IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-5, May 2019 E-ISSN: 2347-2693

Enhancement in Software Reliability Testing and Analysis

P. N. Moharil[1*], S. Jena[2], V.M. Thakare[3]

[1]Department Computer Science, Research scholar, SGBA University, Amravati, India
[2]Departments of CSE&A, SUIIT, Sambalpur University, Odessa, India
[3]Departments of CS&E, SGBA University, Amravati, India

[*]Corresponding Author: pmohril@gmail.com, Tel.: 9421382386

DOI: https://doi.org/10.26438/ijcse/v7i5.5764 | Available online at: www.ijcseonline.org

Accepted: 08/May/2019, Published: 31/May/2019

Abstract— Reliability of software is an important factor of today's software industry. The techniques involved in the designing, testing & evaluation of a software system is called as software reliability engineering. The increasing demand for the reliability of software products need to be estimated and the estimation of such software systems in reliability engineering aspects become more critical for large scale projects. The ability of a system to perform system required operations or functions under the given condition for a specified period of time is called the reliability of the system. Software reliability is described as the probability of failure-free software operation for a given period of time in assign environment. This paper proposes an algorithmic design of reliability analysis model for predicting quantitative & qualitative analysis of software reliability by using probability theory & statistical analysis with a set of techniques and models. Reliability of software is a squeeze with the prevention of errors, faults finding or detection of faults & removal. Reliability of hardware is not predicted with probabilistic functions where the reliability of software is measured of probabilistic function with the notion of time.

Keywords — Software Reliability, Reliability analysis, reliability growth model performance testing

I. INTRODUCTION

Software reliability is the key factor of software quality with multidimensional property counting other aspects such as usability, availability, capability & maintainability. Software reliability predicts the quantitative & qualitative analysis of software by using probability theory & statistical analysis with a set of techniques and models. Software reliability testing becomes an essential factor of any software to improve the performance of software through assessment of the reliability of the software product & software development life cycle. Conclusively testing reliability of software becomes an important aspect. Reliability testing is performed for removal of defects and fault findings before system deployment. Software failures are different from hardware failures. Software failures are measured or estimated using models. Software reliability model determines the general form of the dependencies of the failure process on the principal aspects which affects it [1]. The aspects are fault detection, fault introduction, fault recovery, removal & operational environment. The fault prediction technique performs the evaluation for reliability, which model & measures the reliability of the system during its operations. Reliability of software is a squeeze with the prevention of errors, faults finding or detection of faults & removal. It remains a critical problem because there is no complete solution to the nature of software & there is no clear idea about what aspects are related to software reliability. Reliability metrics are used to bring out the reliability to the software product. Depending upon the type of system & the requirements of the application fields the selection of reliability metrics used [2].

In this paper, an algorithmic design of reliability analysis model is presented for software reliability analysis and performance testing of web applications. The paper is organized as follows. In section II statistical or mathematical reliability models are described. Section III describes the reliability measurement. Section IV. Define an algorithmic design of software reliability analysis model. Section V shows the implementation of software reliability growth model using SMERFs tool. Section VI shows the results and discussion. Section VII. Define conclusion & research work with future directions.

II. RELATED WORK

Software reliability models are statistical models used to make predictions about a software system's failure rate and given the failure history of the system within a given time period [3]. These models make assumptions about the fault detecting and removal process. These assumptions determine the form of the model and the meaning of the model's parameters. These model are categorized into two categories

- Input domain reliability models (IDRMs)
- Time domain software reliability growth models (SRGMs)

- Input domain reliability model(IDRMs)

IDRMs is also called a fault seeding model, where the number of indigenous and seeded faults is counted at the time of testing and the number of indigenous faults and the reliability of the software are estimated.

- Nelson model

In this model, the reliability of software is measured by executing the software for a sample of N inputs. The N inputs are randomly selected from the input domain. The randomized sampling of N input is performed through either probability distribution or simple through user input distribution. Where R has estimated the reliability

$$R = (n - f)/n = 1 - (f/n) = 1 - r \quad (1)$$

Where,

R = is the reliability
n = is the number of input

- Time domain software reliability growth models (SRGMs)

1. Jelinski-Moranda Model

This is mostly used software reliability estimating the model. While performing reliability testing number of N independent faults are considered. During the testing process, new defect or faults are not introduced until the current fault is removed from the test. It calculates software failure rate, failure rate calculated is t time between $(i-1)^{th}$ & i^{th} failure is given by

$$Z(t) = \phi[N - (i-1)] \quad (2)$$

Where,

Φ = is proportionality constant
t = failure rate
N = number of independent faults

2. Goel and Okumoto Debugging Model

In this model, the number of faults in the system at the time t, X (t), is treated as a Markov process whose transaction probability is governed by the probability of testing. The time between the transitions of X (t) is taken to be exponentially distributed with rates dependent on the current fault content of the system. The hazard function during the interval between the $(i-1)^{th}$ and the i^{th} failure is given by the function:

$$Z(t) = \phi[N - p(i-1)]\lambda \quad (3)$$

Where

N = is the initial fault content of the system
p = is the probability of imperfect debugging
λ = is the failure rate per fault

