International Journal of Research in IT and Management (IJRIM)
Available online at: http://euroasiapub.org
Vol. 13 Issue 11, November- 2023
ISSN(o): 2231-4334 | ISSN(p): 2349-6517 | Impact Factor: 8.106

Design and Evaluation of Reliability Prediction model to predict the reliability of interlocking software using computational intelligence.

Nidhi, Research Scholar,
Mewar University, Chittorgarh, Rajasthan
Pradeep Tomar, Research Supervisor,
Professor Dept of ICT, Gautam Buddha University, UP
DOI:euro.ijrim.77390.20980

Abstract: Ensuring the reliability of interlocking software is of paramount importance in safety-critical systems such as railway signaling. The complexity and criticality of interlocking software demand a robust approach for predicting its reliability during the development phase. This research paper presents a comprehensive software reliability prediction model tailored specifically for interlocking software systems. This model leverages historical data, fault analysis, and incorporates various software metrics to accurately estimate the reliability of interlocking software, thereby aiding developers in identifying potential issues and improving overall system dependability.

INTRODUCTION

Software reliability is defined as the probability of failure-free software operation for a specified period in a specified environment (ANSI definition). The Software failures are introduced by the system analysts, designers, programmers and managers during different phases of software development life cycle. To detect and remove these errors, the system software is tested. The quality of the software system in terms of reliability is measured by the removal of these errors. Software reliability prediction is based on parameters associated with the software product and its development environment. The study of software reliability can be categorized into three parts: modeling, measurement and improvement. Software reliability modeling has matured to the point that meaningful results can be obtained by applying suitable models to the problem. There are many models that exists, but no single model can capture a necessary amount of the software characteristics. Assumptions must be made to simplify the problem. Many software reliability models have been developed over the years, and most of these models are based on failure data collected in the process of software system testing or operation. These models can estimate the current software reliability and predict the time needed to achieve a reliability objective. However, failure data collected during the late testing phase may be too late for fundamental design changes, they can't give contribution to the enhancement of software product reliability. Software reliability prediction should be carried out throughout software development life cycle in order to improve software reliability effectively. Early software reliability prediction can provide guidance for developer to adopt proper technique and resources, so as to assure high reliability of software system. Examples of Software reliability prediction models, Rome laboratory developed a early prediction model which is based on software application type and development characteristics; Samuel Keene created a Development Process Prediction Model (DPPM) for early software reliability prediction, which is based on software development Capability Maturity Model (CMM);

Rayleigh model is used to predict software reliability in the software development process; Software engineer metrics, such as McCabe's cyclomatic complexity metric, were also used to predict software reliability. However, these early prediction models just can be applied to a specific software development stage, and have limited prediction capacity. Software reliability growth models, refer to those models that try to predict software reliability from test data. These models try to show a relationship between fault detection data (i.e. test data) and known mathematical functions such as logarithmic or exponential functions. The goodness of fit of these models depends on the degree of correlation between the test data and the mathematical function. Typically two broad categories of software reliability growth models (SRGMs) include parametric models and nonparametric models. Most of the parametric models are based on Non Homogeneous Poisson Process (NHPP) that has been widely used successfully in practical software reliability engineering. Artificial Neural Network (ANN) with software reliability models has aroused more research interest.

THE ROLE OF RELIABILITY PREDICTION

Reliability Prediction has many roles in the reliability engineering process. The impact of proposed design changes on reliability is determined by comparing the reliability predictions of the existing and proposed designs. The ability of the design to maintain an acceptable reliability level under environmental extremes can be assessed through reliability predictions. Predictions can be used to evaluate the need for environmental control systems. In today's competitive electronic products market, having higher reliability than competitors is one of the key factors for success. To obtain high product reliability, consideration of reliability issues should be integrated from the very beginning of the design phase. This leads to the concept of reliability prediction. Historically, this term has been used to denote the process of applying mathematical models and component data for the purpose of estimating the field reliability of a system before failure data are available for the system. However, the objective of reliability prediction is not limited to predicting whether reliability goals, such as MTBF, can be reached. It can also be used for:

- Identifying potential design weaknesses
- Evaluating the feasibility of a design
- Comparing different designs and life-cycle costs
- Providing models for system reliability/availability analysis
- Establishing goals for reliability tests
- Aiding in business decisions such as budget allocation and scheduling

According to the type of available software reliability information, certain software reliability models will be recommended. In the three-step software reliability prediction process, Keene's Development Process Prediction Model (DPPM), Rayleigh model, and models in the Computer Aided Software Reliability Estimation (CASRE) tool suite are used to progressively refine the prediction, which is continuously updated based on the most current reliability Software reliability is the probability of the software components of producing incorrect output. Software should not wear out and continue to operate after a bad result. No single model is either complete or fully developed which can be used in every situation. Software reliability is divided into following two categories based on assumptions, factors and mathematical function.

- **Prediction Model:** This model uses historical data to analyze previous data and some observations. It includes prior development and regular test phases. The model follows the concept phase and the predication from the future time.

- **Estimation Model:** Estimation model uses the current data from the current software development effort and doesn't use the conceptual development phases and can estimate at any time.

