©2010 International Journal of Computer Applications (0975 – 8887)
Volume 1 – No. 13

A SOFTWARE RELIABILITY GROWTH MODEL FOR THREE-TIER CLIENT SERVER SYSTEM

Pradeep Kumar
Information Technology Department
ABES Engineering College, Ghaziabad
Affiliated to UPTU Lucknow, India

Yogesh Singh
Professor, University School of IT
Guru Gobind Singh Indraprastha University
Delhi – 110006, India,

ABSTRACT

With the ever-increasing role that software is playing in our real-life systems, concern has steadily grown over the quality of the software products. In today's life the computers are being used to monitor and control safety critical and civilian systems with a great demand for high-quality software products. So reliability is a primary concern for both software developers and software users. In literature many software reliability growth models have been proposed over the years to estimate and predict reliability of software products. But it is often very difficult for project managers and practitioners to determine which model is more useful in a particular domain and up to what extent. In this paper we propose a NHPP based software reliability growth model for three-tier client server systems. The present model composed of three layers of client-server architecture related to presentation logic, business logic and database stored at backend. Presentation layer contains forms or server pages which presents the user interface for the application, displays the data, collects the user inputs and sends the requests to next layer. Business layer, which provides the support services to receive the requests for data from user tier, evaluates against business rules, passes them to the data tier and incorporates the business rules for the application. Data layer includes data access logic, database driver(s), query engines used for communicating directly with the data store of a database. The model has been validated through standard dataset consists of software failure data on various projects released from the software reliability dataset and applying to a live commercial application.

Categories and Subject Descriptors

Software reliability engineering, client-server models, distributed applications, software metrics, nonhomogeneous Poisson process, failure rate.

General Terms

Reliability, Measurement, Performance, Experimentation

Keywords

Application server, database server, presentation layer, reliability growth factor

1. INTRODUCTION

The present scenario of software development life cycle has emerged into a distributed environment because of the development of network technology & ever increased demand of sharing the resources to optimize the cost. Therefore to improve the process of reliability estimation and prediction of software products we identify and remove the remaining faults during the testing phase in a three-tier client server based systems. Reliability can be grown through various means such as improving the process of designing, effectiveness of testing, manual & automated inspections, familiarization with developers, users & product, and improving the management processes & decisions [1, 2]. The rate at which reliability grows depends on the factors related to how rapidly defects are discovered, how fast corrective action can be identified and implemented & how soon the impact of the changes take place and make operational in the field. In three-tier client server architecture the presentation logic and business logic are split off into separate components resulting into three-tier system shown as in figure 1.

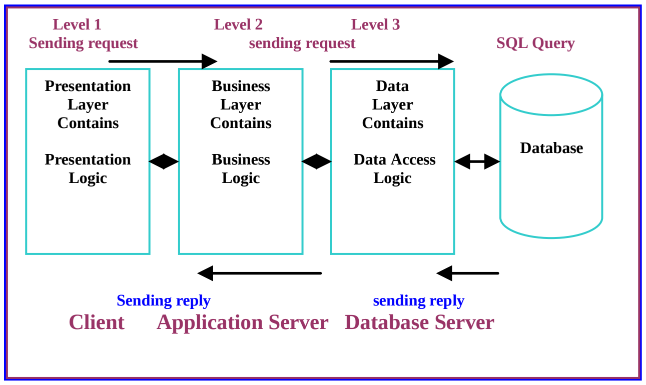

Figure 1. A three-tier client-server architecture view

2. SRGM SPECIFICATION

In a multi node client-server system consisting of various components of software that execute on different nodes it becomes almost mandatory to model the system in such a client-server computing environment if realistic reliability prediction and assessment are to be made. Also in three-tier architecture when there are number of clients and number of servers in a client-server system, it is not always necessarily the case that a software failure in any of the clients or servers will cause the system to fail. There are various factors related to the failure of a system such as transmission failure, networking failure, database-linking failure, query engine failure including software development life cycle (SDLC) failure [4,5]. To address some of these vital issues related to software failure we decompose the present model into three different layers and discuss each layer to identify the causes of errors, level of severity and its impact to improve the reliability of the software during the testing phase. Finally we compute the failure intensity function, probability distribution function, cumulative distribution function, mean time to failure, and reliability of the system as a whole using a real life software reliability dataset [6,7]. The present model facilitates project managers and the practitioners to assess the reliability of a software system based on the amount of efforts put in testing, how accurately parameters are estimated, how efficiently the relevant & updated failure data of modern computer system is collected and to the possible extent the model has been validated using current real life software. This model further can be used to determine the quality of development processes in terms of the number of remaining faults, mean time to failure, time between failure, next expected failure and failure intensity of the software at the beginning of a system test.