3. Littlewood-Verrall Bayesian Model

In this model, the times between failures are assumed to follow an exponential distribution but the parameters of this distribution are treated as a random variable with a gamma distribution that is:

$$F(t1|\lambda 1) = \lambda 1 e^{-\lambda i t1} \quad (4)$$

Where,

λ= is random variables
t = is failure rate
F= failures

4. Musa Execution Time Model

In this model, the reliability of software is analyzed based on execution time. The hazard function for this model is provided as:

$$Z(\tau) = \emptyset f(N - nc) \quad (5)$$

Where,

τ = is the execution time
f = is the linear execution frequency
Θ = is a proportionally constant fault exposure ratio
nc = is the corrected number of faults

5. Goel-Okumoto NHPP Model

It is assumed that at random interval software system is subject to failures due to the faults in the present system. Let N (t) is the cumulative number of failures identified in time t then N (t) can be modeled as the NHPP model for the following.

$$P\{N(t) = y\} = (m(t))^{y}_{y!}{}^{e^{-m(t)}}, y = 0,1,2,.. \quad (6)$$

$$m(t) = a(1 - e^{-bt}) \quad (7)$$

$$\lambda(t) = m(t) = abc^{-bt} \quad (8)$$

Where,

m (t) = is the expected number of failures
λ (t) = observed by time t,
a = is the expected number of failures observed
b = is the fault detection rate per fault.

In this case, the number of faults to be identified is treated as a random variable whose values depend on the test factor [4].

III. RELIABILITY MEASUREMENT

Depending upon the nature of the system under study substantially specified time varies, as the reliability is a function of time. When the system is expected to operate without interruptions high reliability should be required. The predicted probability of the system to operate without failure is the reliability of the software [5]. Reliability $\mathcal{R}(ti)$ of the system at time t is the probability that the system operates without defects in the interval $[0, t]$ specifies that the system was achieving appropriate at time 0. For example, when system supposes to do some work within a short period of time, time is specified with units such as milliseconds, seconds & minutes.

To determine the adequate reliability of the system one should depend on the concepts related to the software application. The failure of the system can be declared in the form of probability. Thus, it specifies that the software reliability bet on the concepts of the probability theory. The probability theory contributes to the basics of software reliability & grants comparisons among the system. It also provides fundamental logic for enhancements of the defect rates, which will occur during the application life cycle [6].

A system may be required to perform various functions, each of which has different reliability level. In addition, in a different time, the software application can have a different probability to perform required functions from the user under declared conditions. Thus, reliability represents the probability of the system. It is defined as the probability that a product will perform a required function without failure for a stated period of time [7]. In other words, it is stated that it is a measure of how long it takes for a network (or a system) to fail.

Mathematically, reliability $R(t)$ is the probability that a system be successful in the time interval: [0 - t]: A failure rate, F, specifies the failure frequency in terms of failures per unit time.

$$F(\lambda) = \frac{Total\ no.of\ failure\ units}{Total\ unit\ test\ time} \quad (9)$$

Where,

λ = failure rate

Example 1: Assume that there are 40 units which operated for 1000 hours on test and 2 of them failed. Then the failure rate (λ) will be

$$F(\lambda) = \frac{2}{40 * 1000} = 5 * 10^{-5}$$

Before calculating the reliability, let's take a look at an example first which is going to figure out build the instinct Example

$$PF = \frac{Number\ of\ failures\ units}{number\ of\ products * total\ number\ of\ test\ time} \quad (10)$$

Where,

$PF =$ Probability failure

Let's say if there are 40 products operated for 1000 hrs in a test and 2 of them failed, then the probability of failure of the component in 1000 hrs is:

$$PF = \frac{2}{40 * 1000} = 0.05$$

$$PS = \frac{Total\ number\ of\ units - Number\ of\ failures\ units}{number\ of\ products * total\ number\ of\ test\ time} \quad (11)$$

Where

PS = Probability of success

$$PS = \frac{38}{40 * 1000} = 0.95$$

And the probability of success for the component in 1000hrs, thus, the Reliability is the probability of no failure within a given operating period.

$$P(S) + P(F) = 1 \quad (12)$$

$$P(S) = 1 - P(F) \quad (13)$$

Thus it is specified that probability of success P(S) is the reliability,

$$F(t) = 1 - e^{\wedge}(-\lambda t) \quad (14)$$

The cumulative distribution function which specifies the probability of failure by time t, If it subtracts that from 1, it will specify the use of p the probability of success of a component by time t which is **Reliability [8]**.

$$R(t) = F(t) => 1 - e^{\wedge}(-\lambda t) => e^{-\lambda t} \quad (15)$$

Where

R(t) = Reliability & probability of a unit failing by the time t from

F (t) = probability

(1-F (t)) = remaining probability

F (t) = probability, so the remaining probability (1-F (t)) gives us the probability of it's not failing by the time t. Reliability is the probability of no failure within a specified function period of time.

