International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 10, December- 2012

A Testing-Effort on Software Reliability Estimation Model

Rahul Chaudhary , Darothi Sarkar

Department of Computer Science, Amity University, Noida, Uttar Pradesh, India

Abstract- The software reliability growth models have become prominent in use. Although we know that the use of such model is delaying the SDLC (software development life cycle) i.e. during testing because these models requires failure time data in order to determine the models parameters. At a time we have more than one model such as Basic Execution Time model which can apply to SDLC in beginning, as its parameters can be directly depends on software characteristics although it only implies uniform execution of program instruction i.e. program without loops and branches. We have another model Extended Execution Time model that consider this non uniformity of program, which is related to measurable characteristics of software product, and this help to identify which model is best to apply for Reliability estimation according to degree of uniformity i.e. Uniform execution- Basic Execution Time Model, Non-uniform execution- Musa okomoto logarithmic model, Our work is to perform EET Model (Extended Execution Time) for software reliability estimation with testing side by side i.e. implementation of Functional Testing (part of Black Box Testing (Structural testing)) with Representative Testing in parallel over the result of EET model.
This helps us as follows:
Applying functional testing together with Representative testing after or parallel to EET model help us to detect and remove the concrete faults and develop concretely reliable software Structural Testing Methods have been underestimated due to less deterministic results [1] although it supports automated constraint solving capacity and on another hand Representative Testing [3] that allow reliability estimation modeling, to give the desired quantification or determination. If we use both testing in parallel or conjunction then results from Structural Testing can help us to update the reliability estimates from EET model that are conventionally connected with Representative testing. Here we use order-statistics to merge the observed failure rates of faults, nothing to attention that how those faults were identified [3].

Index Terms- Basic Execution Time model (BET model), Extended Execution Time Model (EET Model), fault detection, faults spraying, Functional Testing (Structural testing)), imperfect ordering, non-uniform execution, perfect ordering, Representative testing, Sorted activity profile, test case, uniform execution.

I. INTRODUCTION

Positively towards the development of reliable software we use various reliability growth models these various models estimate the appearance of failure occurrence during testing, and removal of these faults helps in improvement of reliability. These various models which are used are analytical in nature; generally have one or more variable whose values are unknown. Hence for the application of this model, one must first estimate values of unknown parameters. The only software reliability model that has usefully in predicting reliability prior the availability of all data (that are required to estimate reliability) from the test – the Musa's Basic Execution Time model [MIO]. Parameters of this model can be similar to such measurable software product and process characteristics as faults- density, number of newly developed source instructions, the number of machine instruction and speed of the machine on which the product runs. Same as BET (Basic Execution Time) model, EET model accept two parameter as in BET model and the third parameter is 'a',

Table 1

a	**Types of execution**
>4	very non uniform, characterized by many infrequently extended path associated with branches and loops with many iterations.
.2, 4	moderately uniform.
< .2	nearly uniform, few infrequently executed path associated with branches, few loops with few iterations associated with each loop.

evolved related to a measure of uniformity of instruction execution. Where "a" is defined as using Table1 [2]: The change in the table 1 is in upper limit which is reduced to 4, because each program complexity increases rapidly with little change. I.e. When 'a =0' all instructions are executed uniformly, at that time EET model identical to BET model and for large value of "a" parameter show very non- uniformity in execution of instruction then, [MIO] the logarithmic model is best applied i.e. for non- uniform execution in case of moderately uniform condition, we refer to apply logarithmic model as we are certain on that the, there is obviously degree of non- uniformity.

The focus of EET model is its capability to estimate the parameter before the collection of failure data. Two parameter can be predicted in precisely the same manner as for BET model as parameter related to program execution hence the" a" parameter can be derived from instruction "execution profile" data.

EET model over comes one problem faced by users in deciding whether to use the BET model or the logarithmic model. The "a" parameter is decided for program using UNIX "profile facility".