First the software development process quality is assessed rough reliability model. Software Engineering Institutes (SEI) Capability Maturity Model (CMM) is used, along with other development and deployment metrics, as a predictor of the latent fault density. This can be successfully done prior to the code development phase. Secondly, once the development phase is underway and code review and inspection data are collected, the Rayleigh model and the Software Error Estimation Program (SWEEP) tool is used as a predictor of the latent fault density of delivered code. Finally, once the code is running and reaches operational testing, the Computer Aided Software Reliability Estimation (CASRE) suite of empirical software reliability models is used to collect failure statistics and plot failure distributions. The failure distributions and statistical failure data are derived from software trouble reports. Some typical software trouble reports will be reviewed, categorizing the failure criticality, and completing an example. Furthermore, integrating software reliability into the overall system model is discussed. The integration of the Software Development Process, with methods to collect Operational Availability and Interruption of Service (IOS) metrics will be discussed.

Related work

According to *Cheung* [1] integrates failure states of individual components into State-Based Models. The paper describes a Hidden Markov Model (HMM) as DTMC, which delivers transition probabilities to failure states. The reliability of service, therefore, depends both on the reliability of the components and the probabilistic distribution of the utilization of the components to provide the service. The application of this model was to determine the expected penalty cost of failures, but the probability of control flow propagation was not included.

P. Kubat [2] proposed a model which includes the information about component execution time, this result in an SMP as a model of software architecture. In this model it is assumed

that the transition among the component follows DTMC such that:
$Q_i(r)$ = probability that task *r* will first call component *i*

P_{ij} (r)= probability that task *r* will call component *j* after the execution of component *i*
Failure intensity for a component '*i*' is given by α_i. The solution of this method is taken as a hierarchical. This is a good model for reliability estimation. This model considers the case for software composed of M components designed for K different tasks. According to *S. Gokhale* [3], they demonstrate the flexibility offered by discrete-event simulation to analyze such complex systems through two case studies, one of a terminating application, and the other of a real-time application with feedback control. They simulate the failure behaviour of the terminating application with instantaneous as well as explicit repair. They also study the effect of having fault-tolerant configurations for some of the components on the failure behavior of the application. In the second case of the real-time application, they initially simulate the failure behavior of a single version taking into account its reliability growth. Later they study the failure behavior of three fault tolerant systems, viz., Distributed Recovery Block (DRB), N-Version Programming (NVP) and N-Self Checking Programming (NSCP), which are built from the individual versions of the real-time application Testing is used to determine application architecture.

A coverage analysis tool ATAC is used to determine the branching probabilities between the components. Component failure behavior is given by a NHPP. The solution of this method is taken as Symbolic Hierarchical Automated Reliability and Performance Evaluator (SHARPE) that solves stochastic models for reliability, performance, and performs ability were selected. This approach describes an analytical model to reliability estimation. *W. Everett* [4], describes an approach to analyzing software reliability using component analysis. The analysis can begin prior to testing the software and can help in selecting testing strategies. It uses the Extended Execution Time (EET) reliability growth model at the software component level. This work describes how to estimate model parameters from characteristics of the software components and characteristics of how test cases and operational usage stress the software components. The work walks through an example illustrating the effects on reliability growth of (1) selecting test cases based on an operational profile versus selecting them based on uniform coverage of test cases, and (2) incremental delivery of software components to system test. The analysis can be done using commercial data analysis programs. Technique used by this model is NHPP. This is an earliest and good model for analyzing CBS reliability. In this approach it is required to keep track of cumulative amount of processing time that is spent in each component. According to *H. Singh et al.* [5], proposed a novel approach to reliability analysis of component-based systems. In this approach, Unified Modelling Language (UML) technique is incorporated into reliability prediction and assessment. The development of a probabilistic technique for reliability analysis applicable at the design-level, before system development and integration phases i.e. as soon as the properly annotated use case diagrams and sequence diagram become available. This approach is scalable because all the calculations are done by an automated tool. According to *Sherif Yacoub et al.* [6], introduces a probabilistic model and a Reliability Analysis Technique that is applicable to high-level designs. The technique is named *Scenario-Based Reliability Analysis* (SBRA). SBRA is specific to component-based software whose analysis is strictly based on execution scenarios. Using scenarios, they construct a probabilistic model named a Component-Dependency Graph (CDG). CDGs are directed graphs that represent components, component reliabilities, link and interface reliabilities, transitions and transition probabilities. In CDGs, component interfaces and link reliabilities are treated as first-class elements of the model. Based on CDGs, an algorithm is presented to analyze the reliability of the application as the function of reliabilities of its components and interfaces. The proposed approach is used to analyze the impact of variations and uncertainties in the reliability of individual components, subsystems, and links between components on the overall reliability estimate of the software system.

Wang Dong [7], in this approach various complex components relationships are analyzed using a path based model. These relationships have great impact on systems reliability .On the basis of these results a new tool has been developed to calculate the software application reliability. It assumes that all components reliabilities and transition probabilities are given. *Fan Zhang et.al.* [8], describes a sub domain-based reliability model to characterize the component into path-based architectural reliability model, and provides the enhanced composition algorithms to solve the model. The model is based upon the CDG. In this approach a systems operational profile is given. It is assumed that control flow transits from component *i* to component *j*, component *j* reliability is calculated as $T_{ij} * (R_{ij} \times W_{ij})$.
T_{ij} = the transition probability from component *i* to component *j*
R_j = reliability vector on each sub domain of component *j*
W_{ij} = the weighted vector for each sub domain of component *j* in transition from component *i* to *j*.