IV. ALGORITHMIC DESIGN OF SOFTWARE RELIABILITY ANALYSIS MODEL

This algorithmic design specifies the reliability measurement process, algorithm describes that the software is quantitatively determined by a study of failures & failure hardness, also by characterizing reliability objectives, & by determining balance among key quality objectives such as reliability, deployment of software, cost & estimation of time [9].

```
Step 1: Determine reliability objectives
Step 2: Perform system test
Step 3: Gathering of failure data
Step 4: Identifying relevant reliability measurement
        model
Step 5: Choose perfect reliability testing tool
Step 6: Calculate the reliability of the current
        product
Step 7: Verify the test for reliability
Step 8: Start deployment
```

Step 1: Determine reliability objectives

The development team has to determine software quality; quality of the software is determined by studying a number of failures & hardness of failures. Characterize the reliability objectives, by determining balance among key qualities of objectives such as reliability, delivery, time, cost, date, &

environment. Tester team also has to determine objectives which are used to manage resources & guide them in design implementation & testing of software

Step 2: Perform system test
The system is configured with the software, hardware & network details. Unit testing & system testing is performed before the reliability test. System capacity and scalability are measured with the workload, where workload characterization is performed for reliability assessment & to identify the number of failures.

Step 3: Gathering of failure data
The development team has to make a collection of a number of failures that occur from the software, hardware & network. Identified collection of such failures forwarded for reliability testing. Before performing reliability test verify and categories collection of failures according to the type of failure. Software reliability parameters should be concentrated while categorizing the number of faults or failures.

Step 4: Identifying relevant reliability measurement model
In reliability measurement process software reliability models are used to measure the quality of software. In this model fault or error detection & fault density of software get identified. There are various models available for reliability test which predicts the software reliability from the test data. These models show a relationship between fault detection data and know mathematical functions such as logarithmic or exponential functions. Choose the appropriate software reliability measurement model among existing models.

- Jelinski-Moranda Model:-This is a continuous time-independent distributed inter failure time and identical error behavior model.
- Goel-Okumoto Model: This is a Nonhomogeneous Poisson process model, which describes the situation that software failure intensity increases slightly at the start and then begins to decrease.
- Musa-Okumoto Model: In this model observation made on the reduction in failure rate resulting from repair action.
- Yamada Delayed S-Shaped Model: This is a modification of the non-homogeneous Poisson process to obtain an S-Shaped curve for the cumulative number of failures detected such that the failure rate initially increases and later decreases.

Step 5: Choose perfect reliability testing tool
Various software reliability testing tools are available for reliability test and growth of reliability. These tools detect and identify the reliability function, failure rate, mean time to failure, median time to failure and the model parameters for each model and help in fault detection and fixing the bugs [10]. Details of available tools are described below:

1. AT&T (AT&T Software Reliability Engineering Toolkit) executes Musa basic and Musa-Okumoto software reliability models and can be applied for both time-domain and interval-domain failure datasets for predicting software reliability.
2. SMERFS (Statistical Modeling and Estimation of Reliability Functions for Software) designed by William Farr (1982) for estimating and predicting the reliability of software during the testing phase and it uses failure count information to make reliability predictions.
3. SRMP (Statistical Reliability and Modeling Programs) developed by the reliability and statistical consultant in 1988. It is a command-line interface tool, where the parameters are estimated using Maximum Likelihood Estimation (MLE) method in order to provide the reliability information such as failure rate, mean time to failure, the median time to failure, etc.
4. Sorel (Software Reliability Program) tool developed by Lab of The National Center for Scientific Research, France in 1991. It has the ability of testing and analyzing the data for making predictions with better accuracy and allows various models to accept both time-domain and interval-domain input data which can be applied for predicting software reliability growth.
5. SARA (Software Assurance Reliability Automation) - Tool incorporates reliability growth model metrics and design code metrics for analyzing the software applicable to time between failure data.
6. CASRE (Computer-Aided Software Reliability Estimation) - developed by Jet Propulsion Laboratories in 1993. It is a user-friendly tool with GUI interface and is capable to customize the user's model using in-build models and can be applied to time-domain failure datasets.

Step 6: Calculate the reliability of the current product
Using the appropriate tool and reliability growth model perform the calculation of the reliability of the current product. Find out the defects or faults and try to recover them. If fault found and achieves the reliability target then forward it to verification & for the deployment of the product [13]. It doesn't get satisfied with the results then continue the processes until the target get achieved.

Step 7: Verify the test for reliability
Through this activity verify that a certain reliability level has been achieved or not and finally reliability can be analyzed in the field to verify the reliability engineering effort & to provide feedback for product & process improvement [14].

Step 8: Start deployment

A reliability model calculates, from failures data collected during system testing such as failure report, data & test time. Finally, the reliability of the product is predicted and ready for use.

These reliability estimates can provide the information useful for product quality management and the reliability of the product at the end of system testing [11]. The amount of additional test time is required to reach the products reliability objective and reliability growth as a result of testing & predicted reliability [12].

V. SOFTWARE RELIABILITY GROWTH MODEL

Software reliability growth model is used for software reliability analysis using the SMERF Tool. SMERFS (Statistical modelling and estimation of reliability functions for software)

This tool is used for reliability analysis. This is a widely used and an accepted software application for evaluation of test data for checking failure rate and fault detection rate forecast. The inputs given to the SMERFS are the collection of values of the time between the detection of defects or the number of defects detected per time period. The model uses the maximum probability method or least squares methods to approximation the parameters used for one or more of these models. This model's output comprises the parameter estimates and a measure of the goodness-of-fit and predicted values using the chi-squared distribution [15].

