IJCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Vol.-6, Issue-6, June 2018 E-ISSN: 2347-2693

A Review: Reliability Evaluation of Interlocking Software based on NHPP model

Nishi[1], Dinesh Kumar[2]

[1,2]Computer Science, Shri Ram Collage of Engineering and Management, MDU, FARIDABAD, INDIA

Available online at: www.ijcseonline.org

Accepted: 03/Jun/2018, Published: 30/Jun/2018

Abstract- Software reliability is one of the main factors to measure the quality of software. Since software errors cause spectacular failures in some cases, we need to measure the reliability factor to determine the quality of software product, predict reliability in the future, and use it for planning resources needed to fix failures. Software reliability models are applicable tools to analyze software in order to evaluate the reliability of software. During the past twenty five years, more than fifty different models have been proposed for estimating software reliability but many of software practitioners do not know how to utilize these models to evaluate their products. In this paper we will present a survey on different models of software reliability and their characteristics.

Keyword: Reliability Prediction, Non-Homogeneous Poisson Process, Failure Intensity

I. Introduction

Software reliability is defined as the probability of failure free software operation for a specified period of time in a specified environment (Musa 1980). Software reliability modeling has gained a lot of importance in many critical and daily life applications, which has led to a tremendous work being carried out in the field of software reliability modeling. Software reliability growth models (SRGMs) successfully have been used for estimation and prediction of the number of errors remaining in the software (Goel and Okumoto 1979; Littlewood 1979; Musa 1980, 1998; Norman 1997; Lyu 2005; Kapur et al. 1999, 2010; Pham 2006). The software practitioners and potential users can assess the current and future reliability through software testing using these SRGMs. In past four decades, the classical models have remained one of the most attractive reliability growth models in monitoring and tracking reliability improvements (Musa 2005, 2007; Pham 2006). Classical models are the NHPP based models that have been widely applied successfully in many real-life applications for estimation and prediction of software reliability such as Musa-basic model, Musa–Okumoto model, Littlewood– Verral model, Goel–Okumoto model. Alternatively, some traditional statistical methods such as maximum likelihood estimation (MLE), least square estimation (LSE), analysis of variance (ANOVA), linear regression analysis (LRA) and logistic regression have also been applied for software reliability estimation and prediction (Kohavi 1995; Phillip 2003). The major challenges of these models do not lie in their technical soundness, but their validity and applicability in real world projects particularly in web-based systems. On the other hand, learning and generalization capabilities of artificial neural networks (ANNs), and its proven successes in complex problem solutions has made it a viable alternative for predicting software failures during the testing phase (Karunanithi et al. 1992). The main advantage of ANN and other machine learning methods over NHPP based models is that it requires only past failure data as inputs, and less assumption required for modeling complex failure phenomena of software.
Machine learning is an approach concerned with the design and development of algorithms that allow computers to evolve the system behavior based on past and present failure data of software. Thus machine learning techniques are focused on learning automatically, recognizing complex patterns and making intelligent decisions based on past data. So that a machine is able to learn whenever it changes its structure, program, or data based on its inputs or in response to the external information in such a manner that it’s expected future performance improves significantly (Kohavi 1995). Thus it is quite natural for software practitioners and researchers to know that which particular method tends to work well for a given failure dataset and up to what extent quantitatively (Aggarwal et al. 2006; Goel and Singh 2009; Singh and Kumar 2010a, b, c).

Here we conduct an empirical study of machine learning methods such as ANNs, SVMs, CCNN, DTs and fuzzy inference system (FIS) for the prediction of software reliability in order to draw stronger conclusions leading to widely accepted and well-formed theories. In this paper we briefly investigate and focus on three main issues: (i) How accurately and precisely do the machine learning based models predict the reliability of software product at any point of time during testing phase? (ii) Is the performance of SVMs and DTs better than CCNN and ANNs using back propagation network (BPN), radial basis function network (RBFN) and Elman network models? (iii) Correlate between SVMs and DTs for software reliability prediction since their performance varies when applied to past failure data in a realistic environment.