Table 1. Causes of Errors at Different Layer of the Model

Model Layers	Possible Causes of Error(s)
Presentation layer	Invalid input(s), non-formatted data such as entering characters in place of a non negative integer value, User authentication and authorization error such as invalid login or password and Lack of security measures such as damaging & mishandling of the system
Business Layer	Logical error such as business logic is not being coded as per the software requirement specifications, Exceptions are not being handled properly, Less tolerance power (degree to which handle the unexpected behavior of the system) and Security measures such as poor encryption / decryption algorithm(s)
Database Layer	Non homogeneous data formats, database connectivity error or intermittent connectivity, ODBC driver failure, query engine failure to execute the query or large amount of data to process and retrieve, availability of low bandwidth to fetch the data, network congestion and security measures such as fire, floods, earthquake or any other mishap.

The main advantage of three-tier client server SRGM is that all business logic has been centralized in one layer. A component in the business layer can be accessed by any number of components in the presentation layer, therefore any changes to business logic can be made in one place and be automatically inherited by all other components without having to duplicate the change in those other components. Also the presentation layer components do not access the database all data is provided by the business layer in the form of XML streams. Any changes made in the presentation layer need to be passed back to the business layer before they can be applied to the database.

2.1 Severity of Errors

We categorize the severity level of error(s) during the execution & operation of present model as follows:

Catastrophic: The system failures may cause to loss of life or heavy damage to the system wherever it is installed.

Gradual: The severity level of this kind of error(s), which may further be critical, marginal or negligible depending upon the kind of application and operational environment.

Critical: may cause complete loss of system such as disaster and applicable to all three layers presentation, application and database of the model.

Marginal: may degrade the system gradually such as infected by viruses, worms or network congestion and heavy load of data to be processed.

Negligible: may lead to minor failure of the system and applicable to the presentation & database layer such as incorrect username & password, invalid user's input, database not found or does not exist, ODBC driver failure or rebooting the system in worst case.

Terminology

Node: A hardware element on a network generally a computer \PC \desktop\ laptop that is installed with a NIC card.

Client: A node that makes request of services in a network or that uses resources available through the servers.

Server: A node that provides some type of services to the clients such as network resources/ files or distributed services.

Client-Server computing: defined as processing capability or available information distributed across multiple nodes.

Software Defect: Any undesirable deviation in operation of the software from its intended operation, as defined in the software requirement specifications.

Errors: are human actions that result in the software containing a fault. Examples of such faults are the omission or misinterpretation of the user's requirements, a coding error etc.

Faults: are manifestations of an error in the software. If encountered then it may cause a failure of the software.

Failure: is the inability of the software to perform its mission for function within specified limits. Failures are observed during testing and operation.

Failure rate: refers to the rate of occurrence of Failure (ROCOF) depending upon the context. The ROCOF is the unconditional rate of occurrence of a failure at a point in time.

Software failure: a failure caused by a software fault. It is to be noticed that software itself does not fail. Faults already present in the software lead to failure of the system under certain conditions.

NHPP: The non-homogeneous Poisson process model (NHPP) represents the number of failures experienced up to time *t* is a non-homogeneous Poisson process $\{N(t), t \geq 0\}$. The NHPP based model provides an analytical framework for describing the software failure phenomenon during testing. The main issue in the NHPP model is to estimate the mean value function of the cumulative number of failures experienced up to a certain point in time.

Assumptions:

- The software failure-occurrence phenomenon is described by an NHPP.

- The software faults detected during the testing phase are corrected certainly and completely, that is no new faults are introduced into the software systems during the debugging phase. On a failure observation an immediate effort takes place to locate the causes of the failure & the error removal takes very small amount of time, which is nearly negligible.
- Software is subject to failures during execution caused by faults remaining in the software.
- The software is developed for three-tier client server based systems.
- A finite number of test cases are prepared to ensure that the software works according to the requirements and specifications. Each test case is designed to execute a finite number of instructions.
- The error removal intensity per execution is proportional to the remaining errors in the software at any point of time.

