Journal of Software Engineering and Applications, **2012, 5, 737-742**
http://dx.doi.org/10.4236/jsea.2012.59086 Published Online September 2012 (http://www.SciRP.org/journal/jsea)

Role of Software Reliability Models in Performance Improvement and Management

Padmanabha Aital[1], P. Sashikala[2]

[1]SIOM, Nashik, India; [2]IBS, Hyderabad, India.
Email: padmanabha.aital@siom.in, pad2aital@gmail.com, sashikala@ibsindia.org, sashikalab@gmail.com

Received May 23rd, 2012; revised June 26th, 2012; accepted July 9th, 2012

ABSTRACT

Software reliability models describe the failure behavior of the software. The models are used to evaluate the software quantitatively. They assess the reliability of the software by predicting faults or failures for a software. Reliability is one of important quality attributes of the software in which software end user is more interested rather than the software developer. Hence, the performance of a software can be improved by incorporating important quality attributes like reliability, maintainability and availability of the software along with performance attributes like response time and throughput. The paper discusses about the role played by important software reliability models in analyzing the failure prediction of the software. It also explores the strong relationship that exists between quality attributes and performance attributes. With some illustrations highlighting the necessity of in-depth understanding of the link that exists between reliability and performance of the software, the derived knowledge helps in improving the performance of the software sustainably over a period of time and manage the software more effectively.

Keywords: Software Reliability; Software Reliability Models; Software Availability Software Performance; Performance Management

1. Introduction

Software reliability is an important aspect of functioning of a software system which may be a combination of different software sub systems or embedded in a computing environment that provides inputs to the software system [1]. Software reliability is the probability that the system will function without failure for a specified period of time under stated conditions. Due to error in software by human action or discrepancy between computed, observed, or measured value and specified value of some important reliability parameter may lead to fault in the software. The fault, if unchecked may or may not result in software failure depending on the operating conditions. The software reliability is similar to hardware reliability in some aspects. Most of the reliability quantities are defined in terms of time. The execution time of software (CPU time) is the actual time spent by the computer in executing the software. The clock time is the time elapsed between starting of computer and shutting it down including idle time [2]. All these times can be converted into calendar time which will be useful for system development personnel to calculate human effort required to develop software. Some of the reliability measures are cumulative failure function, failure intensity function, failure rate function and mean time to failure function [1,3].

The operational profile of a system is defined as the set of operations that the software can execute along with probability with which they will occur. It will help us to identify operations which are failure prone and also affect reliability and performance of the software [2].

The software reliability measurement involves use of failure data by software reliability models to estimate and predict software reliability. The type of failure data used by number of software reliability models belongs to two types—Failure count data and time between failures [2].

Software reliability model specifies the general form of the dependence of the failure process on the principal factors that affect it—fault introduction, fault removal and operational environment [1].

Fault prevention is by construction to avoid fault occurrences. The fault removal talks about detection by testing and removal of fault. Fault tolerancehighlights the issues of redundancy to accommodate any failure in operation. Fault/failure forecasting predictsthe presence of faults and occurrence and consequences of failure [4,5].

2. Software Reliability Models

Software reliability models consist of a wide variety of models based on statistical theory and Bayesian approach.

The paper presents some of the popular models and their model forms. Though the time domain based models overcast the other type of models in terms of usability and wider application, other type of models also find their place depending on suitability and operational environment [2]. Combination of models can also be used to improve reliability measurement. Time domain based model is integrated with input domain based model to form a tree based model which finds its application in large commercial software [1].

2.1. Time between Failures (TBF) Models

The common approach is to assume that time between *i*th and $(i-1)$ failure will follow a certain distribution whose parameters will depend on number of faults remaining in the program during the given interval. From this, the estimates of the parameters are calculated which give reliability and other parameters of interest on subsequent calculations. Another approach is to treat the failure times as realizations of a stochastic process and an appropriate time-series model to describe the underlying failure process [1,2].

Example-Jelinski-Moranda model

One of the earliest models proposed which is still being applied today is the de-eutrophication model developed by Jelinski and Moranda, The elapsed time between failure is taken to follow an exponential distribution with a parameter that is proportional to the number of remaining faults in the software, *i.e.* Mean time between failures (MTBF) is $1/\phi\left(N-(i-1)\right)$. Here t is any point in time between the occurrence of the $(i-1)$th & *i*th fault occurrence.The quantity ϕ is the proportionality constant and N is the total number of faults in the software from the initial point in which the software is observed [1].

Model Form: Hazard Function $Z(ti)=\phi\left[N-(i-1)\right]$ where N = Number of errors present in the software at the beginning of the test phase ϕ = Proportional constant.

Mean failure value function, $\mu(t)=N\left[1-\exp(-\phi t)\right]$ and Failure intensity function, $\lambda(t)=N\phi\exp(-\phi t)$.