A. Software Reliability Definition

Musa and Okumoto in 1984 defined software reliability as the "probability of failure free operation of a computer program in a specified environment for a specified period of time." NASA Software Assurance Standard, NASA-STD-8739.8 [7], defines software reliability as a discipline of software assurance that (Fig.1):

1. Defines the requirements for software controlled system fault/failure detection, isolation, and recovery;
2. Reviews the software development processes and products for software error prevention and/or reduced functionality states; and,
3. Defines the process for measuring and analyzing defects and defines/derives the reliability and maintainability factors.

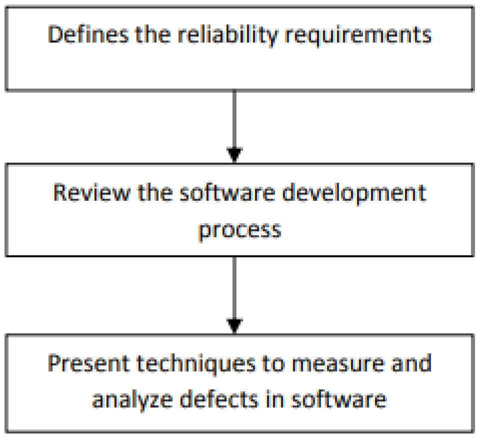
Defines the reliability requirements
Review the software development process
Present techniques to measure and analyze defects in software

Figure (1) Software reliability definition from

NASA Software Assurance Standard view as mentioned in NASA Software Assurance Standard, software reliability process consists of three main phases which are extended through the whole software life-cycle process. Define reliable requirements in initial phases of software lifecycle, review whole software process to prevent potential errors, and present some techniques to measure and analyze defects in software. In this paper we suppose that software process already supports the first and second phases and we try to study and categorize the techniques in the third section. In this research work, everywhere we talk about software reliability models we mean the models which can measure and analyze the defects in software to achieve two goals: representation of software reliability situation at present and estimation of software reliability condition in future. Modeling the failure process in a piece of software is a very challenging exercise partially because of the diverse and interlocking nature of faults that may exist and the methods that may be used to detect or discover them [1]. Software reliability models are used to achieve the quality of software and also to plan for resources needed to fix the problems in the maintenance phase. They have been used as the most important and successful predictor of software quality when it hits the market. In general, most of these reliability models use independent variables and show their modeling results in two-dimensional diagrams based on these independent variables.

Typical scopes of measurement (X axis) in reliability model diagrams include: 1- Calendar time (days of testing), 2- Cumulative testing effort (hours of testing); 3- Computer execution time (e.g. number of CPU hours). And typical dependent variables (Y) include: 1- Number of defects found per life cycle phase or total number ever, 2- Failure rate over time, 3- Cumulative number of failures over time, 4- Time between failures.

B. NHPP MODEL (Non Homogeneous Poisson Processes)

Software reliability is one of the important metrics for software quality assessment. The most commonly used models in the reliability analysis of software are the Non Homogeneous Poisson Processes (NHPPs) which are based on the assumption that the errors or bugs which have been incorporated in the software during the development phase, are removed during the testing phase. A great deal of research efforts in the past has been focused on modeling the software reliability growth during the testing phase. These reliability models which are referred to as black box models, treat the software as a whole; considering its interactions only with the external environment, without regarding the internal structure of the software. These models are based on the stochastic modeling of the failure process, assuming a parametric model of cumulative number of failures over a finite time interval.

II. Litrature Survey

Several authors have studied various software release problems in different scenarios. Chin-Yu et al studied optimal allocation of testing resource considering cost, reliability and testing effort [18]. Dai et al studied optimal testing resource allocation with generic algorithm for modular software systems [19]. Gokhale et al studied incorporating fault debugging activities into software reliability models [20]. Jain et al discussed optimal release policies for software reliability growth model (SRGM) with maintenance costs [21]. Jain and Priya proposed the optimal policies for software release time by employing a Delayed S-shaped model for software reliability growth [22]. They followed the white-box approach for analyzing. Kimura et al. analyzed the software release problems with warranty cost and reliability requirement [23]. Prasad et al studied measurement of software reliability using Sequential Bayesian technique [24]. Quadri et al studied software optimal release policy and reliability growth modeling [25]. Quadri and Ahmad studied software reliability growth modeling with new modified weibull testing the software reliability. [26]. Worwa et al studied a discrete-time software reliability growth model and its applications for predicting the number of errors encounter during program testing [27]. Yamada and Osaki determined an optimal software release policy for a non-homogeneous software error detection rate model [28].

