Measurement of Software Reliability Using Sequential Bayesian Technique

Lalji Prasad*, Ankur Gupta**, Sarita Badoria***

Abstract— This paper presents a new model sequential Bayesian technique for software reliability characterization using a growth curve formulation that allows model parameters to vary as a function of covariate information. The approaches include probabilistic models that aim at predicting reliability and other elements of software quality on the basis of program properties such as size and complexity, and statistical models that base reliability prediction on an analysis of failure data.

We describe a Sequential Bayesian Technique and model evaluation which allows for integration of historical information and expert opinion in the form of prior distributions on the parameters.

***Index Terms*— sequential Bayesian technique probabilistic models, predicting reliability, operational profile**

I. INTRODUCTION

Software reliability engineering is centered around a very important software attribute: reliability. Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment[8]. It is one of the attributes of software quality, a multi-dimensional property including other customer satisfaction factors like functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. Software reliability, however, is generally accepted as the key factor in software quality since it quantifies software failures - which can make a powerful system inoperative or even deadly. As a result, reliability is an essential ingredient in customer satisfaction for most commercial companies and governmental organizations.

The main objective of this paper is to determine the Reliability of Software [16][17]. Requirements Specification defines and describes the operations, interfaces, performance, and quality assurance requirements of the Reliability of Software. The document describes the design constraints that are to be considered when the system is to be designed, and other factors necessary to provide a complete and comprehensive description of the requirements for the software. This paper attempts to focus on analysis of reliability of software using Sequential Bayesian Technique [1][10][13][14][18].

*Lalji Prasad is with Sanghvi Institute of Management & Science /Computer Engineering, INDORE (Email: lalji_prasad@sims-indore.com)

**Ankur Gupta is with Sanghvi Institute of Management & Science /Computer Engineering, INDORE (Email: ankur.gupta@sims-indore.com)

***Sarita Badoria is with MITS, GWALIAR, INDIA (Email: sarita-mits@gmail.com)

II. SOFTWARE RELIABILITY

Reliability of software systems requires implementation of a thorough, integrated set of reliability modeling, allocation, prediction, estimation and test tasks. These tasks allow on-going evaluation of the reliability of system, subsystem and lower-tier designs. The results of these analyses are used to assess the relative merit of competing design alternatives, to evaluate the reliability progress of the design program, and to measure the final, achieved product reliability through testing. As such the Software reliability model[1][3][4][5][6][11][14] in **Figure 1** represents the whole framework being done for analysis

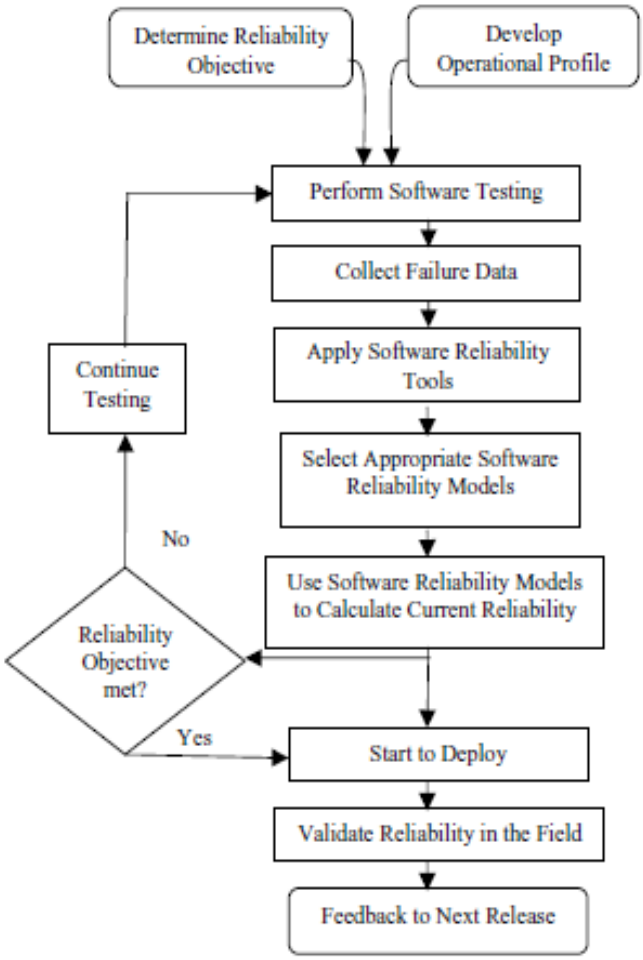

Figure 1. Software Reliability Model

The Reliability of software provides real time information about the software that how much it is reliable on client needs & how much he can be dependable on it. The Product functions are more or less the same as described in the product perspective.