1. Parameter estimation method, here are two parameter estimation methods maximum-likelihood (ML) and least-squares (LS). Least square requires less computation time to estimate parameters than the maximum likelihood. Where the maximum likelihood will produce valid parameter estimates in the minimum time period
2. Collection of failure data over which the models will be applied
3. Number of stages used in the model for the prediction of future data

The example uses the statistical model Jelinski-Moranda Model to predict the reliability of software. This model makes assumptions about the fault detection correction process, the assumption is listed below:

a. The rate of fault detection is proportional to the current fault content of the program.
b. All failures are equally likely to occur and are independent of each other.
c. Each failure is of the same order of severity as any other failure.
d. The failure rate remains constant over the interval between failure occurrences.
e. The faults are corrected instantaneously without the introduction of new faults into the program.

From these assumptions, the instantaneous failure rate will be explained in

$$Z(t) = \phi[N - (i - 1)] \tag{16}$$

If we represent the time between the *i* 'th and the (*i*-1) th failure by the random variable X*i*, from assumption (d) we can see that X*i* has an exponential distribution, f(X*i*), as shown below:

$$f(X_i) = \emptyset(N - (i - 1))e^{-\emptyset(N-(i-1))X_i} \tag{17}$$

Using assumption (b), the joint density of all of the X*i*'s is:

$$L(X_1, X_2, X_3, \dots \dots, X_n) = \prod_{i-1}^{n} f(X_1) = \prod_{i-1}^{n} f\, \emptyset(N - (i - 1))e^{-\emptyset X_1(N-(i-1))} \tag{18}$$

This is the **Likelihood function**, which can be utilized to compute estimates for the parameters θ̂ a, nd *N* [16].
The partial derivative of the log-likelihood function is taken with respect to each of the two parameters, set these equations to 0 and then solve. It gives the estimated values, θ̂ and *N*, for the model parameters *N* & θ̂:

$$\emptyset\hat{} = \frac{n}{N(\sum_{i=1}^{n} X_1) - \sum_{i=1}^{n}(i-1)_{X_1}} \tag{19}$$

Then it necessary to find the value of *N* ˆ numerically from the following equation, and then utilize it for solving the previous equation for θ̂ :

$$\sum_{i-1}^{n} \frac{1}{N\hat{} - (i-1)} = \frac{n}{N\hat{}\,\frac{1}{\sum_{i=0}^{n} X_i\,[\sum_{i=0}^{n}(i-1)X_i]}} \tag{20}$$

Thus i failures are observed, the predicted time to the next failure, MTBF, is given by:

$$MTBF = \frac{1}{z(t)} = \frac{1}{\emptyset(N-i)} \tag{21}$$

It can also derive a set of least-squares estimators for the model parameters. In this case, it tries to minimize the following quantity, which is the sum of the squared differences between the observed xi and their mean values, the **MTBF**:

$$\sum_{i-1}^{n}(X_i - MTBF)^2 = \sum_{i=0}^{n}\left[X_i \frac{1}{\emptyset(N-(i-1))}\right] \tag{22}$$

As with maximum likelihood estimation, we take the partial derivatives of the equation with respect to each of the model parameters, yielding the two following equations that we then set to 0 and solve:

$$\emptyset = \frac{\sum_{i=1}^{n}\frac{1}{(N-i+1)^2}}{\sum_{i=0}^{n}\frac{X_i}{N\,i+1}} \tag{23}$$

and

$$\left[\sum_{i=1}^{n}\frac{X_i}{(N-i+1)^2}\right]\cdot\left[\sum_{i=1}^{n}\frac{1}{(N-i+1)^2}\right] = \left[\sum_{i=1}^{n}\frac{X_i}{N-i+1}\right]\cdot\left[\sum_{i=1}^{n}\frac{1}{(N-i+1)^3}\right] \tag{24}$$

The resulting estimate for **MTBF** has the same as the obtained from maximum likelihood estimation [17].

Online Business Communication System, The Online Business Communication System project at New Software solutions Software Company. The project consists of one unit-manager, one user interface software engineer and a team of software testers. There are four phases in the SDLC process of the project which can be described as Weeks analysis, Design, Coding, and Testing.

The example uses the statistical model Jelinski-Moranda Model to predict the reliability of software. It uses two parameter estimation methods maximum-likelihood (ML) and least-squares (LS). Least square requires less computation time to estimate parameters than the maximum likelihood. Where the maximum likelihood will produce valid parameter estimates in the minimum time period and the Collection of failure data over which the models will be applied. The number of stages used in the model for the prediction of future data.
Data set: Three fields are considered for fault finding and fault reduction factors shown in table 8.