2.2. Fault/Failure Count (FC) Models

Number of faults or failures in a time interval is taken into consideration rather than times between failures. The failure counts are assumed to follow a known stochastic process with a time dependent discrete or continuous failure rate. The useful reliability parameters are calculated from the estimates obtained from the observed value.

Musa's Basic Execution Time Model

This model has had the widest distribution among the software reliability models. This model was one of the first to use the actual execution time of software component on computer for modeling process. Musa feels that execution time is more reflective of the actual stress induced on the software system than the amount of the calendar time that has been elapsed [2]. This model can even use time between failures (tbf) as input failure data. Hence, the categorization of models under TBF Models and FC Models is not rigid. It is entirely depends on what type of failure data is fed to the model and failure data can be converted from one form to the other to suit the model failure data requirement.

Model Form: Mean failure value function, $\mu(t)=\beta_o\left[1-\exp(-\beta_1 t)\right]$ and Failure intensity function, $\lambda(\mu)=\lambda_0\exp(-\beta_1\mu)$ where β_o = Total number of faults that would be detected in the time limit, β_1 = a constant & failure intensity decay parameter. The product of β_o and β_1 = λ_0 where λ_0 is initial failure intensity at the start of execution.

Illustration 1:

Assume that initial failure intensity is 5 failures per execution hour and the failure intensity decay parameter is 0.01 per failure. Let the failure experienced or the fault detected in a time limit is 100 (mean failure value function), Then current failure intensity is

$$\lambda(\mu)=\lambda_0\exp(-\beta_1\mu)=5\exp\left[-(0.01)(100)\right]$$
$$=5\exp(-1)=1.84 \text{ perexecution hour.}$$

A similar illustration can be worked out for Jelinski-Moranda Model which is a TBF Model.

The software faults predicted by these models have to be removed to enhance the reliability of the software [1]. The software faults may not directly affect the functioning and performance of the software unless they result as software failures. Hence, sufficient care should be taken and resources be allotted to remove software faults. The current failure intensity function of these models help in knowing impending failures in the software. The software models help in predicting the performance of the software by showing failure behavior of the software over a time. Depending on the performance level expected out of software in terms of timely response and throughput, failure behavior of the software has to be studied in depth and non-testing methods like Formal Technical Reviews (FTRs), informal reviews, walkthroughs and inspections have to be deployed to find faults. It is advisable to use these methods during analysis and design phases of the software development. If sufficient faults are not found or removed, then extensive testing has to be carried out to raise reliability of software by reducing the number of faults in the software. The reliable software without faults or minimum number of faults will assure good performance of software. Hence, reliability of the software directly affects the functioning and performance of the software.

3. Software Dependability and Its Attributes

Software Reliability is one of the attributes to define dependability (trustworthiness) of the software. The other important attributes are availability, maintainability, safety, confidentiality and integrity [1]. The availability is the preparedness of the software for use. Unless the software is available or fit for use, it is directly affecting reliability and proper functioning of the software and indirectly the performance of the software. The maintainability of the software deals with "down time" or minimum Mean Time to Repair (MTTR) of a software/ system and ease with services are restored. This is one of the important attribute which affects performance of the system. Hence, both reliability and maintainability are important to assure proper availability of the software system to carry out stated functions or operations in stipulated time. The functional failure or sub-optimal functional behavior affects the performance and performance improvement initiatives for the software.

The availability of a software system can be measured in 3 different ways depending on the time elements taken into consideration. They are Inherent availability, Achieved availability and Operational availability [6].

Inherent availability is the probability that a software system will operate satisfactorily when used under stated conditions in an ideal support environment without any scheduled or preventive maintenance [6]. The software system includes both software and hardware. Hence, inherent availability is system availability which is given as below

Inherent Availability, $\mathbf{A_i = MTBF/(MTBF + MTTR)}$ where MTBF is Mean time between failures and MTTR is mean time to repair [6]. It is obvious from above relation that in order to have higher inherent availability, the MTTR should be as low as possible [7].

Illustration 2:

Let us assume a system is having MTBF of 10 execution hours (CPU Hrs) and MTTR which is equivalent to 1.6 execution hours (CPU Hrs), then

$$\text{Inherent availability} = 10/(10+1.6) = 0.8620$$

Achieved availability is taking into consideration active maintenance down time resulting from both preventive and corrective maintenance. Hence, achieved availability is given by following relationship [6].

Achieved availability, $\mathbf{A_a = MTBM/(MTBM + M)}$ where MTBM is mean time between maintenance and M is the mean active down time resulting from both preventive and corrective maintenance. If preventive maintenance and corrective maintenance are ignored, then MTBM becomes MTBF. The achieved availability is usually less than inherent availability of the system.