They assume that the model can fully capture the effect of different operational profiles on the overall reliability of system from aspects of both transition probability and component reliability.

Yuanjie Si [9] has given the approach in which five basic component composition mechanisms and their reliability estimation techniques are proposed. After calculating the reliability for each composition, a procedure is given to estimate the overall application reliability. The proposed approach estimates the reliability based on component composition mechanism and component utilization frequency.

According to *Chao-Jung Hsu et al.* [10], present an adaptive approach for testing path into reliability estimation for complex component-based systems. For path reliability estimation three methods have been proposed namely sequence, branch and loop structures. The proposed path reliability can be used for reliability estimation of overall application. Algorithmic approach is used. A promising estimation of software reliability can be given by this approach when testing information is available..

According to *Singh and Tomar* [11], this work proposed a Reliability Estimation Model for CBS to estimate the reliability based on reliability of individual components, path propagation probability and component impact factor. The path propagation probability depends on the decisions taken at component level which then decide the due course of that path. This model also estimates the impact factor of individual components on overall reliability. The impact factor can be used to focus the efforts to obtain the best reliability improvements. The impact of a component depends on its usage during the execution of the system.

PROBLEM IDENTIFICATION

Reliability is the key factor for interlocking or any other software to perform fault free and user satisfactory operations in real time process. According to the literature survey some problems are identifed : Interlocking software are real time and critical safety prone software for the travelling safety. So it is necessary that control signal of the software should give the fault free and reliable signal, fault should be easily detectable according to the present scenario or condition. observing some literature, it is fact that there is a scope of improvement in the following areas: Sometimes in real time scenario fault could not be detected exactly so it is difficult to decide which techniques or methods should be applicable when the number of data or cases are dynamic. This problem lead to testing coverage problem.

Reliability Prediction Model

Naive Bayes [Good, 1965; Langley et al., 1992] is a simple probabilistic classifier based on Bayes' rule. The naive Bayes algorithm builds a probabilistic model by learning the restricted prospects of each input attribute given a possible worth taken by the output attribute. This model is then used to predict an output value when we are given a set of inputs. This is done by applying Bayes' rule on the restricted probability of seeing a possible output value when the quality values in the given instance are seen together. Before recitation the algorithm we first define the Bayes' rule.

Bayes' rule states that;

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where $P(A|B)$ is distinct as the prospect of observing A given that B occurs. $P(A|B)$ is called subsequent probability, and $P(B|A)$, P(A) and P(B) are called prior probabilities. Bayes' theorem contributes a connection between the subsequent probability and the prior possibility. It allows one to find the probability of observing A given B when the individual probabilities of A and B are known, and the probability of observing B given A is also known. The naive Bayes algorithm uses a set of preparation examples to categorize a new occurrence given to it using the Bayesian approach. For instance, the Bayes rule is realistic to find the probability of observing each production class given the input attributes and the class that has the maximum probability is allocated to the instance. The possibility values used are attained from the counts of attribute values seen in the training set. In endure example, for a given example with two input attributes *temp* A_t*and temp* B_t, with values a and b individually, the value $v_M AP$ allocated by the naive Bayes algorithm to the the output attribute *temp* C_t is the one that has the highest possibility across all possible values taken by output attribute; this is known as the maximum-a-posterior (MAP) rule. The prospect of the output attribute taking a value v_j when the given input quality values are seen together is given by

$$P\,(v_j|a,\,b)$$

This possibility value as such is challenging to calculate. By applying Bayes theorem on this equation is

$$P\,(v_j|a,\,b) = \frac{P\,(a,\,b|vj\,)P\,(vj\,)}{P\,(a,\,b)} = P\,(a,\,b|v_j\,)P\,(v_j\,)$$

Where $P\,(v_j)$ is the probability of detecting v_j as the output value, $P\,(a,\,b|v_j)$ is the probability of detecting input attribute values a, b together when output value is v_j. But if the number of input attributes (a, b, c, d, ...) is large then it will not have sufficient data to estimate the probability$P\,(a,\,b,\,c,\,d,\,....|v_j)$. The naive Bayes algorithm resolves this problem by using the statement of conditional individuality for the all the input qualities given the value for the output. This means it assumes that the values taken by an attribute are not reliant on the values of other attributes in the occurrence for any given output. By relating the conditional individuality assumption, the probability of detecting an output value for the inputs can be attained by multiplying the probabilities of separate inputs given the output value. The likelihood value $P\,(a,\,b|vj)$ can then be simplified as

$$P\,(a,\,b|v_j) = P\,(a|v_j)P\,(b|v_j)$$

Where $P\,(a|v_j)$ is the possibility of observing the value a for the attribute *temp* A_t when output value is v_j. Thus the possibility of an output value v_j to be allocated for the specified input attributes Learning in the Naive Bayes algorithm includes finding the possibilities of $P\,(v_j)$ and $P\,(a_i|v_j)$ for all possible standards taken by the input and output qualities based on the preparation set providing. $P\,(v_j)$ is attained from the ratio of the number of time the value v_j is seen for the output attribute to the total number of occurrences in the training set. For an quality at position i with value a_i, the probability $P\,(a_i|v_j)$ is attained from the number of times a_i is seen in the training set when the output value is v_j. The naive Bayes algorithm needs all attributes in the example to be discrete.

