailoring the software development process in time bound quality driven method.

Key words: Software, Component, Reliability, Process simulation, Time bound, SEP model

1 Introduction:

Quality and timely delivery of software within reasonable costs are critical for the success of many business organizations today. Quality software performs its intended functions correctly and reliably and can be quantitatively expressed in terms of software reliability. Quality could be achieved if the process of software development and the software product is reliable. A common measurement unit of reliability is the number of failures in a time interval. Failures are due to faults in the software, which are introduced by human errors. Human beings make errors either due to their behavior or due to lack of knowledge. Thus we can say that human personality factors has a major contribution in the introduction of errors , not only at various stages of SDLC but also while using the software. Thus system reliability can never be considered in isolation from human factors. Keeping this in mind the system reliability has to be redefined as

System Reliability = Hardware Reliability + Software reliability + Human Reliability
Human reliability can be predicted for software development process as well as software product for use. Since in this paper we are concentrating on the process reliability, so human reliability is also calculated with respect to development process. The human factor can be taken into account at two levels: Normal operations and Emergency operations. In both cases, human reliability is concerned with the understanding of ‘human error’ mechanisms in order to model it. Thus, human reliability focuses on the human factors that may affect the quality of the ultimate product under development and the ways of improving these errors.

Research is being done in the field of HRA like nuclear power plant [?], [?], [?], railroad, positive train control system (PTC) systems, OECD-NEA [?].

THERP is the best-known, most frequently applied technique for human performance reliability prediction. A key aspect of THERP is the determination of the probability that an error or class of errors will result in a system or process failure. Mathematically it is given as

$$Q_i = 1 - (1 - F_i P_i)^{n_i}$$

where $n_i$ is the number of independent operations. The total system or subsystem failure rate is given by:

$$Q_T = 1 - \left[ \Pi_i^{n_i} (1 - Q_k) \right]$$

where $Q_T$ is the probability that one or more failure conditions will result from errors in at least one of the $n$ failure classes.

Process modeling is a method for characterizing the structure of a process. A software engineering process consists of number of components which are processed sequentially or in parallel (depending on processing environment/ platform) and there by representing the SEP model in the form of network consisting of components connected in series and parallel. These components may be developed from starch or can be reused. So, overall reliability of the software depends on the component reliability & human reliability of these individual components and their structured usage in the process of development.

Developing a model of the software development process involves the identification of entities, attributes, interactions, operations and human behavior that are present in that process. Software process model execution can be accomplished by developing appropriate simulators. When the simulator is executed, the dynamics of the model cause attribute values to change over time, throughout the execution period. But the human factors are fixed as team for software development is fixed. The predictions about human reliability help the manager to select appropriate software development team members. Quality of the process depends directly upon the reliability of the process, efficient usage of resources and timely completion of each process. Reliability of any process depends on errors occurrence and rectification of errors. Error fixation is done by software engineer(s), whose scheduling can be simulated, in accordance to the organization policies, so as to complete the development process in time. Both these attributes leads to the third, most important attribute i.e. cost of software product. The attributes (such as cost, schedule and quality, etc.) can be presented graphically so that a Project Manager developer can draw inferences about the performance of the simulator relative to the real live project Rus [?].

All software development process consists of development of five development phases e.g. SRA, Design, coding, Testing & implementation. Developed software
will be reliable if the software engineering process is reliable. In component based software development, some of the pre-developed components can be reused, whose reliability is already estimated. As now a days mostly all the software are development in event driven environment, so special consideration is needed for the design phase. Each of the step in design process (viz. design of user interface, program procedure needed to process data, attach the procedure to program events which will trigger their execution and program instructions [?] is treated as event. The aim of this work is to evaluate the reliability parameters for the system, which are involved in the design of event driven system. Here an attempt has been made to design and develop a rate-based simulator for an event driven application, which has been developed for four steps in design process and five development steps. These design sub phases are treated as events. Since each development or design sub phase is independent of other, some bugs (which are not carried forward and which do not cause failures) can be left as such. These bugs are called as residual bugs. The reliability of any phase depends upon the occurrence pattern of the bugs during different development or design sub phases, which leads to the accumulation of bugs to be fixed by a software engineer(s). If the total time required for fixing bugs in any development or design sub phase is greater than the stipulated time, then an additional software engineer is hired and his services are utilized.

The current simulator developed herein C++ is useful to project managers in planning or tailoring the software development process (in order to produce time bound quality product) and, to compute software reliability parameters like failure density, MTTF, and overall reliability of Software. It also determine the performance of the software engineer(s) involved in the development of SEP model and thereby predicting the development cost of the Software within given time.