Illustration 3:

Let us assume a system with MTBM value of 8 execution hours (CPU Hrs) and M value equivalent to 4 execution hours (CPU Hrs), then

$$\text{Achieved availability is } 8/(8+4) = 0.66$$

The operational availability considers supply down time and administrative downtime which is given as follows [6].

Operational availability, $\mathbf{A_o = MTBM/(MTBM + MDT)}$ where MTBM is mean time between maintenance and MDT is supply downtime and administrative downtime. Hence, operational availability is usually less than inherent availability and achieved availability.

Illustration 4:

Consider the previous illustration with same value for MTBM. But by assuming a MDTvalue equivalent to 6 execution hours, then

$$\text{Operational availability} = 8/(8 + 6) = 0.5714$$

The system availability (A_s) of a software system which comprises both software and hardware components is usually expressed as a complex function of reliability (R_s), maintainability (M_s) and supply effectiveness (S_s),

System Availability, $A_s = f(R_s, M_s, S_s)$

Hence, system availability is a function of tradeoff between reliability and maintainability of the system with stated value of supply effectiveness. As far as the system is functioning properly without any failure, maintainability will be low and all performance related issues and performance improvement may be worked out according to a stated plan. However, for a failed system in terms of functions and performance which is under maintenance, the maintainability issues of the system should be taken into consideration along with changed reliability to work out availability of the system. Hence, performance improvement and management should be addressed with altered perspective.

The other attributes of dependability are safety, confidentiality and integrity. The absence of serious consequences to the environment is safety. The non-occurrence of unauthorized disclosure of information is called confidentiality and absence of alteration of information is called integrity [1]. These three attributes are very important to place highest trust on the functioning of software. The functioning of the software and its improvement in performance will not be useful unless the safety, confidentiality and integrity is achieved for the software.

Hence, it is very essential to ensure dependability (trustworthiness) of the software before launching on any performance enhancement and management program. The software should be built from requirements stage to installation stage taking all six important dependability attributes into account. Ignoring any of these attributes will cost the organization to pay the customer in terms of penalties and other types of compensation.

A tradeoff may be worked out with objective reliability and other attributes like maintain ability and availability to cater to application as far as performance parameters are not sacrificed. Since reliability is most important attribute which directly deals with functioning of software and the user is more interested to have higher reliability for any given software, it is advisable to have most of the performance improvement and management initiatives geared towards ensuring higher reliability and optimal performance of the software.

The organization should have comprehensive plan to address dependability and performance related issues on a broader perspective. It should not be limited to Software Quality Assurance (SQA) group or Software Engineering Process Group (SEPG). All other important groups which are directly or indirectly related to dependability and performance of the software should be involved to strive for developing highly dependable, and robust soft ware in terms of performance. Hence, proper personnel and resources should be allocated throughout the software development cycle (SDLC) to ensure higher dependability and performance of the software.

4. Software Performance and Its Improvement

Software performance is an indicator of how well a software system or component meets its requirements for timeliness. This is measured in terms of response time and throughput. The response time is the time required to respond to a request. It may be the time required for single transaction or end to end time for user task. In embedded systems, it is the time required to respond to the events or number of events processed in a time interval. Throughput of a system is number of requests that can be processed in some specified time interval [8,9].

When failure intensity function increases indicating presence of more faults to be removed to enhance reliability of the software, response time of software increases and throughput decreases. As indicated in earlier two software reliability models, either time between failures should be extended in case of TBF models or fault count in a time interval has to be reduced to achieve higher reliability which may contribute to higher performance with improved response time and throughputs.

Responsiveness is the ability of a system to meet its objectives for response time or throughput. Scalability is the ability of the system to meet its response time or throughput objectives as demand for new software function increases [8,9].

Illustration 5:

From Illustration 1, the number of faults is 1.8 per execution hour (CPU hr). As the time advances, more faults are uncovered and with improved testing efficiency, number of faults uncovered usually decreases. It may not be the case always.

If we assume that second set of faults are detected during 3rd execution hour of the software, then we can take MTBF as 2 execution hours (CPU Hr) between first and third hour of execution.

MTBF indicates failure free operation of software.

The performance parameter response time is indirectly related to MTBF.

If R indicates the response time of a software for a task, then R = K/(MTBF) where K is a constant. As MTBF increases response time decreases. In our illustration, R = K/2 (assuming a linear relationship between R and MTBF which may not be true always).

The second performance parameter, throughput in a time interval is directly dependent on MTBF. As MTBF increases, number of transactions in the time interval (MTBF) increases. In our illustration MTBF is 2 execution hours. If 10 transactions occur in 2 execution hours, then it would be 20 transactions if MTBF is improved to 4 execution hours. In this case, through put is constant. Even the number of transactions in a time interval (improved MTBF) can be increased to improve throughput of the software.