Preliminaries

Let T_i, $i = 1, 2, 3, \ldots, n$ denote the age of the software as software faults are found and Y_i, $i = 1, 2, 3, \ldots, n$ denote the life length of the software at the i^th stage of testing following a modification attempting to remove an error, [22] Several models have been proposed for modelling the MTBF and estimating software failure times when they can be characterized by the Power Law Process.

Bayes, Empirical-Bayes Model

Mazucchi and Soyer developed a Bayes Empirical-Bayes model using similar assumption as the Littlewood-Verrall model, [21, 22]

Bayesian Setting

Given Y1, Y2, Y3, Y4, ...,Yn.

1. The probability density function (pdf) of Yi is given by:

$$f(y_i|\lambda) = \lambda_i e^{-\lambda_i y_i}, y_i > 0 \qquad (1)$$

with failure rates depending on the stage of testing.

2. Given the failure rates at each stage of testing, the life-lengths of the software ateach stage are statistically independent.
3. The failure rates at each stage of testing are random variables. The pdf of λ_i isgiven by:

$$g(\lambda_i|\alpha,\beta) = \frac{\beta^\alpha \ \lambda^{\alpha-1} e^{-\beta\lambda i}}{\Gamma(a)^i} \qquad (2)$$

4. Given the parameters α and β, the failure rates at each stage of testing are statistically independent.
5. Given the background information H, uncertainty of α and β is expressed via the prior marginal pdf's:

$$\Pi(\alpha|H) = \mu \quad 0 < \alpha < \mu \qquad (3)$$

$$\Pi(\beta|H) = \frac{b^a}{\Gamma(a)}\beta^{a-1}e^{-b\beta} \qquad (4)$$

where $\mu > 0, a > 0, and b > 0$ are known quantities.

1. Given background, the information H, $\alpha and \beta$ are statistically independent.
2. Given α, $\beta and (\lambda_1, \lambda_2, ..., \lambda_n)$, the $Y_i^j s$ are statistically independent with each Y_i sta- tistically independent of α,β and all λ's other than λ_i .

Computation of the posterior failures rates leads to integrals which cannot be expressed in closed form, Lindley's approximation, [22] is used to approximate the integrals. Mazucchi and Soyer's model was reported to be an improvement over the littlewood/Verrall model, after applying this model to some actual software failure data first reported in [22].

System Testing

Software testing is an important element of Software quality assurance and represents the ultimate review of specification, design and coding. The increasing visibility of S/W as a system element and the costs associated with Software failure are motivating forces for well planned, through testing. Though the test phase is often thought of as separate and distinct from the development effort–first develop, and then test–testing is a concurrent process that provides valuable information for the development team.There are at least three options for integrating Thesis Builder into the test phase:

- Testers do not install Thesis Builder, use Thesis Builder functionality to compile and source-control the modules to be tested and hand them off to the testers, whose process remains unchanged.
- The testers import the same Thesis that the developers use.
- Create a Thesis based on the development Thesis but customized for the testers (for example, it does not include support documents, specs, or source), who import it.

Unit Testing

This is the first level of testing. In this different modules are tested against the speci- fications produced during the design of the module. During this testing the number of arguments is compared to input parameters, matching of parameter and arguments etc. It is also ensured whether the file attributes are correct, whether the Files are opened before using, whether Input/output errors are handled etc. Unit Test is conducted using a Test Driver usually.

Output Testing

After performing the validation testing, next step is output testing of the proposed system since no system could be useful if it does not produces the required output generated or considered in to two ways. One is on screen and another is printed format. The output comes as the specified requirements by the user. Hence output testing does not result in any correction in the system.

Integration Testing

Integration testing is a systematic testing for constructing the program structure, while at the same time conducting test to uncover errors associated within the interface. Bottom- up integration is used for this phase. It begins construction and testing with atomic modules. This strategy is implemented with the following steps.

- Low-level modules are combined to form clusters that perform a specific software sub function.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upward in the program structure.

Test Cases

This provides the final assurance that the software meets all functional, behavioral and performance requirements. The software is completely assembled as a package. Validation succeeds when the software functions in which the user expects. After performing the validation testing, next step is output testing of the proposed system since no system could be useful if it does not produces the required output generated or considered in to two ways. One is on screen and another is printed format. The output comes as the specified requirements by the user. Hence output testing does not result in any correction in the system.

RESULTS AND IMPLEMENTATION

Software Implementation or Simulation (Tool) Pro- gramming Area For Experimental Results

Basically MATLAB is cast-off as an experimental and simulation software for the configuration of system established up and for location up the data transmission among various nodes existing in the set-up. MATLAB is an essential software design and commands are used as a replication device.
Then subsequent segment labels an outline approximately this supple and controlling device second-hand in numerous uses of engineering, science and arithmetic. A Program of

software corresponding MATLAB permits for operates and thinks about data, achieve controls and software design. It can achieve both refined and modest responsibilities.