NHPP model as it incorporates both model depending on value of the parameter "a". Measure of "uniformity" done by finding linear execution frequency [1- section2] and measure

of "non- uniformity" by **activity profile** and **sorted activity profile** [1-section 5]. Hence various models simulates failures occurrences during testing and whose removal result growth in reliability is already defined, now applying functional testing together with Representative testing help us to detect and remove the concrete faults and develop concretely reliable software. In functional/ structural testing test cases are created using functional specification [6] or program requirement. Example when – I option used in CMD line of UNIX sorting program it causes sort to ignore characteristics outside ASCII Range. Hence we to point that different individual are able to construct different test sets for testing a single program in hand. Thus, it is also likely that the designed test sets will cause to detect different tests sequences which are further used in designing to variations of functional testing used to elaborate more to reliability test. All this testing done with representative testing which gives the required quantification and when it is merged with structural testing, this combination is shifting the observed random variable from inter failure time to a detailed analysis of debugged faults. Representative testing helps to find most commonly occurring faults, first hence time to detect new faults that are generated due to new innovations increases over time. Hence merge of these two testing methods i.e. structural and representative testing may do well because

Structural testing- considered as defensive approach against potential uncertainly [3- section 1], in the operational profile and catalyses the faults detection states in detection of testing process.

Representative testing- can give quantified measures of progress being made using order statistics to combine the observed failure of faults. Section 3 explains the summary of EET model, section 4 explains the parallel implementation of testing paradigms, and section 5 is conclusion.

II. Reliability and Testing

Reliability is a famous concept that has been celebrated for years as a necessary attribute of a product or a person. Its beginning was in 1816, far before than any one guess. The word "reliability" was first given by Samuel Taylor Coleridge who was a poet. In statistics, reliability is the consistency of a set of parameters, which are used to satisfy a test. Reliability is inversely related to unknown error. In Psychology, reliability refers to the consistency of a symptom. A test is considered reliable if we get the same result in every pass of test. For example, if a test is created to measure introversion, then the time the test is heading to a subject, the results should be approximately the same every time.

Testing is a necessary and critical part of the software development process, on which the reliability and quality of the product completely depend. Testing is not bounded to the finding of "bugs" in the software, but also boost up confidence in its proper working and with the selection of functional and nonfunctional properties. Testing related activities measure the complete progress of process and may consume an enough effort required for producing software.

A. Challenges in reliability estimation: (1) how can we effectively use the architectural specifications of a component to construct an efficient reliability estimation model? And (2) how do we deal with the uncertainties that stay in this model due to the absence of operational profile?

B. Estimating and analyzing software reliability: during testing is a branch with over 30 years of past experience. Many reliability models have been given: Software Reliability Growth Models i.e. SRGMs which are used to estimate software reliability using statistical approaches [6, 8, 9, and 10]. The common theme across all of these approaches, however, is their possibilities to implementation-level artifacts, and reliability estimation during testing. At architectural level, existing reliability estimation approaches consider only the structure of the system in hand. The exceptions are [7, 11, 12, and 13]. However, none of them approaches consider the effect of a component's internal behavior on its reliability.

III. Summary of EET Model

EET modal summarized as follow first divide the system into parts (cells) of equal size, although it should be focused that there is uniform execution of the instruction in each cell. The failure caused by fault occurrence can be modeled with the help of BET model[1- section 2] , considering a unique functional form to show the amount of time spent in each cell that allow the bet models for all the cell to be merged to give a compound model for the software in whole. Consider our division consists of m cells. Let e_i represent the execution time utilized for instruction in a cell i and $F_i(e_i)$ are the total failure that occurs in execution of instruction in cells i .Now $F_i(e_i)$ is random variable whose distribution [1- section 2]

$$\text{Prob}\{F(e_i) = n\} = \frac{\mu_i^n(e_i)}{n!} e^{\mu_i(e_i)} \qquad 2.1$$

$\mu_i(e_i)$ - means value of $F_i(e_i)$

e_i - Execution time for machine instruction in cell i .