III. Fault Prediction Techniques

1). Decision Tree: Decision trees are great and standard tools for classification and prediction. It produces classifiers in a form of structure of tree where each leaf node represents decision node. In this technique, classification starts from root of the tree and continues to move down until leaf node is reached. Classification helps in classifying faulty and non-faulty modules. Prediction helps in predicting faulty and non-faulty modules. Decision tress helps in developing fault prediction models that predicts faults.

2). Neural Network: Neural Network helps in recognizing patterns from the data set. An artificial neural network is composed of many artificial neurons that are interconnected together according to specific network architecture. The goal of the neural network is to transform the inputs into meaningful outputs. Adaptive Resonance Neural Network is generally used for defect prediction in software systems. It helps in identifying faulty modules very excellently. The benefit of using this technique is that it assists in decreasing effort and cost of developing software.

3). Density based clustering approach: Density Based Clustering is a clustering algorithm. It can be used to estimate the number of faulty and non-fault modules in software system. Clusters are defined as areas of higher density. Following are the parameters used in this: Eps: supreme radius of the nighbourhood MinPts: least possible number of points in an Eps-neighbourhood of that point.

4). Bagging method: It creates base learners on many data subsets that are uniformly sampled from the original data, and then uses a linear combination to aggregate them. It is also referred as Bootstrap Aggregating. Combination technique can be majority voting. It also helps in identifying faulty and non-faulty modules with data sets that suffers from imbalance problem. This method can increase the performance of the defect data predictions.

5). Naïve Bayes: It is a classifier based on Bayes theorem used in software fault prediction. It resolves the several difficulties like spam classification (to predict whether email is spam or not), medical diagnosis (given list of symptoms, predict whether patient has cancer or not) and so on. This method can be used to predict faulty and non-faulty modules.

IV. Challenges In Software Reliability

The evolution of computer technology is creating for safety-critical systems new challenges and different types of failure modes. Modern computer processors are often delivered with errors, while intelligent hardware subsystems may exhibit nondeterministic behaviour. Operating systems and programming languages are becoming increasingly complicated and their implementations less trustworthy. In addition, component-based multi-tier software system architectures exponentially increase the number of failure modes, while Internet connectivity exposes systems to malicious attacks. Finally, IT outsourcing and blind reliance on standards can provide developers with a false sense of security. Planning in advance for the new challenges is as important as embracing the new technology. Although there have been a lot of researches on software reliability growth models, the problem that limited considerations of imperfect debugging phenomenon in the existing models is still not solved.

Despite the important advances made over the last decades in the area of software engineering and the successful realisation of many safety-critical software systems, the evolution of computer technology is creating new challenges and different types of failure modes.

Modelling the failure process in a piece of software is a very challenging exercise (partially because of the diverse and interlocking nature of faults that may exist and the methods that may be used to detect or discover them).

V. Application of software reliability model

Software reliability is one of the most important characteristics of software quality. Its measurement and management technologies during the software life-cycle are essential to produce and maintain quality/reliable software systems. In this paper, we discuss software reliability modeling and its applications. As to software reliability modeling, hazard rate and NHPP models are investigated particularly for quantitative software reliability assessment.

Software reliability is an important component in critical business applications. Developing reliable software is very difficult because there is interdependence among all the software modules as of existing software. This is also very difficult to find out whether the software being delivered is reliable or not. The users or customer feedback i.e. problem reports, system outages, complaints or compliments indicate the reliability of any software product.

VI. Conclusion

In this paper, we studied various techniques (like decision tree, neural network, naïve Bayes and so on) to predict faults in software systems. The main aim is to review the performance of different techniques in software fault prediction. Fault prediction using these techniques helps in improving the quality of the software. The fault prediction in software is significant because it can help in directing test effort, reducing cost, and increasing quality of software and its reliability.