Simulation Result

Software reliability is defined as the probability of failure free operation of software in a specified environment during a specified period. Software reliability has been the focus of several researches over the last four decades. One of the earliest software reliability models is the exponential Non homogeneous Poisson process developed by Goel and O kumoto in 1979. Most research works have considered fitting different software reliability models to different software reliability data where the estimates of the parameters of the models are

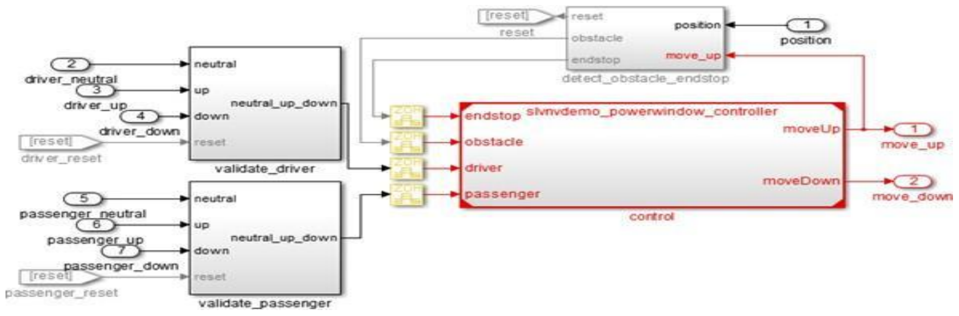

Figure 1.1: A slvnvdemo power window controller with validated passenger

obtained by Na¨ıve bayes prediction method. However, the problem of predictive analysis on the Goel – Okumoto software reliability model has not so far been explored despite the fact that predictive analysis is very useful for modifying, debugging and determining when to terminate software development testing process. This would lead to improved software reliability and efficient use of resources during software development testing. To assess and improve software reliability, software developers have to perform operational profile testing where they emulate the end-user environment during software testing. Operation profile testing is difficult and time consuming especially when there are multiple types of end-users and hence there is the need for software predictive analysis. The main objective of this study was to perform Bayesian predictive analyses on the Goel – Okumoto software reliability model. Informative and non-informative priors for one-sample case and non-informative prior for two-sample case has been used in the study.

The model starts by logging input signals to the component implementing the controller in its parent model and creating harness model for the controller from that logged data. A new test case in the harness model it captures all test cases and simulates the controller model for model coverage. Finally, it execute the controller with those test cases in sim- ulation mode and Software-In-the-Loop (SIL) mode. The slvnvdemopowerwindow model contains a power window controller and a low-order plant model. To create a harness model for the controller with the signals that simulate controller in the plant model, first log the input signals and then invoke harness genera- tion with that logged data. It can modify the test data in a harness model by manually editing the data values us- ing the Signal Builder user interface. It cans also new test cases by creating new signal

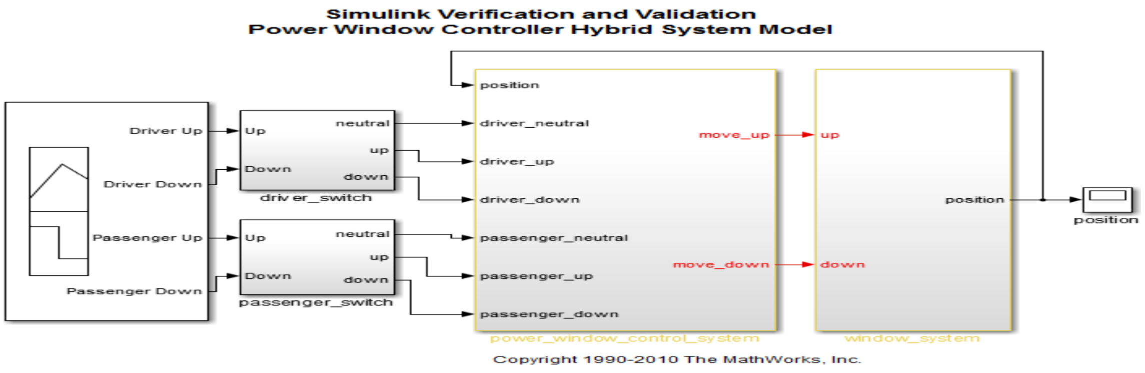

Figure 1.2: A slvnvdemo power window controllerhybrid system model with passenger Up and down