Notations:

a	total number of errors in the software
$N(t)$	number of errors corrected up to time t
$m(t)$	the mean value function or expected no. of faults detected or removed by time t
b_1	error correction rate during the initial testing phase of presentation layer
b_2	error correction rate during the testing phase of business layer
b_3	error correction rate during the final testing phase of database layer
r_1	error generation factor due to correction of errors in initial testing phase of presentation layer
r_2	error generation factor due to correction of errors in testing phase of business layer
r_3	error generation factor due to correction of errors in testing phase of database layer
t_1	time spent in initial testing phase at presentation layer
t_2	time spent in testing of business layer
t_3	time spent in testing at database layer
t	total time spent in all the three phases of testing
$\lambda(t)$	intensity function for NHPP models or fault detection rate per unit time
T_k	software life cycle length
$R(t)$	reliability of the software developed
$F(t)$	cumulative distribution function (cdf)
$f(t)$	probability distribution function (pdf)
MTTF	mean time to failure

3. MATHEMATICAL MODEL

We consider a software in which failures are caused by software errors. Let $\{N(t), t \geq 0\}$ be the total number of errors corrected up to time t during the total testing phase. A stochastic process $\{N(t), t \geq 0\}$ is a non–negative process where N(t) is a random variable which represents the cumulative no of faults detected up to a testing time t. The fault detection process is described by NHPP with the mean value function m(t) as follows:

$$\Pr\{N(t) = n\} = \frac{\{m(t)\}^n \exp[-m(t)]\}}{n!}$$

where n=0, 1, 2…

$$m(t) = \int_0^t \lambda(x)\,dx \qquad (1)$$

where Pr{N(t)} denotes the probability of event N(t) and m(t) is the mean value function, which represents the expected cumulative no. of faults detected in the testing time interval (0,t] and λ(t) is an intensity function which represents the fault-detection rate per fault. The NHPP model is characterized by its mean value function defined as follows:

$$m(t) = a(1 - e^{-bt}) \qquad a>0,\ b>0 \qquad (2)$$

where a, is the expected no of initial inherent fault before testing and b is the software failure occurrence rate per inherent fault.In three-tier client server based model there are three type of faults and some faults are easier to detect then others based upon the efforts required to detect the cause of failure in order to fix and remove it. In the present model these faults are associated with presentation layer, business layer and database layer during the total testing phases. Also we consider that error correction rate and error generation factor is different for both these phases, i.e. during the initial testing phase more errors are likely to occur which consequently decreases as the testing progresses. During the process of error correction at presentation layer, a few errors may be generated at business layer and database layer, which will affect the total performance of the system. Thus m(t) for the proposed model can be written as:

$$m(t) = a\sum_{i=1}^{3}(1 - \exp[-b_i t_i])*(1 - r_i) \qquad (3)$$

where $t_1 + t_2 + t_3 \leq t$, $a > 0$,
$0 < b_3 < b_2 < b_1 < 1$, $0 < r_i < 1$

For three types of fault at each layer the intensity function can be written as dm(t) / dt that is

$$\lambda(t) = a\sum_{i=1}^{3}\{b_i \exp[-b_i t_i] - r_i \exp[-b_i t_i] b_i\}$$

$$= a\sum_{i=1}^{3} b_i \exp[-b_i t_i](1 - r_i) \qquad (4)$$

This is the instantaneous error detection rate, i.e. the expected number of detected errors per unit time at time t. Also we can derive the expressions for various software reliability assessment measures from this new model given by eq. (3).

The expected no. of faults remaining at the system testing time t which is obtained by taking expectations of random variables $\{N(\infty) - N(t)\}$i.e.

$$n(t) = E[N(\infty) - N(t)] \qquad (5)$$

The error detection rate per error (per unit time) at time t is defined by dp(t) as follows:

$$dp(t) = \frac{\lambda(t)}{[a - m(t)]}$$

$$= \frac{a\sum_{i=1}^{3}(1 - \exp[-b_i t_i](1 - r_i)}{a - a\sum_{i=1}^{3}(1 - \exp[-b_i t_i](1 - r_i)}$$

$$= \frac{a \sum_{i=1}^{3} b_i \exp[-b_i t_i](1-r_i)}{r_i + \exp[-b_i t_i] - r_i \exp[-b_i t_i]} \quad (6)$$

Applying the boundary conditions when t=0 and t=∞ we get

$$dp(0) = \sum_{i=1}^{3} b_i (1-r_i) \text{ and } dp(\infty)=0 \quad (7)$$

The expected no. of errors remaining in the software at time t is given by N(t)=a – m(t) i.e.,

$$N(t)= a \sum_{i=1}^{3} [\ (1-r_i)\exp(-b_i t_i) + r_i\ ] \quad (8)$$

The probability that a software failure does not occur during (s, s + x), given that the last occurrence time of a software failure was s, is given by

$$R(x/s)=\exp(-a\sum_{j=1}^{3}[\{\exp[-b_i s]-\exp[-b_i (s+x)]\}((1-r_i) + r_i]) \quad (9)$$

The conditional probability function $R_p(x/s)$ is known as software reliability of NHPP model with m(t). The mean value function m (t) represents the number of errors actually corrected.