Table

Data Set	**Fault finding and fault reduction factors**		
	Field 1	***Testing time (in weeks)***	***1 2 3 4 5 6 7 8 9 10 11 12 13 14 15***
	Field2	***Failures***	***2 1 12 3 4 3 5 6 7 8 9 4 3 11 13 14 16 17 12***
	Field3	***Cumulative failures***	***2 3 4 7 9 11 12 13 15 17 19 20 21 23 25 27***

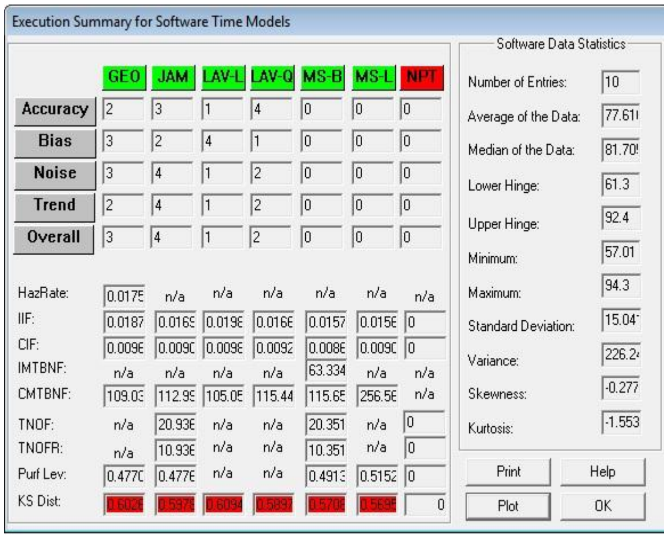
For a period of 10 weeks, the data was collected meanwhile the testing was started and stopped many times. Errors detection is broken down into subcategories to help the development and testing team to sort and solve the most difficult modification requests. These subcategories are referred to as the severity levels depending on the nature of the defect with 3 being the minor problem, with 2 being a major problem and 1 being with the most critical problem.

Table 9 shows the data set maps into the week, consists of three types of errors: seve.1, seve.2, and seve.3 the observation time (OT) in a week and the number of errors detected per week are presented as follows.

Table 9 shows business communication failure data

Data Set	**Observation time (OT) (In weeks)**			
	Time (in weeks)	***Seve.1***	***Seve.2***	***Seve.3***
	1	**4**	**3**	**10**
	2	**1**	**3**	**6**
	3	**2**	**5**	**3**
	4	**3**	**1**	**5**
	5	**1**	**3**	**6**
	6	**0**	**7**	**11**
	7	**1**	**1**	**8**
	8	**0**	**5**	**9**
	9	**2**	**1**	**10**
	10	**1**	**1**	**4**

Table9. Shows the data set maps into the week, consists of three types of errors: seve.1, seve.2, and seve.3 Where, Seve. = Severer problems, OT = Observation time


Execution Summary for Software Time Models

	GEO	JAM	LAV-L	LAV-Q	MS-B	MS-L	NPT
Accuracy	2	3	1	4	0	0	0
Bias	3	2	4	1	0	0	0
Noise	3	4	1	2	0	0	0
Trend	2	4	1	2	0	0	0
Overall	3	4	1	2	0	0	0
HazRate:	0.0175	n/a	n/a	n/a	n/a	n/a	n/a
IIF:	0.0187	0.0169	0.0198	0.0166	0.0157	0.0156	0
CIF:	0.0098	0.0090	0.0098	0.0092	0.0086	0.0090	0
IMTBNF:	n/a	n/a	n/a	n/a	63.334	n/a	n/a
CMTBNF:	109.03	112.99	105.05	115.44	115.65	256.56	n/a
TNOF:	n/a	20.936	n/a	n/a	20.351	n/a	0
TNOFR:	n/a	10.936	n/a	n/a	10.351	n/a	0
Purf Lev:	0.4770	0.4776	n/a	n/a	0.4913	0.5152	0
KS Dist:	0.6026	0.5979	0.6094	0.5897	0.5706	0.5695	0

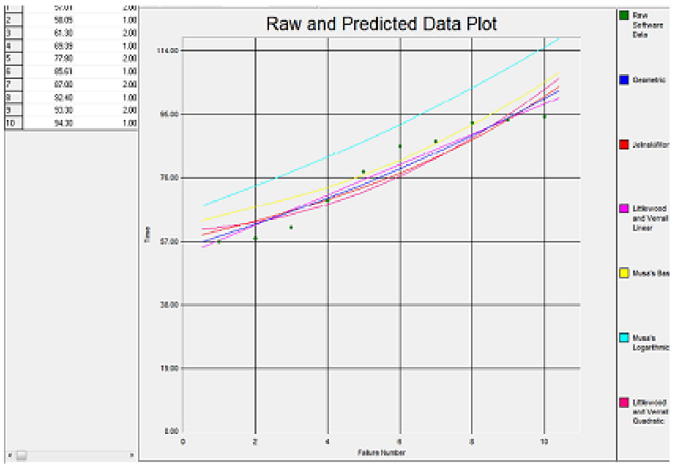
Software Data Statistics

Number of Entries:	10
Average of the Data:	77.61
Median of the Data:	81.70
Lower Hinge:	61.3
Upper Hinge:	92.4
Minimum:	57.01
Maximum:	94.3
Standard Deviation:	15.04
Variance:	226.2
Skewness:	-0.277
Kurtosis:	-1.553

Figure1 shows the statistical for software time models

In the execution summary software data statistics are shown in figure 1with multiple fields such as a number of entries, an average of the data-77.610, a median of the data-81.709, standard deviation- 15.047, variance- 226.27 and skewness- -0.277 and Kurtosis- -1.553. The model applicability analysis defines default ranges as a starting point and ending point and with four different modes such as Accuracy, Bias, Noise, and Trend.