III. RELIABILITY TOPOLOGY

Reliability topology[15] is the relationship between the failures of an individual function to the failure of the aggregate system. Generally, software functions or operations are related in a "series" topology, meaning that the failure of one function results in the failure of the software system. Software fault tolerance techniques can result in systems that can survive the failure of one or more functions. Software fault tolerance consists of a set of techniques which are not covered by this notebook, but are described in depth in the general software engineering literature.

IV. TYPES OF FAULTS

The following types of faults are being found while determining the reliability of software.

Activity introducing fault	**Fault type or root cause**
Requirements	Missing requirements. Misinterpreted requirements. Requirements not clear. Changed requirements. Conflicting requirements.
Design	Design not to requirements. Missing design. Top level design logic. Low level design logic. Design not robust.
Code	Code not implemented to design. Code not implemented to requirements. Missing code. Initialization error. Storing error. Mismatched parameters. Math operations not robust. I/O operations not robust. Memory errors. Domain errors.
Maintenance and corrective action	New fault generated in maintenance.

V. OPERATIONAL PROFILES

The reliability of a software-based product depends on how the computer and other external elements will use it. Making a good reliability estimate depends on testing the product as if it were in the field. The *operational profile* (OP), a quantitative characterization of how the software will be used, is therefore essential in any Software Reliability Engineering (SRE) application. It is a fundamental concept which must be understood in order to apply SRE effectively and with any degree of validity. This section provides a detailed description of the OP.

A *profile* is a set of independent possibilities called *elements*, and their associated probability of occurrence. The *operational* profile is the set of independent operations that a software system performs and their associated probabilities. Developing an operational profile for a system involves one or more of the following five steps:

1. Find the customer profile
2. Establish the user profile
3. Define the system-mode profile
4. Determine the functional profile
5. Determine the operational profile itself

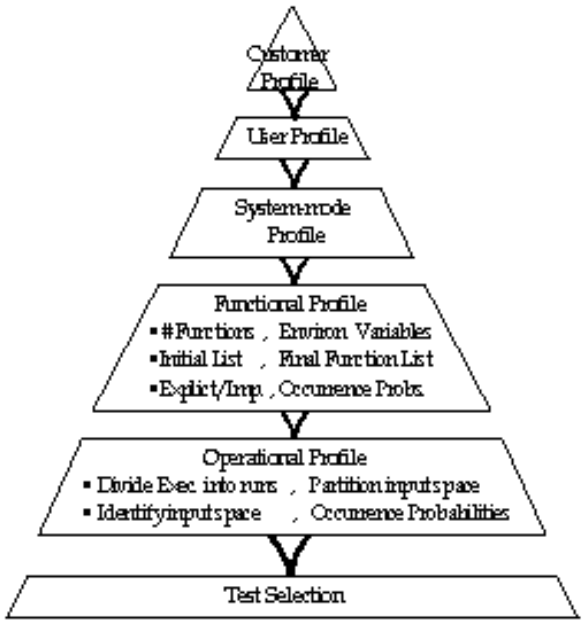

Figure 2. Operational Profile Developments

Figure 3 shows the elements involved in determining operational profiles from functions. A function may comprise several operations. In turn, operations are made up of many run types. Grouping run types into operations partitions the input space into domains. A domain can be partitioned into sub domains, or run categories. To use the operational profile to drive testing, first choose the domain that characterizes the operation, then the sub domain that characterizes the run category, and finally the input state that characterizes the run.

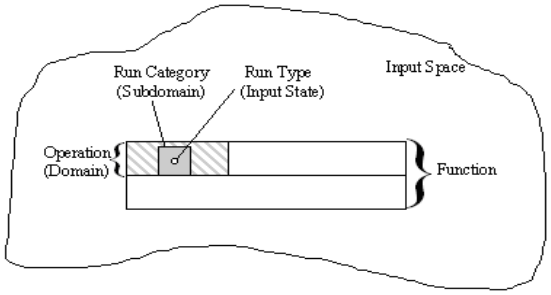

Figure 3. Operational Elements

VI. SEQUENTIAL BAYESIAN TECHNIQUE

A sequential maximum a posteriori estimation procedure based on Bayesian approach[10][13][14][18] is discussed here. The procedure is capable of utilizing the prior information. Let the general regression model be

$$Y = B_0 + \sum_{j=1}^{q-1} B_j X_j + \varepsilon. \qquad (1)$$

The equation for the Bayesian estimation of the model parameters, $\hat{B}$ is given as

$$\hat{B} = M + PXTQ{-}1(Y - XM), \qquad (2)$$

Where P is the covariance matrix of estimators (q x q), given as

$$P = (X^T Q X^{-1} + V^{-1}) \qquad (3)$$

$\hat{B}$ *: Estimated parameter vector (q x 1)*
M: mean value of parameter vector (q x 1) known from the prior information
X: independent variable matrix (n x 1)
V: covariance matrix of B known from prior information
Q: covariance matrix of errors.