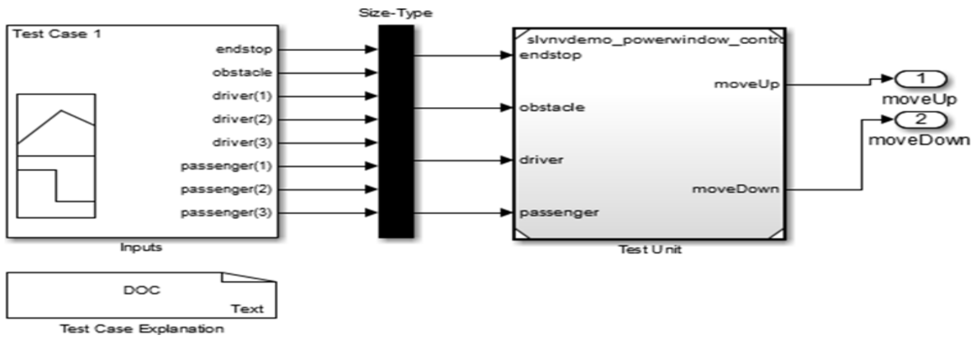

Figure 1.3: Test case explanation and unit testing of passenger for test case 1

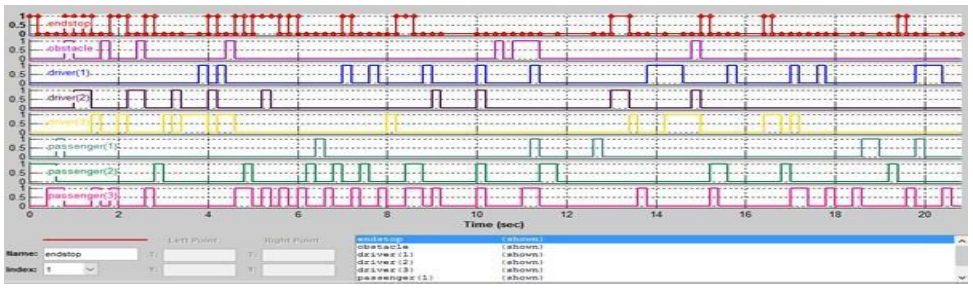

Figure 1.4: Signal presentation of end stop, obstacle, driver 1, 2 and passenger 1, 2

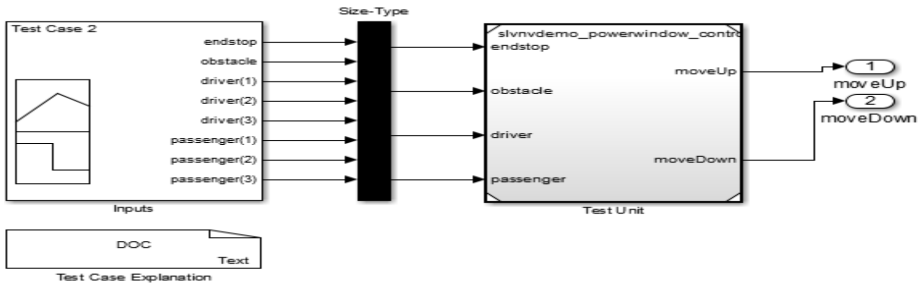

Figure 1.5: Test case explanation and unit testing of passenger for test case 2

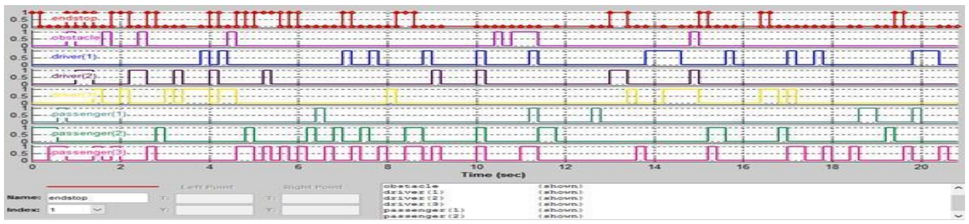

Figure 1.6: Signal presentation of end stop, obstacle, driver 1, 2 and passenger 1, 2

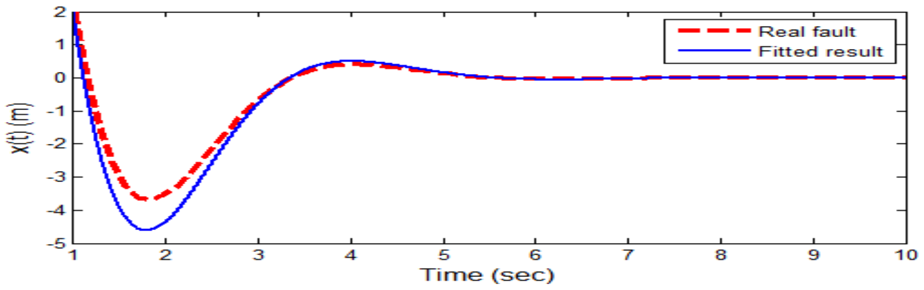

Figure 1.7: Real and fitted data

groups in the block. Alternatively, we can use the signal builder command to accomplish the same thing programmatically.

In order to programmatically execute the model slvnv demo power window controller with the test cases designed in the harness model, it first need to use the slvnvlogsignals function to capture the input values of all test cases in the necessary data format.

Use the slvnvruntest function to simulate the controller with all of the test cases designed in the harness and measure model coverage.Use the slvnvruncgvtest function to execute the model slvnvdemo power window controller in simulation mode, with test cases cap- tured from the harness model.