4. DATA COLLECTION

The sanctity of collected failure data depends on how accurately & efficiently we observe failure data from real life software products of modern computer systems which is very complex procedure and that need to be addressed further separately for better validation of the model by the community of researchers and practitioners. In this paper we have taken software failure data on various projects from the Software Life Cycle Empirical/Experience Database (SLED) published by Data & Analysis Center for Software (DACS). Further to validate our model for estimating reliability growth of three-tier client server system we have applied the model to the data set of On-line Data Entry Software Package test data (Obha 1984a) and Real-Time Control Systems (Hou et al., 1997) assuming that the no. of failures-detection data set is observed from the system-testing phase after confirmation of the integration of all modules\ software components. The observation of failure and repair times can be represented by $t_1, t_2, \ldots, t_n$ where t_i represents the time of failure of ith unit. It is assumed that each failure represents an independent sample from the same population. The population is the distribution of all possible failure times and may be represented by f(t), R(t), F(t) or λ(t). Therefore the basic problem reduces to determine the best failure distribution implied by the n failure times comprised in the sample. In all cases the sample is assumed to be a simple random or probability sample. A simple random sample is one in which the failure or repair times are independent observations from a common population. If f(t) is the probability density function of the underlying population then $f(t_i)$ is the probability density function of the ith sample value. Since the sample comprises of n independent values therefore the joint probability distribution of the sample is the product of n identical and independent distributions i.e.

$$f_{t_1,t_2 \ldots t_n}(t_1,t_2 \ldots t_n)=f(t_1)f(t_2).,f(t_n) \quad (10)$$

Table 2. Failure Datasets applied to the model

S.No.	Project Description	Number of Failures	Source #
1	Real Time Command & Control	136	DACS
2	Real Time Command & Control	54	DACS
3	Real Time Command & Control	58	DACS
4	Real Time Command & Control	53	DACS
5	Commercial Subsystem	73	DACS
6	On-line Data Entry Software Package	46	Obha 1984
7	Real-Time Control Systems	481	(Hou et al., 1997)

4.1 Method of Parameter Estimation

The value of six unknown parameters of the proposed model given in equations (3) and (4) are obtained by the method of Maximum Likelihood Estimation (MLE). Let X be the discrete variable representing the no. of trials necessary to obtain the first failure. Here we assume that the probability of failure remains a constant p and each trial is independent then

$$\Pr\{X = x\} = f(x) = (1-p)^{x-1}.p \quad (11)$$

where x=1,2,….

and which is the probability of (x-1) successes i.e. probability $=(1-p)^{x-1}$ followed by a failure probability (probability = p).If $x_1, x_2, \ldots, x_n$ represents a sample of size n from this distribution then from equation (10) the joint distribution may be written as:

$$f_{x_1, x_2 \ldots x_n}(x_1, x_2 \ldots x_n) = f(x_1)f(x_2).,f(x_n).$$

$$=(1-p)^{x1-1}.p(1-p)^{x2-1}.p\,(1-p)^{x3-1}.p\ldots,(1-p)^{xn-1}.p$$

$$=p^n.(1-p)\exp[\sum_{i=1}^{n}(x_i-1)] \quad (12)$$

Equation (12) is called likelihood function and represents the probability of obtaining the observed sample. Since equation (12) contains the unknown parameter p we find a value of p consistent with the observed sample. If a value of p is found that maximize the likelihood function then it also maximize the probability of obtaining the observed sample.

$$\max g(p) = p^n.(1-p)\exp[\sum_{i=1}^{n}(x_i-1)]$$

for 0<=p<=1

Therefore we solve this equation to get maximum of a function by finding the point at which the first derivative is equal to zero as follows:

$$\max \log g(p) = \log[\, p^n.(1-p) \quad \exp[\sum_{i=1}^{n} (x_i - 1)]]$$

$$= n \log p + \sum_{i=1}^{n} (x_i - 1) \log(1 - p) \qquad (13)$$

Now putting first derivative of max log g(p) = 0 we get i.e.

$$d/dp\, [\max \log g(p)] = d/dp\, [\, n \log p + \sum_{i=1}^{n} \log (1 - p)\,] = 0$$

$$n / p + \sum_{i=1}^{n} (x_i - 1) (-1) / (1-p) = 0$$

$$n / p (1 - p) = \sum_{i=1}^{n} (x_i - 1)$$

$$\max (p) = n / \sum_{i=1}^{n} x_i \qquad (14)$$

where max (p) is defined as the Maximum Likelihood Estimator of the given distribution.