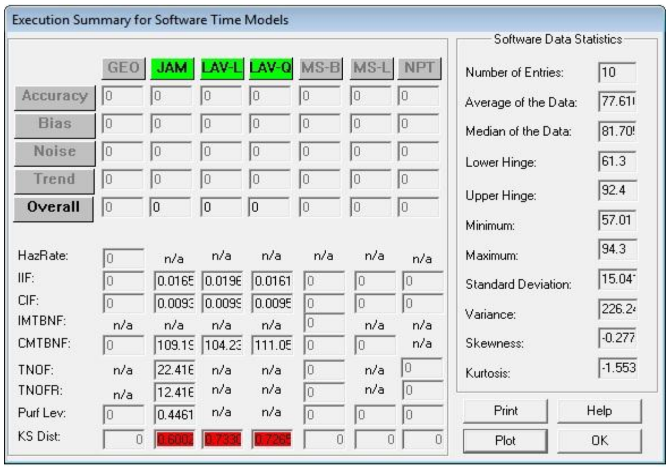
The Littlewood & Verrall Linear model and Littlewood & Verrall quadratic models the computed initial estimates for the maximum likelihood execution are Beta 0 (B0) and Beta 1 (B1). Where B0-50.9993 & B1-4.8393 and for a quadratic model, it is B0-62.0121 & B1-0.4052. The executive summary for software model shows Hazrate-0.0175, CIF-0.0098 and IIF-0.0187. After execution of summary, it is ready to predict the data plot with the number of units and severity levels using all statistical models.



1	57.01	2.00
2	58.05	1.00
3	61.30	2.00
4	69.39	1.00
5	77.80	2.00
6	85.61	1.00
7	87.00	2.00
8	92.40	1.00
9	93.30	2.00
10	94.30	1.00

Figure 2 shows the graphical representation of time and failures

In the above figure, the predicted software data plot shows the time between failure data for software time models. All the models are represented with different colors with a number of failures and time (in weeks). Each statistical model execution using a maximum likelihood method used to plot the number of failures.


Execution Summary for Software Time Models

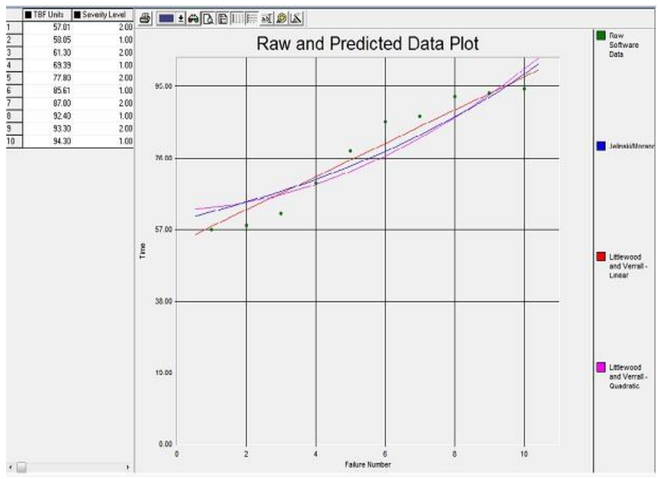
	GEO	JAM	LAV-L	LAV-Q	MS-B	MS-L	NPT
Accuracy	0	0	0	0	0	0	0
Bias	0	0	0	0	0	0	0
Noise	0	0	0	0	0	0	0
Trend	0	0	0	0	0	0	0
Overall	0	0	0	0	0	0	0
HazRate:	0	n/a	n/a	n/a	n/a	n/a	n/a
IIF:	0	0.0165	0.0196	0.0161	0	0	0
CIF:	0	0.0093	0.0099	0.0095	0	0	0
IMTBNF:	n/a	n/a	n/a	n/a	0	n/a	n/a
CMTBNF:	0	109.19	104.23	111.05	0	0	n/a
TNOF:	n/a	22.416	n/a	n/a	0	n/a	0
TNOFR:	n/a	12.416	n/a	n/a	0	n/a	0
Purf Lev:	0	0.4461	n/a	n/a	0	0	0
KS Dist:	0	0.6002	0.7330	0.7265	0	0	0

Software Data Statistics

Number of Entries:	10
Average of the Data:	77.61
Median of the Data:	81.70
Lower Hinge:	61.3
Upper Hinge:	92.4
Minimum:	57.01
Maximum:	94.3
Standard Deviation:	15.04
Variance:	226.2
Skewness:	-0.277
Kurtosis:	-1.553

Figure 3 shows the statistics of number cumulative failures and statistical models using the least square method.