Since execution of instruction is uniform

In each cell, then according to BET model $\mu_i e_i$ given as

$$\mu_i(e_i) = w_{o_i}(1 - e^{-\phi_i e_i}) \qquad 2.2$$

w_{o_i} - Faults in cell i at time $t_i = 0$

ϕ_i - The rate of failure for each fault in i^{th} cell.

Recursively both parameter w_{o_i} and ϕ_i can further be allocated in terms of software characteristics of cell i [1-section 2]

$$w_{o_i} = DS_{I_i} \qquad 2.3$$

$$\phi_i = K \frac{R}{M_{I_i}} \qquad 2.4$$

Where

S_{I_i} - Source instruction cell i

M_{I_i} - Machine instruction in cell i

D - Fault density

Let e = $\sum_{i=1}^{m} e_i$ (i.e. Total execution time of instructions in cells of software components) and express e_i as $\alpha_i e$. The factor α_i is proportion of time spent in i^{th} cell. Since the cell i give a partition of the software component [1-section 2]. $\sum_{i=1}^{m} \alpha_i$ =1

Let $F(e) = \sum_{i=1}^{m} F_i(e_i)$ using the super position property of Poisson probability distributions, the distribution of $F(e)$ is also derived same i.e. as Poisson random variable with mean

$$\mu(e) = \sum_{i=1}^{m} \mu_i(e_i) = \sum_{i=1}^{m} \mu_i(\alpha_i e_i) \qquad 2.5$$

Now equation 2.5 is for any of the software partition and for any execution time distribution in partition of each cell. Consider a single distribution of time utilized in each cell for the division. Division of software component in to m equal cells, in such a way that there each cell contains equal number of machine instruction. Hence

$$M_{I_i} = \frac{M_I}{m} \qquad 2.6$$

Without generality loss, sorting the partition according to time spent in each cell. It is a Sorted Activity Profile [2- section 2].

Let time proportion spent in cell i be

$$\alpha_i = \frac{1}{c}\left[\frac{i}{m}\right]^a \qquad 2.7$$

Since $\sum_{i=1}^{m} \alpha_i$ =1, then the constant(c) can be evaluated as:

$$c = \left[\frac{i}{m}\right]^a$$

$$c = m\sum_{i=1}^{m}\left[\frac{i}{m}\right]^a \frac{1}{m} \qquad 2.8$$

Equation 2.8 is identified as the numeric approximation of the integral $\int_0^1 x^a dx$, where $0 \le x \le 1$ is partitioned into m cells of size $\frac{1}{m}$ and the area under curve x^a is probabilistic by the sum of individual cell area. Hence equation 2.8 written as

$$c = m\int_0^1 x^a dx = \frac{m}{a+1} \qquad 2.9$$

Thus, equation 2.7 written as –

$$\alpha_i = (a+1)\left[\frac{i}{m}\right]^a \frac{1}{m} \qquad 2.10$$

α_i, play vital role in formation of Sorted Activity Profile of the software component. Note, $(a+1).x^a$ is continuous analog of $(a+1).(1/m)^a$, and similarly dx continuous analogue of $\frac{1}{m}$.

Consider the density of fault D is equal is equal for each cell the S_{I_i} i.e. source instructions and the M_{I_i} i.e. machine instructions are uniformly distributed over the cells, the fault number w_{o_i} in cell i can be given by

$$w_{o_i} = D\frac{S_I}{m} = \frac{w_o}{m} \qquad 2.11$$

Where $w_o = DS_I$, total software component fault. The fault exposure ratios X_i are same to each cell i, the per fault intensity ϕ_i [1] [sec-4] can be given as

$$\phi_i = X\frac{R}{\frac{M_I}{m}} = m\phi \qquad 2.12$$

Where $\phi = X\frac{R}{M_I}$

Indicates per fault failure intensity of total software component, in case where execution is uniform. Thus

$$\mu(e) = \frac{w_o}{m}\sum_{i=1}^{m} 1 - e^{-\phi_i \alpha_i e} \qquad 2.13$$

$$\mu(e) = w_o \sum_{i=1}^{m} 1 - e^{-\phi_i \alpha_i e}\frac{1}{m}$$

$$\mu(e) = w_o\left[1 - \sum_{i=1}^{m} 1 - e^{-\phi_i (a+1)\left[\frac{i}{m}\right]^a e}\frac{1}{m}\right] \qquad 2.14$$