Figure 1.7 shows the fitted data and real fault. During the fitted data it also get some fault as shown in figure 1.7. Measures the percentage of the total variation about the mean accounted for by the fitted curve. It ranges in value from 0 to 1. Small values indicate that the model does not fit the data well. The larger, the better the model explains the variation in the data. The error approximation figure 1.10 incorporating the effect of time and covariates simultaneously and Sometimes, the effect of covariates are insignificant and the reduced form of the model may prove to be a better fit for the data and this can easily be obtained by setting *f* = 0, which gives us the better model. Similarly, when both covariate effect and time trend are insignificant, the model reduces to a NHPP model, with a constant recurrence rate, a. The number of intervals is always less or equal to number of failures that we observed because there can be more than one failure in any time interval.

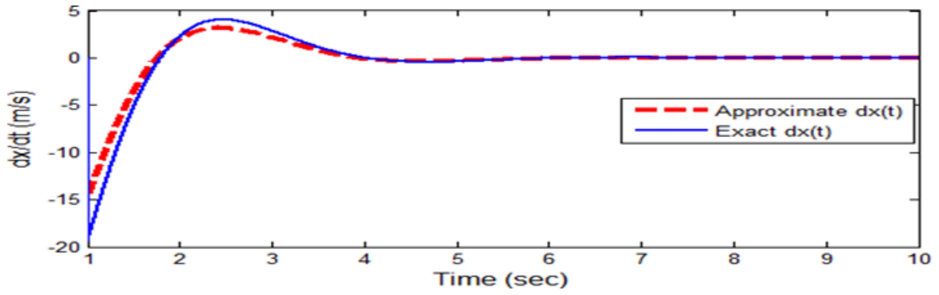

Figure 1.8: Approximate dx (t) and Exact dx (t)

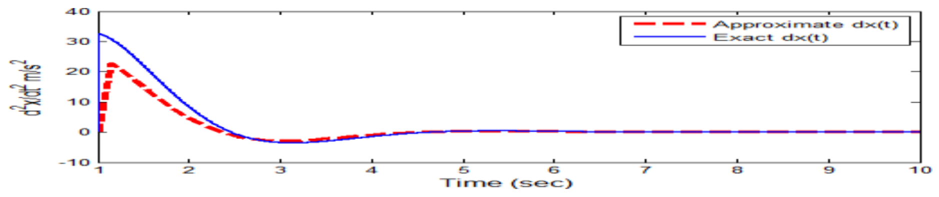

Figure 1.9: Approximate dx (t) and exact dx (t) on time

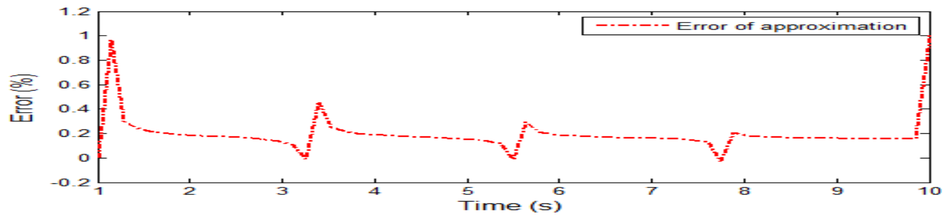

Figure 1.10: Error of approximation of software prediction

Conclusion

A new reliability model addressing the testing coverage is presented. Testing coverage is a very important measure for both software developers and users. Comparisons of this model with other existing NHPP models have also been presented. Two sets of software testing data have been used to examine the goodness-of-fit of all models. Two criteria have been used to compare the models and the results show that the new model fits significantly better. A software cost model has also been developed incorporating the testing cost, fault removal cost, and risk cost due to potential problems remaining in the uncovered code. Optimal software release policies are also obtained and it can be used to determine when to stop testing the software so as to minimize the expected total cost and satisfy the software reliability requirements. In this work the usage of Na¨ıve bayes in the problem of modeling the software reliability in the case of the dependence components performance is analysed.
Further research will be addressed to the practical confirmation of results and the practical application of the pattern and compared it with existing alternative approaches.

REFERENCES

[1] R. Cheung, (1980), “A User-Oriented Software Reliability Model”, in *IEEE Transactions on Software Engineering, 6(2), pp. 118-125.*

[2] P. Kubat (1989), “Assessing Reliability of Modular Software”. *Operations Research Letters*, (8), pp. 35-41.

[3] S. Gokhale and K. Trivedi (1998), “Dependency Characterization in Path-Based Approaches to Architecture-Based Software Reliability Prediction,” in proceeding of *IEEE Workshop on application Specific Software Engineering and Technology (ASSET’98)*, Richardson, Texas, Mar. 26–28, pp. 86–89.