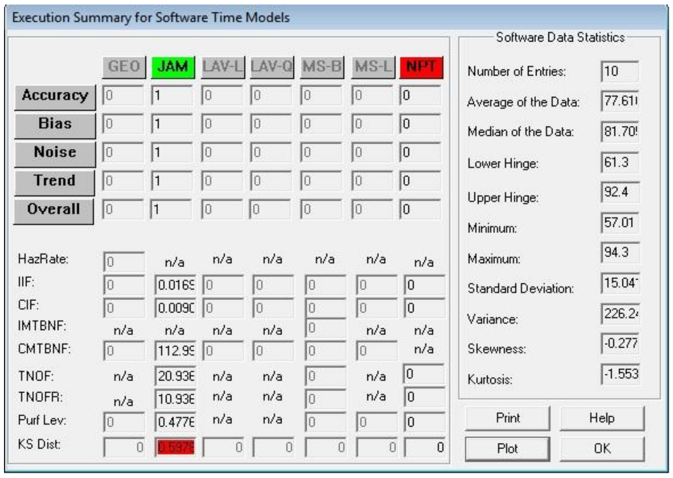
In the above figure, the predicted software data plot shows the number of cumulative failures and the time between failure data for software time models. The least square method used to plot the number of failures in weeks. The software data statistics are presented with a number of entries such as the average of the data-77.610, a median of the data-81.709, standard deviation- 15.047, The statistics also show the variance- 226.27 and skewness- -0.277 and Kurtosis- -1.553.


Raw and Predicted Data Plot
95.00
76.00
57.00
38.00
19.00
0.00
0
2
4
6
8
10
Failure Number
Time
Raw Software Data
JelinskiMoranda
Littlewood and Verrall - Linear
Littlewood and Verrall - Quadratic

	TBF Units	Severity Level
1	57.01	2.00
2	58.05	1.00
3	61.30	2.00
4	69.39	1.00
5	77.80	2.00
6	85.61	1.00
7	87.00	2.00
8	92.40	1.00
9	93.30	2.00
10	94.30	1.00

Figure 4 shows testing results in graphical representation time in weeks and the number of cumulative failures occur

The above figure shows the predicted software data plot using three statistical models JM model, Littlewood &Verrall linear model, and Littlewood & Verrall quadratic model. Each model counts the number of failures in using statistical model formula and represents it. Each model output is shown with separate color with a number of failures and time (in weeks). Here the Least square execution method is used to plot the number of failures.


Execution Summary for Software Time Models

	GEO	JAM	LAV-L	LAV-Q	MS-B	MS-L	NPT
Accuracy	0	1	0	0	0	0	0
Bias	0	1	0	0	0	0	0
Noise	0	1	0	0	0	0	0
Trend	0	1	0	0	0	0	0
Overall	0	1	0	0	0	0	0
HazRate:	0	n/a	n/a	n/a	n/a	n/a	n/a
IIF:	0	0.0169	0	0	0	0	0
CIF:	0	0.0090	0	0	0	0	0
IMTBNF:	n/a	n/a	n/a	n/a	0	n/a	n/a
CMTBNF:	0	112.99	0	0	0	0	n/a
TNOF:	n/a	20.936	n/a	n/a	0	n/a	0
TNOFR:	n/a	10.936	n/a	n/a	0	n/a	0
Purf Lev:	0	0.4776	n/a	n/a	0	0	0
KS Dist:	0	0.5979	0	0	0	0	0

Software Data Statistics

Number of Entries:	10
Average of the Data:	77.61
Median of the Data:	81.70
Lower Hinge:	61.3
Upper Hinge:	92.4
Minimum:	57.01
Maximum:	94.3
Standard Deviation:	15.04
Variance:	226.2
Skewness:	-0.277
Kurtosis:	-1.553

Figure 5 shows statistical analysis using Jelinski-Moranda model

The software reliability growth model Jelinski-Moranda model is used to count the number of failures with maximum likelihood execution method. The above figure shows the executive summary for the JM model.

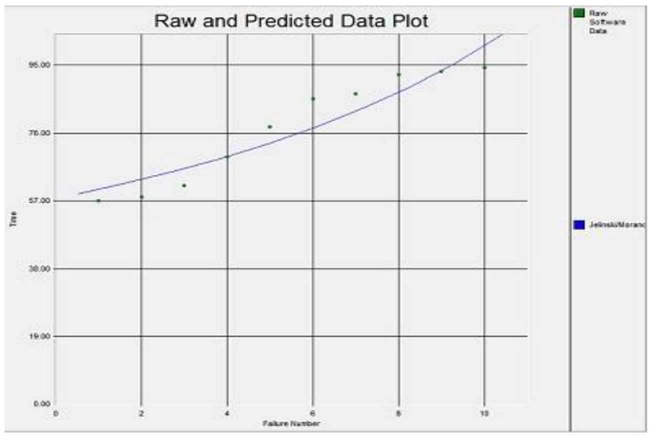


Figure 6 shows the statistical representation using Jelinski-Moranda model

Figure 6 shows the predicted software data plot using the software reliability growth model Jelinski-Moranda model. The number of failures is predicted with time (weeks). The Jelinski- Moranda model is a statistical model used with maximum likelihood execution method.