Hence equation 2.14 summation is just integral approximation

$$\mu(e) = \left[\int_0^1 e^{-\phi_i (a+1)x^a e} dx\right]$$

$\mu(e)$, can be expressed as:

$$\mu(e) = w_o\left[1 - \int_0^1 e^{-\phi_i (a+1)x^a e} dx\right] \qquad 2.15$$

Equation 2.15 can further compressed as

$$\mu(e) = w_o[1 - H(a,l)] \qquad 2.16$$

$$H(a,l) = \int_0^1 e^{-lx^a}\, dx, l = (a+1).\phi.e \qquad 2.17$$

Now equation 2.17 is for sorted activity profile (SAP) that can be draw with function x^a. Execution time $e = 0$, correspond to $l = 0$ which on placing in equation 2.17 give $H(a,0) = 1$, hence equation 2.16 after substitution of above value becomes $\mu(0) = 0$.

Similarly, when e reaches ∞, then equation 2.17 gives $H(a,\infty)=0$, so then equation 2.16 becomes $\mu(\infty)=w_o$ and the failure intensity

$$\lambda(e)=w_o.\phi.H.(\frac{a}{a+1},l) \qquad 2.18$$

Hence, the above result not depend on "a" i.e. factor of uniformity, hence model reduced to BET model, properties of $H(a,l)$ are summarized in appendix A.

Now, suppose $a->\infty$ in equation 2.17 and 2.16 then execution of instruction is very non uniform. If l depend on a and reaches ∞, however let $a->\infty$ with $(a+1)\phi$ remaining constant. This show ϕ would have to approach 0, due to which w_o ϕ approach 0. Unless w_o approaches ∞ .Take $a->\infty$ limit such a way $(a+1)\phi=\beta_2$ and $w_o\phi=\beta_1$ remain constant. From equation 2.18 for failure intensity becomes

$$\lambda(e)=\beta_1 H(1,\beta_2 e) \qquad 2.19$$

$$\lambda(e)=\beta_1\frac{1-e^{-\beta_2 e}}{\beta_2 e}$$

Integrating last two equations, we get

$$\mu(e)=\frac{\beta_1}{\beta_2}\int_0^{\beta_2 e}\frac{1-e^{-x}}{x}dx \qquad 2.20$$

$$\mu(e)=\frac{\beta_1}{\beta_2}o(\beta_2 e)$$

Properties of $o(x)$ function are summarized in Appendix A.

Now equation 2.19 & 2.20 shares many same properties of Logarithmic NHPP model described in reference [MIO].This lead to EET model to Logarithmic NHPP model.

The cumulative failure number and failure intensity may be expressed as

$$\mu(e)=\frac{\beta_1}{\beta_2}\log(\beta_2 e+1) \qquad 2.21$$

$$\lambda(e)=\frac{\beta_1}{\beta_2 t+1} \qquad 2.22$$

Due to asymptotic behavior of both models, cumulative number of failure and failure intensity as $e->\infty$

$$\mu(e)=\frac{\beta_1}{\beta_2}\log(\beta_2 e)$$

$$\lambda(e)=\frac{\beta_1}{\beta_2 e}$$

Similarly, for small e, the EET model cumulative number o failure can be →

$$\mu(e)=\frac{\beta_1}{\beta_2}\left[l-\frac{l}{2.2!}^2+\frac{l}{3.3!}^3-\ldots\right],$$

This is much same as Logarithmic NHPP expansion for small e for the cumulative number of failure

$$\mu(e)=\frac{\beta_1}{\beta_2}\left[l-\frac{l}{2}^2+\frac{l}{3}^3-\ldots\right]$$

In this section, we can reach very near to find the reliable software as by finding equations that can be implemented to draw sorted activity profile and cumulative failure versus time graphs.