2 Boundaries for Proposed Simulator:

A rate-based simulator for evaluating the reliability parameters for software development process, with re-use of component in the time bound event driven software projects, with the following distinctiveness:

I There are m components working in series or parallel.
II The reliability of reused components is pre defined.
III For each component, software development team is fixed. Thus the human reliability for that component is also fixed.
IV There are in all five-development phases in any process.
V But the design phase has four sub phases.
VI All the design sub phases are carried out in series in event driven software development.
VII Total number of bugs occurred in each development phase or design sub phase follow Halstead's software metric model.
VIII Inter-occurrence of bugs and time taken for fixing bugs, in each of the design sub phase, follow a Poisson distribution and negative exponential distribution, respectively.
IX The time required for fixing the bugs in the overall design phase follow 4-\beta Erlang Distribution, where \( \beta \) is the average time required to fix bugs in each of the sub-phase (which is derived by generating pseudo-random value from a negative exponential distribution)
The bugs are fixed by multiple numbers of software engineers having their own separate queue.

Every software engineer is capable of debugging any kind of bug independently.

At least 85% of bugs must be fixed using first come first serve discipline and residual bug fixation is not carried to the next development or design sub phase.

Stipulated time for any one of the development or design sub phase is assumed as 2.5 times the number of bugs occurred (Ref. Discussion held with software consultants of repute).

If software engineer is free he shall fix the bug occurring during the process of designing a sub phase. But if busy the bug is kept pending. Hence there is delay in debugging there by increasing the total number of bugs.

When a software engineer becomes free before the occurrence of next bug, idle time of the software engineer while waiting for the allocation of next debugging job, is recorded.

If the time taken to fix the pending bugs in any of the sub phase exceeds specified development time (equal to manpower of the engaged software engineer), then additional software engineer(s) will be provided instantaneously. His services will be terminated as soon as debugging is completed. The services of any software engineer are

Reliability of each design sub phase, for any component, is the product of reliability of all the four design sub phases.

Reliability of each component is the product of each development phase.

Overall reliability of any component is the combination of component reliability and human reliability.

Process reliability of the software depends on the satisfactory execution of any one of the m components depending on their usage in series and/or parallel (that is - mixed configuration).

3 Notations:

i........current serial number of bug
k..........maximum number of software engineers/professionals
m..........number of components
β........the average time required to fix bugs in each of the sub-phase
at(i).......time gap between occurrence of the (i-1)th bug and ith bug
cat(i).....Cumulative occurrence time of the ith bug
s(i,k).....time taken for fixing the ith bug by the kth software engineer
cdt(i,k)....cumulative fixation time of ith bug by the kth software engineer
wt(i,k)....time gap between the occurrence of the ith bug and action initiated by kth software professional fixing the same
ql(k)......total number of bugs immediately after the occurrence of ith bug (to be fixed by kth s/w engineer)
idt(i,k)...idle time of the professional while waiting for the allocation of debugging job in respect to the ith bug
fe(j)......number of failures for jth interval
T............total time for design sub phase
t...........simulation period for each design sub phase(s) in respect to removal of pending bugs
ef(j) . . number of residual bugs during j\textsuperscript{th} design sub phase
efd(j) . . . failure density during j\textsuperscript{th} design sub phase
er(j) . . . reliability during j\textsuperscript{th} design sub phase
esfd . . . . sum of failure density
emttf . . . . mean time to failure
ce(j) . . number of fixations during j\textsuperscript{th} design sub phase
s(j) . . . development time for j\textsuperscript{th} design sub phase
r(m) . . reliability of m\textsuperscript{th} component
F1 . . . Probability of process/system failure
P_i . . Probability that error will occur
Q_i . . Probability that class of errors will lead to system failure
n_i . . number of independent operations
Q_T . . probability that one or more failure conditions will result from errors in at least one of the and failure cases.
Hr(m) . . . Human reliability for m\textsuperscript{th} component

4 Simulation of Rel_Time_Des_Comp_SW

In this section, a simulator program has been developed for Rel_Time_Des_Comp_SW. When this program is run on a computer it will evaluate reliability parameters of the time bound component based SRM defined above for design phase of SDLC.