4.2 Model Validation

Based on the data available given in table (2) the performance analysis of the proposed model is measured by the four common criteria SSE as the sum of squared errors, R-square, Adjust R-square & RMSE for the model comparison of goodness of-fit as follows:

Sum of square of Error (SSE): This statistic measures the deviation of the responses from the values of responses. A value closer to 0 indicates a better estimation. It is calculated as:

$$SSE = \sum_{j=1}^{k} \sum_{i=1}^{n} [\, y_{ij} - m_j(t_i)]^2 \qquad (15)$$

where y_{ij} is total number of type j failures observed at time t_i according to the actual data $m_j(t_i)$,the estimated cumulative number of type j failures at time t_i for i =1,2,…,n and j =1,2,…, k.

Mean Square of fitting Error (MSE): It is calculated as:

$$MSE = \frac{\sum_{i=1}^{n} [\, m_j(t_i) - y_{ij}]^2}{n} \qquad (16)$$

where $y_{ij}(m_j(t_i))$ is the actual estimated value of the total number of errors removed in interval (0, t]. The MSE measures the distance of a model estimate from the actual data with the consideration of the number of observations and the number of parameters (*N*) in the model.

RMSE – is defined as the root of mean squared error and for a computed value closer to 0 it indicates a better approximation & estimation.

That is,

$$RMSE = \sqrt{MSE} \qquad (17)$$

R-square: This statistic measures how successful the model is in explaining the variation of the data, which may be defined as the square of the correlation between the response values and the predicted response values. It is also called the square of the multiple correlation coefficients and the coefficient of multiple determinations. R-square can take on any value between 0 and 1, with a value closer to 1 indicating a better estimation of the model. For example if R-square = 0.8234 means that the estimation explains 82.34% of the total variation in the data about the average.

Adjusted R-Square: The degrees of freedom uses the R-square statistic and adjusts it based on the residual degrees of freedom. The residual degree of freedom is defined as the number of response values n minus the number of fitted coefficients m estimated from the response values.

$$v = n-m \qquad (18)$$

where v indicates the number of independent pieces of information involving the n data points that are required to calculate the sum of squares. A value closer to 0 indicates a better estimation of the model.

5. RESULT ANALYSIS

In this section we show the result of our model applied to a set of failure data extracted from various projects listed in table2. Figure (2) to figure (12) exhibits the result of various computed quality attributes using equations (3) and (4) such as failure intensity λ(t), reliability of the software at any instance of time during testing phase R(t), cumulative distribution function (CDF), probability distribution function (PDF), mean time to failure (MTTF) & variance factor. Here we have modeled the daily defect arrival data during the testing phase of system based on the cumulative failures, length of failure interval and the day of failure it was reported whereas tracking of the data for software reliability estimation has been done on a calendar-time basis and the testing effort is homogeneous throughout the testing phase. We have simulated the seven failure datasets taken as one-dimensional data with the help of non-linear fitting functions using Matlab 7.0.1 under Windows XP environment.

Table 3. Goodness of fitness for different projects

Goodness of fitness criteria	SSE	R_ Square	Adj. R-Square	RMSE
Project 1	0.04451	0.9754	0.9703	0.03423
Project 2	0.00744	0.4824	0.2237	0.02158
Project 3	0.00008	0.9997	0.9995	0.00298
Project 4	0.00002	0.9999	0.9998	0.00147
Project 5	0.20080	0.5353	0.3495	0.03168
Project 6	0.10920	0.5693	0.2822	0.09539
Project 7	0.34290	0.8517	0.8401	0.18330

5. 1 OBSERVATIONS

Typically software reliability growth model estimate the time to next failure or the expected number of remaining failures or when to stop the testing and release the product to the customer. Time is measured in terms of test time including CPU execution time, lines of code tested, system operating time as a calander time i.e. the duration of testing such as no. of hours \days\weeks & months.As a result the probabilistic models are used in describing software reliability and normally a decreasing failure rate is observed if software failures are fixed as they occur and the fix does not generate any new failures. Thus software testing can be likened to reliability growth testing in which the software is executed in an attempt to discover failure, analyze the causes of failure mechanism and initiate the corrective measures. Following are the observations made from applying the model on seven projects listed in table (2) and table (3). The different reliability attributes computed using datasets of project (6) and (7) are shown in figures (9) to figure (13) with significant and improved results. The present model exhibits constant failure rates and the exponential distribution in many respects, which is the simplest reliability distribution to analyze and reveals from the observations that if the failure rates of all failure modes of a component are constant & independent then the overall failure rate of the component is also constant. There are several interesting physical processes that give rise to the cause why have we chosen exponential probability distribution for implementing our model. A constant failure rate implies completely random and independent failures over time and hence results in lack of memory. In fact these three characteristics related to randomness, constant failure rates and memorylessness more or less exhibit different form of same phenomenon.