Substituting

$$Bi = Bi{+}1, \bar{M} = Mi,\ Y = Yi{+}1,\ P = Pi{+}1,\ V = Pi,$$
$$X = Xi{+}1 \text{ and } Q = Ci{+}1$$

We get the recursive form of equation (2) and (3). Here C is a m x m diagonal covariance matrix of error and m is the number of observations. Substituting the above expressions in equation (2) and (3) we get

$$B_{i+1} = B_i + P_{i+1} X^T_{i+1} C_{i+1} (Y_{i+1} - X_{i+1} B_i) \qquad (4)$$

And

$$P_{i+1} = (X^T_{i+1} C^{-1}_{i+1} X_{i+1} + P^{-1}_i)^{-1} \qquad (5)$$

From matrix inversion theorem we know that,

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B\,(C^{-1} + DA^{-1}B)^{-1} DA^{-1}$$

Hence equation (5) may be written as follows

$$P_{i+1} = P_i - P_i X^T_{i+1} (X_{i+1} P_i X^T_{i+1} + C_{i+1}) \qquad (6)$$

Let $R = P_i X^T_{i+1} C_{i+1}$ & $H = X_{i+1}$

Then the following matrix identity holds

$$(I + RH)^{-1} R = R\,(I + HR)^{-1}$$

Therefore, substituting the values of R and H we get,

$$(I + P_i X^T_{i+1} C_{i+1} X_{i+1})^{-1} P_i X^T_{i+1} C_{i+1} = P_i X^T_{i+1} C^{-1}_{i+1} (I + X_{i+1} P_i X^T_{i+1} C^{-1}_{i+1})^{-1}$$

$$(X^T_{i+1} C_{i+1} X_{i+1} + P^{-1}_i)^{-1} P_i X^T_{i+1} C_{i+1} = P_i X^T_{i+1} C^{-1}_{i+1} (I + X_{i+1} P_i X^T_{i+1} C^{-1}_{i+1})^{-1}$$

$$(X^T_{i+1} C^{-1}_{i+1} X_{i+1} + P_i)^{-1} P^{-1}_i X^T_{i+1} C^{-1}_{i+1} = P_i X^T_{i+1} C^{-1}_{i+1} C_{i+1} (C_{i+1} + X_{i+1} P_i X^T_{i+1})^{-1}$$

$$P_{i+1} X^T_{i+1} C^{-1}_{i+1} = P_i X^T_{i+1} (X_{i+1} P_i X^T_{i+1} + C^{-1}_{i+1})^{-1} \qquad (7)$$

Substituting equations (6) & (7) in (4) & (5) we get,

$$A_{i+1} = P_i X^T_{i+1} \qquad (8)$$

$$D_{i+1} = C_{i+1} + X_{i+1} A_{i+1} \qquad (9)$$

$$K_{i+1} = A_{i+1} D_{i+1} \qquad (10)$$

$$E_{i+1} = Y_{i+1} - X_{i+1} B_i \qquad (11)$$

$$B_{i+1} = B_i + K_{i+1} E_{i+1} \qquad (12)$$

$$P_{i+1} = P_i - K_{i+1} A^T_{i+1}. \qquad (13)$$

Equations (8) to (13) are the governing equations for the sequential estimation procedure of the parameters. If the number of observations is one then no matrix inversion is involved and the computation becomes efficient. Thus for one observation, equation (8) to (13) may be rewritten as follows:

$$A_{i+1} = \sum P_{uk,i} X_{k,i+1} \quad (14)$$

$$D_{i+1} = \sigma^2_{i+1} + \sum X_{k,i+1} A_{k,i+1} \quad (15)$$

$$K_{u,i+1} = \frac{A_{u,i+1}}{D_{i+1}} \quad (16)$$

$$E_{i+1} = (Y_{i+1} - \sum X_{k,i+1} B_{k,i}) \quad (17)$$

$$B_{u,i+1} = B_{u,i} + K_{u,i+1} E_{i+1} \quad (18)$$

$$P_{uv,i+1} = P_{uv,i} - K_{u,i+1} A_{v,i+1,} \quad (19)$$

Where $u = 1, 2, 3, \ldots q$, $v = 1, 2, 3, \ldots, q$, q is the number of parameters and σ^2_{i+1} is the variance of $Y_{i+1.}$ Here in equation (15) S is used instead of σ^2_{i+1} to denote the error variance obtained from linear regression method. So equation (15) becomes

$$D_{i+1} = S + \sum X_{k,i+1} A_{k,i+1.} \quad (15A)$$

VII. ESTIMATION OF MODEL PARAMETERS

In the following paragraphs the description of the model is followed by the parameter estimation using proposed algorithm.

Let $X_t = X^{\theta}_{t-1}\delta$ where θ **is constant** and values of $\theta > 1$ **mean growth of reliability and** $\theta < 1$ **means decay of reliability**. **δ is the error due to some uncertainty in power law**. Taking natural logarithm on both sides we get

$$\log X_t = \theta \log X_{t-1} + \log \delta \quad (20)$$

$$\text{or } Y_t = \theta Y_{t-1} + B_1, \quad (21)$$

$$\text{Where } Y_t = \log X_t, B_1 = \log \delta.$$

To apply the above-mentioned algorithm for general sequential procedure given in equations
(14) to (19) the expression (21) becomes

$$Y = B_1 X_1 + B_2 X_2,$$

Where Y is Y_t, $B_1 = \log \delta$, $B2 = \theta$, $X_2 = Y_t - 1$ and X_1
is a dummy variable taking a constant value 1. Here t denotes the stage of testing and Xt denotes the time between failures.

VIII. SOFTWARE RELIABILITY PREDICTION:

Software reliability predictions[2][17] are made during the software development phases that precede software system test, and are available in time to feed back into the software development process. The predictions are based on measurable characteristics of the software development process and the products produced by that process.

Figure 4 shows the software reliability prediction process. Product and process metrics are collected and used to predict the initial failure rate and fault content. From these quantities, the reliability growth model parameters are predicted, and then the growth model is used to obtain estimates of the test time and resources needed to meet reliability objectives.

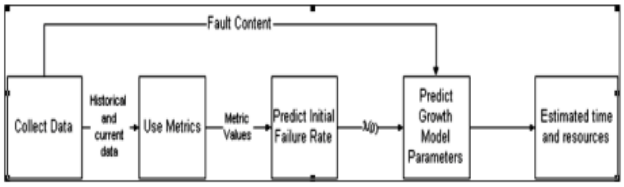

Figure 4. Software Reliability Prediction Procedure

The final outcomes of a software reliability prediction include:

- Relative measures for practical use and management.
- A prediction of the number of faults expected during each phase of the life cycle.
- A constant failure rate prediction at system release that can be combined with other failure rates.

IX. SOFTWARE RELIABILITY GROWTH TESTING:

Software reliability growth testing[20][22] takes place during the software system test phase, after the software has been fully integrated. During growth testing, the software is executed in an environment with inputs that most closely simulate the way the software is expected to be used in the field. In particular, the inputs are randomly selected in accordance with the software's operational profile.

The quality of testing is directly related to reliability growth and is a function of various system level tests that validate the software from more than one perspective. System tests can validate domains, paths, states, transaction flow, error handling, etc. The quality of testing is also related to testing the functionality that is executed most often by end user, most critical to end user, and most error prone.

An operational profile associates each input state or end-user function with a probability of occurrence. Testing according to the operational profile is efficient with respect to failure intensity reduction, because it reveals those faults that the user is most likely to encounter in use, those faults that contribute most to the program failure rate. When a failure is observed, the execution time, among other information, is recorded. The observed failure times are used as input to a statistical estimation technique that determines the parameters of the software reliability growth model. This way, the current reliability can be measured and the future reliability can be forecasted. **Figure 5** depicts a failure intensity curve.