Algorithm:Rel_Time_Des_Comp_SW

1 [Read number of components and total number of depth levels ]
read (m,l)
2 [Within each level read the component number processed in series and in parallel]
for (z=1 to l)
read(cs(z),cp(z))
3 [reliability algorithm for each component]
for (l=1 to m)
begin
3.1 [Human reliability and reusability factor for each component] read (Hr(l), cr(l))
3.2 if cr(l) = 1 [component is being reused so just read the predefined reliability of this component]
begin
read reli(l)
goto 3
end;
3.3 for i= 1 to 5 if i ≠ 2 [Calculate reliability and statistical parameters for each development phase except design phase] call reli()
else [Calculate the reliability and statistical parameters for each design sub phase] begin
for j=1 to 4
call reli()  
call compreli()  
end;

4 [Calculate component reliability of the components which is the product of each phase]
   call compreli()

5 [Calculate overall reliability as a function of component reliability and human reliability which are mutually exclusive to each other]
   call oreli()

6 [Calculate process reliability as a function of network of components working either in series or in parallel ]
   call proreli()

**Algorithm: compreli()**

[This algorithm determines various reliability parameters and statistical parameters]

begin
a) [read stipulated development time for each phase]
   read (s[i])
b) [Generate random number for failure occurrence of i<sup>th</sup> interval]
   fe[i] ← ran1()
c) [read stipulated development time for each phase]
   s[i] ← 2.5*fe[i]
d) [calculate cumulative failure occurrence]
   cf[i] ← cf[i-1]+fe[i]
e) [Generate random number for bug fixation of i<sup>th</sup> interval]
   ce[i] ← erlang ()
f) [calculate cumulative bug fixation]
   tcce[i] ← tcce[i-1]+ce[i]
g) [For each development phase, generate Poisson variates for inter-occurrences of bugs and generate time samples for fixing them using negative exponential distribution]
   call rread(ce[i],i)
h) [Assign the job of debugging to the first software engineer (if available), else hire an additional software engineer on work basis. Evaluate parameters such as pending number of bugs, idle time of the software engineer, delay in fixing of bugs]
   call service(ce[i],i)
i) [calculate and print various reliability parameters]
   call stat()
end;

5 Implementation

Number of runs = 1000

**Sample Input Data:**

1. Number of components: m = 3
2. Number of levels: \( l = 2 \)
3. Number of development phases = 5
4. Number of design sub phases = 4
5. Length of time quantum for each design sub phase: \( t = 1 \)

**Sample Output Data:**

- **Table 1**: Component reliability
- **Table 2**: Reliability parameters such as Failure density and reliability of the each development and design sub phase of Software.
- **Table 3**: The detailed output of parameters in respect to design sub phase-I i.e. for example, evaluation of parameters such as idt, wt and ql.
- **Graph 1, graph 3, graph 5, graph 7**: Scattered graph for overall reliability in 1000 simulation runs.
- **Graph 2, graph 4, graph 6, graph 8**: Histogram for overall reliability frequency with reliability range varying from minimum as 0.1 to maximum as 1.0, with width as 0.1.

**Reliability of component c1:**

\[
R = \prod_{j} r(j) = 0.57744858
\]

**Validation Checks:**

- In the any phase, number of bugs fixed should not be greater than number of bugs occurred.
- If all the bugs are not corrected in the any sub phase, then it is not being debugged at all.
- On occurrence of bug, the status of software engineer is checked if either he is free or total time taken for fixing the pending bugs exceeds the specified development time for design sub phase.
- If software engineer is free, bug is assigned to him for fixation process.
- If the time taken to debug the pending bugs exceeds specified development time for \( j^{th} \) design sub phase, then the 2\( ^{nd} \) software engineer will be provided to the system.
- The service of second software engineer is terminated as soon as he becomes idle.
- The reliability of reused component is not calculated again and again.

**6 Discussion:**

Let us illustrate the above simulator with the help of example, in which we have assumed the software consists of only three components i.e. \( c_1, c_2 \) and \( c_3 \) (Table 1) where \( c_2 \) and \( c_3 \) are parallel while \( c_1 \) is in series with them. Bugs occur in any phases (intervals) of equal length, which are being debugged by a single software engineer (initially). Stipulated time period for each development or design sub phase is calculated. The reliability and statistical parameters for each development or design sub phase is calculated. The human reliability for each component is inputted. In this particular simulator, there are five development phases, but it due to event driven property of software, it design phase has four sub phases. Let’s elaborates on the design phase of \( c_1 \) component. The stipulated time period for each design
Fig. 1.