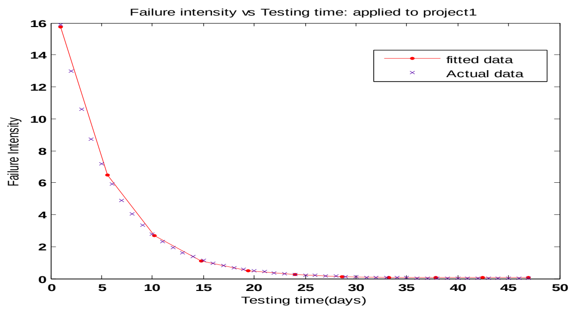

Figure 2. Failure intensity vs. testing time

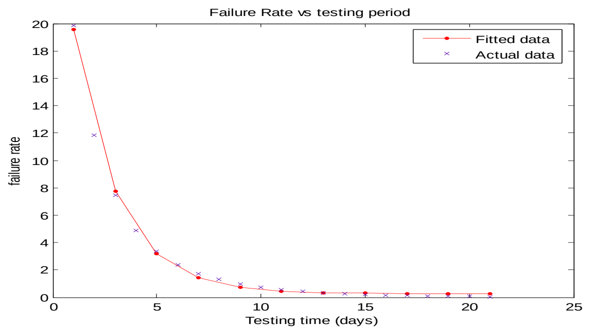

Figure 3. Failure intensity vs. testing time

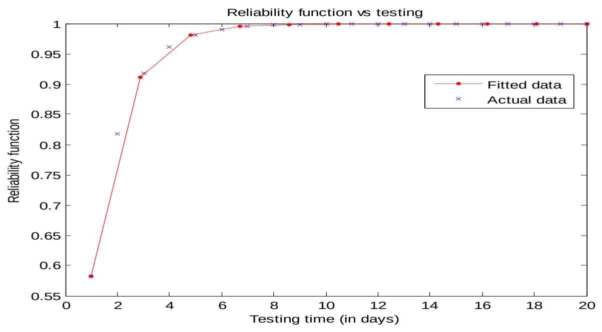

Figure 4. Reliability function vs. testing time

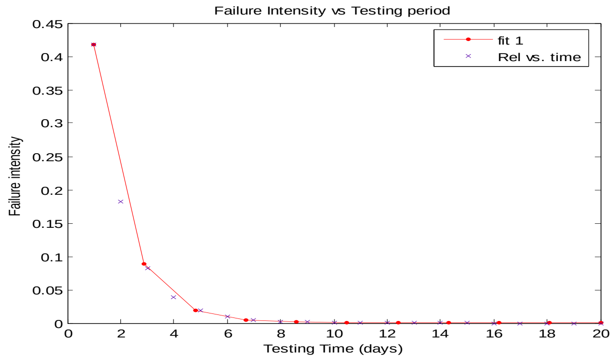

Figure 5. Failure intensity vs. testing time

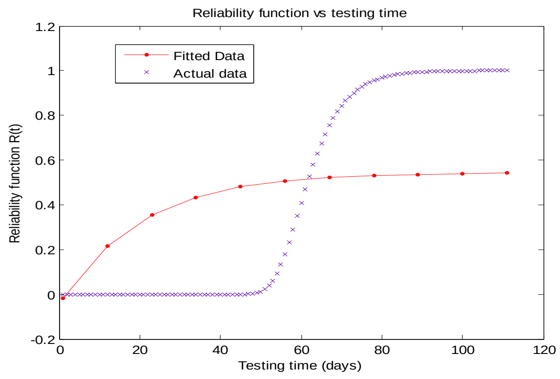

Figure 6. Reliability function vs. testing time

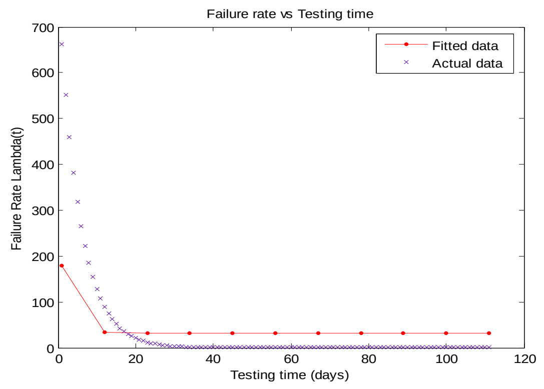

Figure 7. Failure intensity vs. testing time

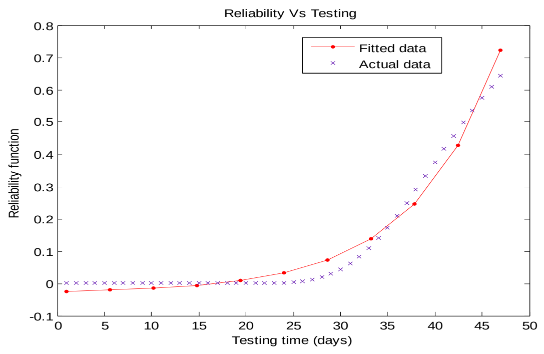

Figure 8. Reliability function vs. testing time

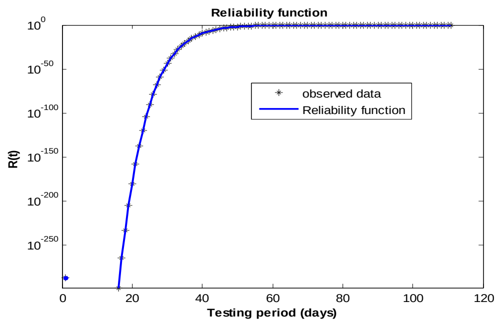

Figure 9. Reliability function vs. testing time

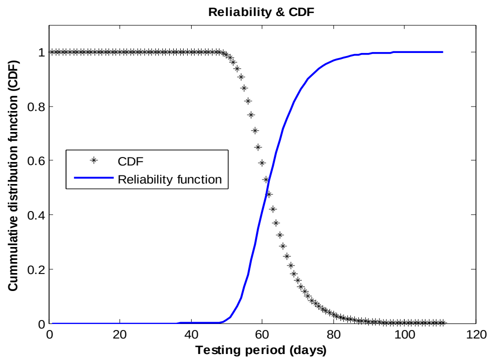

Figure 10. Reliability & CDF vs. testing time

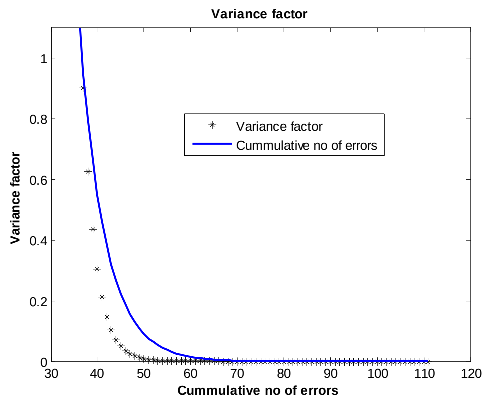

Figure 11. Cumulative errors vs. Variance factor

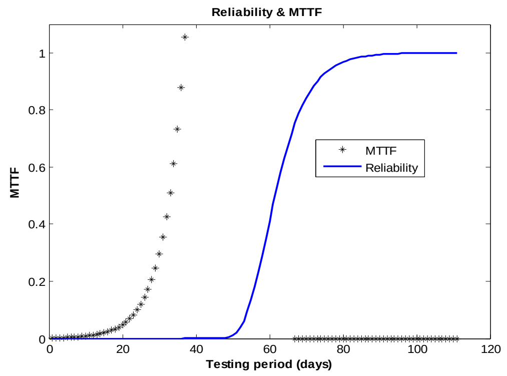

Figure 12. Cumulative distribution function vs. testing time

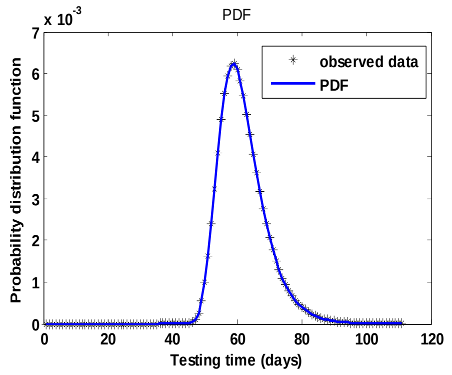

Figure 13. Probability distribution function vs. testing time

6. CONCLUSION & FUTURE WORK

Based on the above approach it seems to be quite feasible to develop such a software reliability growth model for a three-tier client-server system. However, in order to implement the present model it is necessary to partition the failures and defects into three categories associated with each presentation, application & database layer of the present model. In this paper we have designed a software reliability growth model for three-tier client-server system based on nonhomogeneous Poisson process, which incorporates the exponential software reliability growth model for estimation and prediction of software reliability. We have discussed various aspect related to the severity level of errors and its impact on the respective layer of the proposed model. The model also has been validated using failure data of seven real life datasets of various projects released by software reliability dataset DACS. Further if we are able to estimate the values of the parameters more precisely then we can enhance software reliability assessment measures more accurately with the help of our model in comparison with the conventional existing models.