VI. RESULTS AND DISCUSSION

In this paper, an algorithmic design introduced describes the software reliability testing process for estimating and measuring the reliability of software. The implementation using SMERFs tool specifies that the analytical models are useful in estimating & monitoring. The case study of Business communication system shows the reliability results with fault reduction factor and number of failure counts. It is viewed as a measure of estimation of software reliability & to enhance the quality of software.

VII. CONCLUSION AND FUTURE SCOPE

Software reliability is a measuring technique for identifying the defects that cause software failures in which software behavior is different from the specified behavior in a defined environment with a fixed time**.** In this paper, various software reliability models are reviewed and statistical analysis is performed. The algorithm introduced describes the software reliability testing process for estimating and measuring the reliability of software. It specifies that the analytical models are useful in estimating & monitoring and it is viewed as a measure of estimation of software reliability & to enhance the quality of software.

ACKNOWLEDGMENT

REFERENCES

[1] Sudsanguan Nagamsuriyaroj, Pak Rattidham, “Performance Evaluation of Load balanced Web”, *IEEE 978-6- 7695- 4338 March 2011.*

[2] M. Andreolini, M. Colajanni, and M. Pietri, “A scalable architecture for real-time monitoring of large information systems,” in *IEEE Second symposium on network cloud computing & applications, 2012.*

[3] Guanyin Zhao, Kaigui Wu, Tao Li, “Two-Dimensional Software Reliability Assessment with Multiple Change-points”, *International Journal of Digital Content Technology and its Applications (JDCTA), Volume, DOI:10.4156/jdcta.vol6.issue13.60 No.13, July 2012.*

[4] Manohar Singh, “Software Reliability Testing Tools: An Overview and Comparison”, *International Journal Of Engineering And Computer Science ISSN: 2319-7242 Volume 5 Issue, Page No. 18886-18891, 11 Nov. 2016.*

[5] Shiho Hayashida, Shinji Inoue, and Shigeru Yamada, “Software reliability assessment using exponential-type change-point hazard rate model”, *International Journal of Reliability, Quality and Safety Engineering Vol. 21, No. 4, 1450019 (12 pages), World Scientific Publishing Company, DOI: 0.1142/S0218539314500193 (2014).*

[6] G. Gayathry**,** and R. Thirumalai Selvi, “Classification of Software Reliability Models to Improve the Reliability of Software”, *Indian Journal of Science and Technology, Volume 29, DOI: 10.17485 / v8i29/85287, November 2015.*

[7] Durga Patel, Pallavi, “Software Reliability: Models”, *International Journal of Computer Applications (0975 – 8887) Volume 152 – No.9, October 2016.*

[8] Razeef Mohd, Mohsin Nazir, “Software Reliability Growth Models: Overview and Applications”, *Journal of Emerging Trends in Computing and Information Sciences, VOL. 3, NO. 9, ISSN 2079-8407, SEP 2012.*

[9] Mao Chengyong, LI Qiuying, “A testing-coverage software reliability growth model considering the randomness of the field environment”, *IEEE International Conference on Software Quality, Reliability and Security Companion, 978-1-5090-3713-1/2016.*

[10] Guoxin Su, David S. Rosenblum, “Reliability of Run-Time Quality-of- Service Evaluation using Parametric Model Checking”, *IEEE/ACM 38th IEEE International Conference on Software Engineering, ICSE ’16, Austin, TX, USA, ACM. ISBN 978-1-4503-3900 - May 14-22, 2016.*

[11] M.Ravichandran and A.V. Ramani, “A Methodology for Measuring Web Software Reliability”, *Int. J. on Recent Trends in Engineering and Technology, ACEEE DOI: 01.IJRTET.7.1.70 Vol. 7, No. 1, July 2012*

[12] Heiko Koziolek, Bastian Schlich, Steffen Becker, Michael Hauck, “Performance and reliability prediction for evolving service-orient software systems”, *Empir Software Eng 18:746–790 DOI 10.1007/ s1 0664-012-9213-0-2013.*

[13] Ian Andrusiak and Qusay H. Mahmoud, “A Reliability-Aware Framework for Service-Based Software Development”, *IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE), 978-1-5090-5538-8/2017.*

[14] Suhas Honore, Santanu Kumar Rath, “A Web Service Reliability Prediction using HMM and Fuzzy Logic Models”, *6th International Conference On Advances In Computing & Communications, Cochin, India. ICACC 2016, 6-8, September 2016.*

[15] D.Swamy doss, Dr. Kadhar Nawaz, “Enhanced version of growth model in web-based software reliability engineering”, *Journal of Global Research in Computer Science, Volume 2, No. 12, December 2011.*

[16] Divya Bindal, “Estimating Reliability of Web Application Using Markov Model”, *International Journal of Advanced Research in Computer Science and Software Engineering, IJARCSSE Volume 3, Issue 6, June 2013.*

[17] Md Umar Khan and Dr. T.V. Rao, “Web application’s reliability improvement through architectural patterns*”, International Journal of Web & Semantic Technology (IJWesT) Vol.5, No.3, July 2014.*

[18] Gurpreet Kaur, Kailash Bahl, “Software Reliability, Metrics, Reliability Improvement Using Agile Process*”, IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 3, May 2014.*