References

[1] Musa JD (1980) Software life cycle empirical/experience data, data & analysis center for software. Available at http://www.dacs.org. Accessed 17 Sep 2010

[2] Goel AL, Okumoto K (1979) Time-dependent fault detection rate model for software and other performance measures. IEEE Trans Reliab 28(3):206–211

[3] Littlewood B (1979) Software reliability model for modular structure. IEEE Trans Reliab 28(3):241–246

[4] Musa JD (1998) More reliable, faster, cheaper testing with software reliability engineering. Softw Qual Prof 1(1):27–37

[5] Norman F (1997) Application of software reliability engineering for NASA space shuttle. International Symposium on Software Reliability Engineering (ISSRE) 1:71–82

[6] Kapur PK, Garg RB, Kumar S (1999) Contributions to hardware & software reliability. World Scientific, Singapore

[7] Kapur PK, Gupta A, Jha PC, Goyal SK (2010) Software quality assurance using software reliability growth modelling: state of the art. Int J Bus Inf Syst 6(4):463–496

[8] Musa JD (2005) Software reliability engineering: making solid progress. Softw Qual Prof 7(4):5–16

[9] Pham H (2006) System software reliability. Springer, London

[10] Kohavi R (1995) The power of decision tables. In: The eighth european conference on machine learning (ECML-95), Heraklion, Greece, pp 174–189

[11] Phillip S (2003) DTReg predictive modeling software available at http://www.dtreg.com. Accessed 8 Jan 2011

[12] Karunanithi N, Whitley D, Malaiya Y (1992) Prediction of software reliability using connectionist models. IEEE Trans Softw Eng 18(7):563–574

[13] Aggarwal KK, Singh Y, Kaur A, Malhotra R (2006) Investigating the effect of coupling metrics on fault proneness in object-oriented systems. Softw Qual Prof 8(4):4–16

[14] Goel B, Singh Y (2009) An empirical analysis of metrics. Softw Qual Prof 11(3):35–45

[15] Singh Y, Kumar P (2010a) A software reliability growth model for three-tier client–server system. Int J Comp Appl 1(13):9–16. doi: 10.5120/289-451

[16] Singh Y, Kumar P (2010b) Determination of software release instant of three-tier client server software system. Int J Softw Eng 1(3):51–62

[17] Singh Y, Kumar P (2010c) Application of feed-forward networks for software reliability prediction. ACM SIGSOFT Softw Eng Notes 35(5):1–6

[18] Chin-Yu Huang, Sy-yen Kuo, Lyu, M.R., 2004, Optimal allocation of testing resource considering cost, reliability and testing effort. 10th Pacific Rim International Symposium on Dependable Computing, 2004, 103-112.

[19] Dai Y., Xie M., Poh K. and Yang B., 2003, Optimal testing resource allocation with generic algorithm for modular software systems, *Journal of Systems and software*, 66, 1, 47-55.

[20] Gokhale S.S., Lyu N. and Trivedi K., 2006, Incorporating fault debugging activities into software reliability models: A simulation approach, *IEEE transactions on reliability*, 55, 2, 281-292.

[21] Jain M. Maheshwari S. and Priya K., 2005, Optimal release policies for software reliability growth model (SRGM) with maintenance costs, *Journal of ICT*, l4, 99-115.

[22] Jain, M. and Priya, K., 2002, Optimal policies for software testing time, *Computer Society of India*, 32,3, 25-30.

[23] Kimura, M., Toyota, T. and Yamada, S.,1999, Economic analysis of software release problems with warranty cost and reliability requirement, *Reliability Engineering and System Safety*, 66, 49-55.

[24] Prasad L., Gupta A. and Badoria S., 2009, Measurement of Software reliability using Sequential Bayesian Technique, Proceedings of the world congress on Engineering and Computer Science, 11, 242-246.

[25] Quadri S., Ahmad N. and Peer M., 2008, Software optimal policy and reliability growth modeling, *Computation for national development*, 2008, 247-256.

[26] Quadri S.M.K. and Ahmad N., 2010, Software Reliability Growth Modeling with new modified Weibull testing effort and optimal release policy, *International Journal of Computer Applications*, 6 ,12

[27] Worwa K., 2005, A discrete-time software reliability growth model and its applications for predicting the number of errors encountered during program testing, *Control and Cybernetics*, 34, 2, 589-606.

[28] Yamada, S. and Osaki, S., 1986, Optimal software release policy for a nonhomogeneous software error detection rate model, *Microelectronics and Reliability*, 26, 691-702.