However we have assumed a perfect debugging environment to validate and implement the present model, which may not be realistic in many real life development processes that is the removal of all software error(s) or faults is performed perfectly at each particular layer of the model during the testing phase. Therefore to overcome this kind of deficiency we need to collect more realistic data little bit more precisely from real life projects released under the imperfect debugging environment of modern computer systems with the possibility of introducing new faults at different layers of the model. Since the software testing consumes a large amount of efforts required to locate and fix the error during the testing phase of a software system, which consequently increase the allocated budget for the development of the system. Therefore, in the future it is very much essential and required to develop a mechanism of when to stop the testing process and release the products to the end user with higher quality, within budget and without any delay.

REFERENCES

[1] A Software Reliability Growth Model for a Distributed Development Environment Electronics and Communications in Japan, Part 3, Vol. 83. No. 12, 2000, Shigeru Yamada, Yoshinobu Tamura and Mitsuhiro Kimura.

[2] Determination of software release instant using a nonhomogeneous error detection rate model Microelectron Reliability, Vol. 33. No. 6. pp. 803-807, 1993, printed in Great Britain, K.K. Aggarwal and Yogesh Singh.

[3] Software Reliability Engineering: more reliable software faster and cheaper second edition published by TMH publications 2007, Musa J D.

[4] Software reliability model for modular structure IEEE Transactions on Reliability, R-28, No. 1979, Littlewood B.

[5] Topics in safety, reliability and quality Reliability Engineering published by Kluwer publications 1993, K.K. Aggarwal.

[6] Software reliability modeling published by World Scientific publications 1991, Min Xie.