IV. PARALLEL IMPLEMENTATION OF TESTING PARADIGAM

Here comes the most crucial step for your research publication. Ensure the drafted journal is critically reviewed by

This section is described in two parts

3.1) Step to implement functional testing

3.2) Representative testing

3.1) Step to implement functional testing:

All part is assumed to be conducted over the UNIX sorting program [2] or variations obtained by adding faults. We have Notations

e, e'	[Execution, exposure] time
ϕ	Per-fault hazard rate
e_i	Time of failure i
E_{Fi}	Reliability estimates for functional testing ($i=1,2$)
F_R	Fault reduction factor

The two functional testing usage causes explained in introduction.

Hence we use two different specification sequences and also there combinations for functional testing. Testing that we use is referred as functional_test_1 & functional_test_2 below given discussion as follow: these two testing performed on data collected [summarized in appendix-B], failure data gathered with the help of two different sets of functional test data from appendix-B.

These failure data were used with EET model to compute reliability estimates. Reliability, estimates that were obtained- After K failures.

The execution time (e-units), and by comparing performance of different method of testing.

Select K=10 and e=12.3, as they were the smallest value at which $F_R.\phi$ > 0. Exposure time -> total execution time expanded after the software brought to its operational environment.

The above explanation is further used to draw graph between reliability and exposure time for the function_test_1, function_test_2, also for the block testing and random testing.

3.2) Representative Testing (as explained in introduction)

3.2.1) Perfect Ordering

Let start with the consideration that is common to most of reliability models, that are used during the testing, faults detected in non-increasing order by operational failure rate. I.e. our debugging and testing process remove the most common faults first. The relation between total set of faults and observed faults, which is then given by:

$$\psi_i=\psi_{k-i+1.k}=\phi_{m+1-i.m}, 1\le i\le k \qquad 3.2.1$$

ψ_i =operational failure rate.

And then after K faults the program failure rate is

$$\lambda_k = \sum_{i=1}^{m-k} \phi_i m$$

We have to follow little hood model [9] and we use to take the effect of fault ordering as more visibly by explicit use of order statics.

Although we don't know value of ϕ for faults that have not yet been detected, but we try to compute their expected estimate.

$$E(\phi_{r.m}) = \int_0^\infty \phi . f_{r.m}(\phi) d\phi$$

Using density function [4]

$$E(\lambda_k) = \sum_{j=1}^{m-k} \int_0^\infty \phi . f_{j.m}(\phi) d\phi \qquad 3.2.2$$

This is resulting estimation for λ_k.

3.2.2) imperfect ordering

Here is assumption that decreasing order of failure rate is only an approximation detected by representative testing. For any two faults, there must be a certain probability that less common fault happened to found firstly. A perusal of data that publish sets such as [5] reveals although failure rate range may indeed several order span of magnitude, hence the expected decrease in failure rate from one to the next fault is quiet small when compare to overall program failure rate, i.e. some fault which are out of order are found in any particular test history.

We would think that order of this sort odd deviation or fault detection will be comparatively small. I.e. a fault is not to be found more than a few positions out of expected order. Thus if, we were to sort fault into non increasing order by failure rate, the reconstruction of resulting sequence is almost happen expected order of fault detection. We modeled by assuming that

$\psi_i = \psi_{k-i+1.k} = \phi_{m+1-i.m}, 1 \le i \le k$, is a much weak condition that the equation 3.2.1 although equation still hold as an estimate of λ_s.

V. CONCLUSION

We have given reliability growth model which allow quantification of the effect of applying directed tested method over the EET model. By shifting directly observed inter failure time quantity to the individual faults failure rate. This model allows all test plans (that are the combination of representative and directed (functional) testing methods and contribute) to attain a common goal of reliability, although doing the EET model with parallel implementation of representative testing and directed (functional) testing is great idea. This is also a way to explore more in reliability estimation. Implementation of this is done future trends since it combines the advantages of both EET model and testing techniques, thereby providing optimal solution of reliability estimation. According to Recent researches, by the end of 2013 there will be many new methods are evolved in reliability estimation field by using this paper.