[4] W. Everett, (1999), “Software Component Reliability Analysis”, in proceeding of *Symposium Application Specific systems and Software Engineering Technology (ASSET’99), pp 204-211.*

[5] H. Singh, V. Cortellessa, B. Cukic, E. Gunel and V. Bharadwaj (2001), “A Bayesian Approach to Reliability Prediction and Assessment of Component-Based Systems”, in proceeding of *12th IEEE International Symposium on Software Reliability Engineering, Hong Kong, pp. 12-21.*

[6] 18. S. Yacoub, B. Cukic, and H. Ammar. (2004), “A Scenario-Based Reliability Analysis Approach for Component-Based Software”, in *IEEE Transactions on Reliability, Vol. 28, No. 6, pp. 529-54.*

[7] Wang Dong , Ning Huang, Ye Ming (2008), “Reliability Analysis of Component–Based Software Based on Relationships of Components”, in proceeding of *IEEE Conference on Web Services, pp 814-815.*

[8] Fan Zhang, Xingshe Zhou, Junwen Chen, Yunwei Dong (2008), "A Novel Model for Component-Based Software Reliability Analysis" in proceeding of *11th IEEE High Assurance Systems Engineering Symposium, pp 303-309.*

[9] Yuanjie Si, Xiaohu Yang, Xinyu Wang,Chao Huang,Aleksander J. Kavs, (2011), "An Architecture-Based Reliability Estimation framework Through Component Composition Mechanisms", in proceeding of *International Conference on Computer Engineering and Technology, pp.165-170*

[10] Chao-Jung Hsu and Chin-Yu Huang (2011), "An Adaptive Reliability Analysis Using Path Testing for Complex Component based Software Systems" in *IEEE Transaction on Reliability Vol. 60, No 1 pp 158-170.*

[11] Singh A. P. And Tomar P. (2012), "A Proposed Methodology for Reliability Estimation of Component-Based Software", in proceeding of *International Conference on Optimization Modelling and Applications.*

[12] Tae-Hyun Yoo "The Infinite NHPP Software Reliability Model based on Monotonic Intensity Function"July 2015.

[13] Kapur K., Garg B., and Kumar S., Contributions to Hardware and Software Relia-bility, World Scientific, New York, 1999.

[14] S. M. K. Quadri, N. Ahmad, Sheikh Umar Farooq "Software Reliability Growth modeling with Generalized Exponential testing –effort and optimal SOFTWARE RELEASE Policy" February 2011

[15] Bijoyeta Roy1, Santanu Kr. Misra2, AradhanaBasak "A Quantitative Analysis of NHPP Based Software Reliability Growth Models" January 2014

[16] Cobra Rahmani"Exploitation of Quantitative Approaches to Software Reliability" 2008

[17] Richard Lai*, MohitGarg "A Detailed Study of NHPP Software Reliability Models"JOURNAL OF SOFTWARE, VOL. 7, NO. 6, JUNE 2012

[18] Chin-Yu Huang, Wei-Chih Huang "Software Reliability Analysis and Measurement Using Finite and Infinite Server Queueing Models" IEEE TRANSACTIONS ON RELIABILITY, VOL. 57, NO. 1, MARCH 2008

[19] Jagvinder Singh1, Adarsh Anand2, Avneesh Kumar3 " A Discrete Formulation of Successive Software Releases Based on Imperfect Debugging" September (2014)

[20] N. Ahmada, M. G. M. Khanband L. S. Rafi "Analysis of an Inflection S-shaped Software Reliability Model Considering Log-logistic Testing-Effort and Imperfect Debugging" January 2011

[21] JavaidIqbal "Software reliability growth models: A comparison of linear and ex-ponential fault content functions for study of imperfect debugging situations" 20 January 2017

[22] Carina Andersson, "A replicated empirical study of a selection method for software reliability growth models," Journal of Empirical Software Engineering, Vol. 12, No. 2, pp. 161–182, Apr 2007.

[23] Chin-Yu Huang, Sy-Yen Kuo and Michael R. Lyu, "An Assessment of Testing-Effort Dependent Software Reliability Growth Models," IEEE Transactions on Reliability,

Vol. 56, No. 2, pp. 198-211, Jun 2007.

[24] Lev V. Utkin, Svetlana I. Zatenko and Frank P.A. Coolen, ”Combining imprecise Bayesian and maximum likelihood estimation for reliability growth models,” In Proc. of the Sixth International Symposium on Imprecise Probability: Theories and Applications, Durham, UK, 2009.

[25] N. Ahmad, S. M. K Quadri and RazeefMohd, ”Comparison of Predictive Capability of Software Reliability Growth Models with ExponentiatedWeibull Distribution,” International Journal of Computer Applications, Vol. 15, No. 6, pp. 40-43, Feb 2011.

[26] P. K. Kapur, H. Pham, Sameer Anand and KalpanaYadav, ”A Unified Approach for Developing Software Reliability Growth Models in the Presence of Imperfect Debugging and Error Generation,” IEEE Transactions on Reliability, Vol. 60, No. 1, pp. 331-340, Mar 2011.

[27] S. M. K. Quadri, N. Ahmad and Sheikh Umar Farooq, ”Software Reliability Growth modeling with Generalized Exponential testing –effort and optimal Software Release policy,” Global Journal of Computer Science and Technology, Vol. 11, No. 2, pp. 27-42, Feb 2011.

[28] RadekDobias, Hana Kubatova “FPGA Based Design of the Railway’s Interlocking Equipments”

[29] K.VenkataSubba Reddy, Dr.B.Raveendrababu “Software Reliability Growth Model With Testing-Effort by Failure Free Software”International Journal of Engineering and Innovative Technology (IJEIT) Volume 2, Issue 6, December 2012

[30] Shinji Inoue, Shigeru Yamada “Lognormal Process Software Reliability Modeling with Testing-Effort” February 6th, 2013