[7] System Software Reliability published by Springer Series in Reliability Engineering 2006, Hoang Pham.

[8] Handbook of Software reliability engineering edited and published by IEEE computer society press and TMH publications 2007, Michael R Lyu.

[9] Operational profile in software reliability engineering IEEE software 1993, Musa J D.

[10] Software Reliability Engineering for Client-Server Systems Proceedings of the Seventh International Symposium on Software Reliability Engineering (ISSRE '96), 1071-9458/96, 1996 IEEE, Norman F Schneidewind.

[11] An Architecture-Based Software Reliability Model Computer Science Department, SUNY Albany 2000, Wen-Li Wang, Ye Wu, Mei-Hwa Chen.

[12] Software Engineering: programs, documentation & operating Procedures published by New Age International publications 2007, K.K. Aggarwal and Yogesh Singh.

[13] Post-Release reliability Growth in Software Products ACM Transactions on Software engineering and Methodology, Vol. 17, No.4, Article 17, pub. Date: August 2008, Pankaj Jalote, B Murphy, Vibhu Saujanya Sharma.

[14] Contributions to Hardware & Software Reliability published by World Scientific publications 1999, P K Kapur, R B Garg, S K Kumar.

[15] Software Reliability Carnegie Mellon University 18-849b Dependable Embedded Systems Spring 1999 Authors: Jiantao Pan ,jpan@cmu.edu , Jiantao Pan.

[16] Probability and Statistics with Reliability, Queuing and Computer Science Applications, second edition published by John-Wiley publications 2007, Kishore S Trivedi.

[17] Software Metrics and Reliability *Software Reliability Engineering the 9th International Symposium, 1998, Germany,* Rosenberg, L., Hammer, T., Jack S.

[18] Metrics and Models in Software Quality Engineering published by Pearson education 2008, Stephan H Kan.

[19] Reliability and maintainability engineering published by TMH publications by Charles E. Ebeling 2007.

[20] An Assessment of Testing-Effort Dependent Software Reliability Growth Model, IEEE Transactions on Reliability, Vol, 56,No,2, June 2007 by Chin-Yu Huang, Sy-Yen Kuo, Michel R. Lyu.