.

APPENDIX A

Properties of $\mathbf{H}(a,l)$

This appendix drives and summarized properties of function $H(a,l)$. The definition of $H(a,l)$ is

$$H(a,l) = \int_0^1 e^{-lx^a} dx \qquad \text{A.1}$$

Note, that $H(a,0) = 1$ and $H(a,\infty) = 0$ with a little change of variable (exchange lx^a with x) during integral in (A, 1), the following alternate expression for $H(a,l)$ can be obtained.

$$H(a,l) = \frac{1}{a l^{\frac{1}{a}}} \int_0^1 e^{-x} x^{\frac{1}{a}-1} dx \qquad \text{A.2}$$

$$H(a,l) = \frac{1}{a l^{\frac{1}{a}}} \gamma(\frac{1}{a}, l)$$

Where $\gamma(\frac{1}{a}, l)$ is the incomplete gamma function [AB page number 260].

If the exponent function in (eqn-A.1) is integrated step by step and expanded in its power series, the following power series expansion for $H(a,l)$ results.

$$H(a,l) = \sum_{m=0}^{\infty} \frac{(-l)^m}{(ma+1)m!} \qquad \text{A.3}$$

$$-\infty < l < \infty; a \ge 0$$

For specific value of a, $H(a,l)$ reduces to simple functions:

$$H(a,l) = \int_0^1 e^{-l} dx = e^{-l} \qquad \text{A.4}$$

$$H(a,l) = \int_0^1 e^{-l} dx = \frac{1-e^{-l}}{l} \qquad \text{A.5}$$

For integer values of $\frac{1}{a}$ (e.g. $\frac{1}{a} = k$), equation A.2 can be repeatedly integrated by parts to obtain

$$\mathrm{H}(\frac{1}{\mathrm{k}}, \mathrm{l}) = \frac{k! e^{-l}(e^l - E_k(l))}{l^k} \qquad \text{A.6}$$

Where $E_k(l)$ are partial sums of the power series for the exponential function i.e.

$$E_k(l) = \sum_{m=0}^{k-1} \frac{(-l)^m}{m!}, E_0(l) = 0 \qquad \text{A.7}$$

Differentiating the power series expansion (A.3) for $H(a,l)$ step by step, the result obtained is:-

$$\frac{dH(a,l)}{dl} = \sum_{m=0}^{\infty} \frac{-(-l)^{m-1}}{(ma+1)(m-1)!}$$

Using the relation

$$(ma+1) = (a+1)\left[(m-1)\frac{a}{a+1}+1\right] \qquad \text{A.9}$$

An asymptotic expansion for $H(a,l)$ valid for large values of positive l can be obtained by first rewriting (A.2) as

$$H(a,l) = \frac{1}{al^{\frac{1}{a}}}\left[\Gamma\left(\frac{1}{a}\right)\int_1^{\infty} e^{-x}x^{\frac{1}{a}-1}dx\right] \qquad \text{A.10}$$

Integrating by parts in the integral in (A.10) results in

$$H(a,l) = \frac{1}{l^{\frac{1}{a}}}\Gamma\left(\frac{1}{a}+1\right) - e^{-l}\left[\frac{1}{al}+\frac{1-a}{(al)^2}+\frac{(1-a)(1-2a)}{(al)^3}+\ldots\right] \qquad \text{A.11}$$

We define the function $o(l)$ as

$$o(l) = \int_0^l H(1,x)dx = \int_0^l \frac{1-e^x}{x}dx \qquad \text{A.12}$$

This function appears in the limiting version of the EET model where a-$>\infty$. Integrating the power series expansion for the exponential function results in

$$o(l) = x - \frac{x^2}{2.2!}+\frac{x^3}{3.3!}\ldots \qquad \text{A.13}$$

Following a similar approach as was done in deriving A.11, the following asymptotic expression for

$$o(l) = \log(l) + 0.5172 + e^{-z}\left(\frac{0!}{l}-\frac{1!}{l^2}+\frac{2!}{l^3}-\ldots\right)$$

APPENDIX B

Failure data collection

Experiment conducted to obtain failure data using 3 testing methods i.e. random testing, block testing and functional testing, all parts of this experiment have been implemented using Unix sorting program or variants of this program obtained by spraying faults.