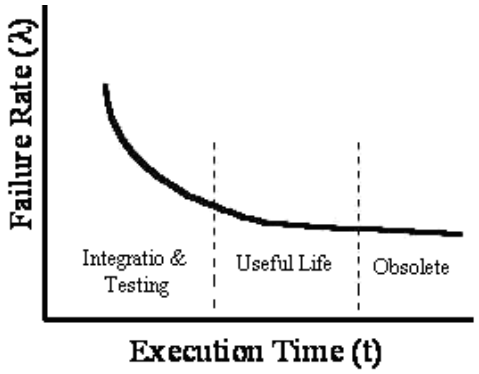

Figure 5. Software Failure Intensity Curve

Software reliability growth testing assumes that faults exist in the software and they will be uncovered during execution to produce software failures. As testing proceeds, failures will occur, the faults underlying the failures are identified and removed, the system is recompiled, and new input states are selected randomly from the operational profile. As software faults are removed, the failure intensity should decrease over time. This should continue until enough faults have been removed from the system to meet reliability goals.

X. CONCLUSION

This research paper attempts to focus on analysis of reliability of software using Sequential Bayesian Technique. The document describes the design constraints that are to be considered when the system is to be designed, and other factors necessary to provide a complete and comprehensive description of the requirements for the software.

REFERENCES

[1] Basu, S. and Ebhrahimi, N. (2003). Bayesian software reliability models based on martingale processes, *Technometrics*, 45, 150 – 158.

[2] Chatterjee S, Misra R B, Alam S 1997 Joint effect of test effort and learning factor on software reliability and optimal release policy. *Int. J. Sys. Sci.* 28(4): 391–396

[3] Chatterjee S, Misra R B, Alam S S 1998 A generalized shock model for software reliability. *Comput. Elect. Eng.-An Int. J.* 24: 363–368

[4] Fakhre-Zakeri I, Slud E 1995 Mixture models for reliability of software with imperfect debugging: Identifiably of parameters. *IEEE Trans. Rel.* 44: 104–113

[5] Goel A L, Okumoto K 1979 A time-dependent error detection rate model for software reliability and other performance measure. *IEEE Trans. Rel.* R-28: 206–211

[6] Gokhale S S, Lyu M R, Trivedi K S 2006 Incorporating fault debugging activities into software reliability models: A simulation approach. *IEEE Trans. Rel.* 55(2): 281–292

[7] Ibrahim, J.G., and Chen, M.H. (2000), "Power Prior Distributions for Regression Models," *Statistical Science*, 15, 46-60.

[8] Institute of Electrical and Electronics Engineers, ANSI/IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 729-1991, 1991.

[9] Jelinski Z, Moranda P B 1972 Software reliability research statistical computer performance evaluation. W Freiberger, Ed. Academic, NY, 465–484

[10] Jeske, D., Qureshi, M., and Muldoon, E. (2000) "A Bayesian methodology for estimating the failure rate of software," *International Journal of Reliability, Quality, and Safety Engineering*, 7, 153-168.

[11] Jeske, D., and Pham, H. (2001) "On the maximum likelihood estimates for the Goel-Okumoto software reliability model," *The American Statistician*, 55, 219-222.

[12] Kan, S.H., Parrish, J., and Manlove, D. (2001) " In-process metrics for software testing," *IBM Systems Journal*, 40, 220-241.

[13] Kuo, L., and Yang, T.Y. (1996), " Bayesian computation for nonhomogeneous Poisson processes in software reliability," *Journal of the American Statistical Association*, 91, 763-773.

[14] Littlewood B, Verrall J L 1973 A Bayesian reliability growth model for computer software. *Appl.Statist.* 22: 332–346.

[15] MIL-HDBK-781 Reliability Test Methods, Plans, and Environments for Engineering Development, Qualification, and Production, 14 July 1987.

[16] Musa J D 1975 A theory of software reliability and its application. *IEEE Trans. Software Eng.* SE-1:312–327

[17] Musa J D, Iannino A, Okumoto K 1987 Software reliability measurement. Prediction, Application, McGraw-Hill Int. Ed.

[18] Schick G J, Wolverton R W 1978 An analysis of competing software reliability model. *IEEE Trans. Software Eng.* SE-4: 104–120

[19] Soman K P, Misra K B 1993 On Bayesian estimation of system reliability. *Microelectronic Reliabilit*33: 1455–1459

[20] Sumita U, Shantikumar J G 1986 A software reliability model with multiple-error introduction & removal. *IEEE Trans. Rel.* R-35: 459–462

[21] Xie M 1987 A shock model for software reliability. *Microelectronic Reliability* 27: 717–724

[22] Yamada, S., Ohba, M., and Osaki, S. (1983), " S-shaped reliability growth modeling for software error detection," *IEEE Transactions on Reliability*, 32, 475-478.

[23]Zeephongsekul P, Xia G, Kumar S 1994 Software-reliability growth model: Primary failures generate secondary-faults under imperfect debugging. *IEEE Trans. Rel.* 43: 408–413.