sub phase is (25, 47.5, 32.5 and 12.5) respectively. The numbers of bugs that have occurred during these four intervals were generated as 10, 19, 13 and 5 respectively, by using pseudo random number generator. The numbers of bugs rectified (using Erlang distribution for generating pseudo random number) are 9, 16, 12 and 5 in the above said four intervals, respectively. The 1st, 2nd and 3rd column gives the values of fe(j)’s, - the number of failures, ce(j)’s- the number of bugs corrected and residual bugs, respectively, in jth interval. With in the first design

<table>
<thead>
<tr>
<th>Component number (m)</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability (r(m))</td>
<td>0.5774</td>
<td>0.8674</td>
<td>0.6704</td>
</tr>
<tr>
<td>Human reliability</td>
<td>0.9789</td>
<td>0.9341</td>
<td>0.9191</td>
</tr>
</tbody>
</table>

sub phase, we have in all nine bugs to be rectified. The inter arrival occurrence time interval of bugs (0, 2, 5, 20, 14, 10, 22, 5, and 1) respectively, follows Poisson distribution random number generation pattern. The time taken for rectifying the bugs (4, 6, 4, 10, 5, 25, 5, 7, 6, and 2) respectively was also generated using the coding for negative exponential distribution function. The first bug is assigned to the first software engineer for debugging. Initially, since the software engineer is free, no queue is formed and debugging process starts. The second, third and fourth bug on their arrival, is also assigned to first software engineer, forming a queue, as the cumulative departure time is less than the stipulated time of development of the first phase i.e. 25. Rest of all arrived bugs will be assigned to new hired software engineer so that all the nine bugs are fixed in bounded time. Since the residual bugs are not to be carried to next development phases, thus the simulation process completes along with the completion of the development phases. In reference to Table-2 only nine bugs, out of twelve bugs, were fixed during the 1st phase, rest of the three will be treated as residual bugs. The results obtained in respect to the reliability parameters such as, Failure density and Software reliability corresponding to each of the development phase have been presented in columns 6th, 7th and 8th respectively, of Table-2. The process of simulation will be repeated in the similar manner for other design sub phases. Since each phase of event driven software is developed in series, so total reliability for design phase is the product of reliabilities of each phase of design.
Table 1.

<table>
<thead>
<tr>
<th>Bug number</th>
<th>At (i)</th>
<th>St (i)</th>
<th>cat (i,1)</th>
<th>Cat (i,2)</th>
<th>Cdt (i,1)</th>
<th>Cdt (i,2)</th>
<th>wt (i,1)</th>
<th>Idt (i,1)</th>
<th>wt (i,2)</th>
<th>Idt (i,2)</th>
<th>q1 (i,1)</th>
<th>qi (i,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>-</td>
<td>4</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>10</td>
<td>7</td>
<td>-</td>
<td>20</td>
<td>-</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>5</td>
<td>27</td>
<td>-</td>
<td>25</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td>25</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>25</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>30</td>
<td>-</td>
<td>-</td>
<td>15</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>22</td>
<td>7</td>
<td>-</td>
<td>32</td>
<td>-</td>
<td>37</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>2</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>6</td>
<td>-</td>
<td>37</td>
<td>-</td>
<td>43</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>3</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>38</td>
<td>-</td>
<td>45</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>0</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.

Pictorial representation for overall reliability (table 1) for 1000 simulation runs is given by scattered graphs. These graphs conclude that the overall reliability of system tends to unity as there is an increase in the number bugs to be fixed.

Histograms for overall reliability verses frequency in 1000 simulation runs are plotted with major axis set to 0.1 with width as 0.1.

The patterns for histograms are reasonably close to normal distribution, which justifies the selection of our model (with erlang distribution) for simulation.

Graph 7 is the pictorial representation for average overall reliability verses percentage of bugs fixed. This graph predicts that as percentage of errors fixed increases, average overall reliability also increases.

The results confirm the results obtained by scattered graphs.
7 Conclusion

This simulator will be an asset to software developers for evaluating various SRE parameters, planning, tracking, monitoring and predicting the effect of management decisions, and thereby it helps the project managers in developing DSS/ESS for planning or tailoring the software development process for releasing the software.
product of given quality parameters within specified time. This Simulator can be extended to simulate the cost of the software product also.

References

Fig. 7.


Simulator for process reliability with reuse of component


[Ree04] Bernhard Reer, "Sample Size bounding and Context Ranking As Approaches to The HRA Data Problem", Reliability Engineering & System Safety, 2004


[SS05] Soni Ritu, Suri P.K, "Quality Assurance through Simulator", The International Conference on Modeling, Simulation and Visualization Methods (MSV'05), USA