For random testing, we come with different seeds to get 29 different sequences of failure data $\tilde{s}_i, i = 1,2,3,\ldots,29$. $\tilde{s}_i$ is obtained by executing sorting on a total of T_i test cases. Testing stop after all 9 faults in sorting were removed.

Notations:

$$\tilde{s}_i \left\{\tilde{t}_i, \tilde{t}_{i.2}, \ldots, \tilde{t}_{i\,\tilde{k}_i}\right\}, 1 \le i \le 29$$

$\tilde{k}_i$ Total failure required before all the 9 faults revealed from sort

$\hat{s}$ Sequence of average failure interval, which help to obtain estimation of reliability

f_i Fault $i, i \in [1,9]$

Rearrange the sequence such that $k_1 \ge k_2 \ge \ldots \ge k_{29}$ to

Obtain the sequence $s_1, s_2, \ldots, s_{29}$ then,

$$\hat{s} = \left\{\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_{\tilde{k}_i}\right\},$$

$$k = gilb\left(\left[\sum_{i=1}^{29} k_i\right]\Big/ 29\right)$$

$$\hat{t}_j = \left[\sum_{i=1}^{l} t_{ij}\right]\Big/ l,\ j = 1,2,\ldots,k$$

$$l = \max(\, m \mid k_m \ge j);$$

All failure data obtained while testing are used to calculate reliability estimates from the EET model and described in Section 2. Parameter in EET model were estimated with the help of failure data, if debugging is imperfect, implies that even in case on which program failed, testing is continued without repairing the fault.9 faults were sprayed in to the Unix sorting program ,the fault type are a sample from commonly occurred faults that are entitled by other researchers. The sorting program itself is about 1000 lines of executable c-language code. The failure data was generated is by executing the below given sequences of steps.

1: Ten faulty versions were conducted on sorting as follow
Sort-0: sorting having all 9 faults
Sort-1: sorting contains faults f2,…,f9
Sort-I: (2<=i<=8): sorting having faults f1 , … , fi-1, fi+1, … ,f9
Sort-9: sorting having f1,…,f8 faults

2: Repeat this step for each of 3 testing methods until every fault is not found.
[Initialize] faults-detected to 0
[test-case construction] construct a test case C, this is manual work for functional & block testing, and by program in case of random testing. In any case, output correctness is checked automatically by an oracle.
[Program execution] determine if test case C causes at least one fault to remove. Execute sorting and sort 0 against C. if result of sort 0 is correct then C is success with respect to UNIX sorting program. Repeat step 2b. If the output of sort 0 is not correct then a failure has occurred. Note the total execution time b/w previous & this failure.
[Fault removal] imperfect debugging procedure is used. Execute sort 1 to sort 9 on C constructed in step 2b as the output of all sort i is reviewed sequentially, starting with i=1. The first correct sort $i, i \in [1,9]$ implies that f_i is the fault which is responsible for the failure. In this case f_i is the fault which is responsible for the failure. In this case f_i is considered removed. Replaced sort 0 by sorts i, increment faults-founded by 1 and go to step 2e. If none result from all 9 sort is correct then assume that the fault is not removed and repeat from step 2b.
[Check for termination of the experiment] if faults founded < 9, then repeat from step 2b; otherwise this procedure

terminates for one testing method

Step 2d implements prefect debugging by analyzing the outputs of all variants of the sort program since sort 0 having all 9 faults correct behavior of this program says that test case C cannot remove any of these faults. However, if sort 0 not work or fails then we need to select which fault, or a combination results in failure.

To do so, we execute the other remaining 9 versions of sort. If exactly 1 sort out of 9 sorts generates correct output on Test case C, we say that \

The fault that caused sort to failure is f_i .

The fault selected & corrected.

If none of all version of sorts i, generate correct output. Then it is obvious that failure is caused due to combination of all 9 faults and not by only one fault. In this case no fault is corrected.

All experiment were conducted on a sun-sparc-machine ([2] section 4) and method used is one of perfect debugging assumption. But it is not found yet that how imperfect debugging actually work in real environment ([2] section 3).

ACKNOWLEDGMENT

The authors are very thankful to their respected Ms. Darothi Sarkar, faculty, computer science department, Amity University, Noida, Uttar Pradesh, Authors also pay their regards to Dr. Abhay Bansal, professor & Head of ASET, Amity University,Noida, Uttar Pradesh for giving their moral support and help to carry out this research work.

REFERENCES

[1] *An "extended execution time" software reliability model; W.W.Everett;AT&T Bell Laboratories Holmdel, New Jersey;1992 IEEE*

[2] *Effect of Testing Technique on Software Reliability Estimates Obtained Using A Time-Domain Model;IEEE; 1995 march*

[3] *A Reliability Model Combining Representative and Directed Testing;Brian Mitchell and Steaven J.Zeil;Old Dominio University;Deptt. Of Computer Science Norfolk,VA 23529-0162*

[4] *DAVID, H.A. Order Statistics, Second ed.Jhone Wiley and Sons, 1981.*

[5] *W.E. Howden,"Functional Testing",IEEE Trans.Software Engineering, vol SE-6, 1980 MAR, pp 162-169.*

[6] *Goel A.L., Okumoto K., Time-Dependent Error-Detection Rate Models for Software Reliability and Other Performance Measures, IEEE Trans. on Reliability, 28(3):206–211, 1979.*

[7] *Goseva-Popstojanova K. et al., Comparison of ArchitectureBased Software Reliability Models, in ISSRE 2001, pp. 22-31.*

[8] *Jelinski, Z. and Moranda, P. B., Software Reliability Research, Statistical Computer Performance Evaluation, edited by W. Freigerger, Academic Press, 1972.*

[9] *Littlewood, B.A., and Verrall, J.L., A Bayesian Reliability Growth Model for Computer Software, Applied Statistics, Volume 22, pp. 332-346, 1973.*

[10] *Musa J.D., and Okumoto K., Logarithmic Poisson Execution*

[11] *Reussner R., Schmidt H., Poernomo I., Reliability prediction for component-based software architectures, In Journal of Systems and Software, 66(3), Elsevier Science Inc, 2003.*

[12] *Wang W., Wu Y., Chen M., An architecture-based software reliability model, in Proc. of Pacific Rim International Symposium on Dependable Computing, 1999.*

[13] *Yacoub S.M., Cukic B., Ammar H.H., Scenario-Based Reliability Analysis of Component-Based Software, in 10th Int'l Symposium on Software Reliability Engr., Boca Raton, Nov. 1999.*

[AB] *Abramowitz, Milton and Irene A. stegun; Handbook of Mathematical Functions; National Bureau of Standards Applied mathematics Series-55; june 1964*

[MIO] *Musa J. D; A. Iannino, K.Okumoto; Software Reliability-measurment, prediction, Application; McGraw Hill; 1987.*

AUTHORS

First Author –Rahul Chaudhary, pursuing master of technology (1st year), from computer science department, Amity University, Noida, Uttar Pradesh, and done there Bachelor of technology degree from IAMR college of engineering, Meerut, Uttar Pradesh, His area of interest includes Software engineering and Reliability, Email id: rahulch369@yahoo.in

Second Author– Darothi Sarkar, M.Tech, faculty, computer science department, Amity University, Noida, Uttar Pradesh, Email id: dsarkar@amity.edu.

Correspondence Author –Rahul Chaudhary, pursuing master of technology (1st year), from computer science department, Amity University, Noida, Uttar Pradesh, and done there Bachelor of Technology degree from IAMR college of Engineering, Meerut, Uttar Pradesh, His area of interest includes Software engineering and Reliability, Email id: rahulch369@yahoo.in.