Hence, reliability parameter MTBF has impact on performance parameters response time and throughput.

These facts may be corroborated with failure data taken from Reference [2] which gives number of illustrations to calculate MTBF and other reliability parameters using specific software reliability model.

The reference [9] gives illustrations to calculate performance parameters response time and throughput with the support of data. The guiding principle for good response time and throughput is failure free operation of software (higher reliability with higher MTBF values).

Hence, it is just the logical extension to establish the relationship between reliability parameter MTBF and performance parameters response time & throughput since both of these parameters are independently supported by data and illustration in references [2,8].

It may look exploratory in nature as far as exact relationship between these parameters. Nevertheless, it holds true for simple & general applications. The exact relationship can be worked out in a new or subsequent research paper as future work.

The higher reliability of the software ensures higher scalability for a new function without much difficulty as far as other performance parameters are addressed in tandem.

The following are the consequences of performance failures—damaged customer relations, business failures, additional resources and reduced competitiveness [9]. The causes which hinder the optimum performance of software may be due to internal and external conditions.

The internal conditions may be defective computer system with presence of faults in the system (both hardware and software). The external conditions may be environment in which computer system operates. Hardware faults may be due to imperfect material or manufacturing process to build computer system. But most of the software faults are design fault which are difficult to identify and rectify.

5. Software Performance Management

The software performance has to be managed to deliver optimum performance. There are basically two different methods to manage software performance [10].

5.1. Reactive Performance Management

Reactive performance management focuses on actions or remedies that will be taken, once the software performance problem is encountered. It may be for any of the reasons like not meeting optimal performance requirements with presence of faults or malfunctioning of some critical components or non-functioning due to exhausted resources or system overloading. Reactive performance management has the risk of cost overrun, delayed project and poor product/service delivery [9].

5.2. Proactive Performance Management (with Inputs from Software Reliability Engineering)

It anticipates potential software performance problems and steps to detect and removal of those problems early in process. The characteristics of performance management are as follows [1,10].

1) Quantify product usage by specifying reliability and performance level so that failure free functioning and optimum performance is possible.

2) A performance engineer does track and communicate the issues related to performance and everyone is informed about his role in performance management. He should closely work with Reliability Engineer to check the product trustworthiness (Reliability, availability and maintainability). Performance is not possible without reliable product or service.

3) An organization wide standard business processes and best practices in software reliability engineering (SRE) should be established to meet any unforeseen outcome related to reliability and performance problems.

4) Analyze, manage and improve the reliability of the software predicted by software reliability models and match the outcome reliability with reliability assured to the customers. Improvement in reliability with presence of very few faults will improve the performance of the software.

5) Team members should be trained in processes related to Software performance engineering (SPE) and software reliability engineering (SRE).

6) A performance risk management plan will be put in place to provide contingencies for performance risk.

7) The important SPE strategies like simple modeling strategy, Best and worst case strategy and adoption to precision strategy for modeling with suitable estimates should be chosen. Failure behavior of the software predicted by software reliability models along with resource requirements needed for the performance of the software are used to formulate these estimates.

8) The software performance models involve construction of software execution model followed by building the system execution models by studying and measuring execution patterns of computer systems. In addition, workloads are characterized and input parameters are developed. The developed model is validated by comparing model results with observed and measured data for the computer system. The model is calibrated until its results match measured data. [10]

9) The software reliability models can be used in conjunction with SPE models, while designing software execution models and subsequent system execution models. The execution patterns of software will reveal failure behavior and reliability of the software. The software performance engineer can work on these issues before characterizing work loads and developing inputs for the model. The workloads may be varied and inputs may be changed or altered according to the behavior of the software. If a higher reliability is achieved by removing defects of software during execution, greater workload and more/selective inputs can be chosen for higher performance while carrying out SPE.

6. Conclusions and Inferences

1) An overview of concepts related to software reliability, software reliability models along with two important models are discussed to highlight their importance in analyzing the failure behavior of software.

2) Failure behavior of the software as predicted by software reliability models has important implication in understanding the performance of the software and its improvement.

3) Resources can be allocated and optimized by selecting suitable non-testing methods and also testing methods by targeting objective software reliability and optimal performance of the software.

4) The dependability of software with six attributes are discussed, highlighting the importance of each of them. But reliability and maintainability are found to be more dominating attributes among other attributes. Nonetheless, other attributes are equally important to form overall

dependability perspective.

5) The different definitions related to availability of software are discussed along with formulation and illustrations. The availability as function of reliability, maintainability and supply effectiveness are discussed. Hence, reliability emerges as an important attribute directly affecting the performance of the software.

6) Software performance management with two different approaches (Reactive and Proactive Performance Management) are discussed in detail with inputs from SRE to Proactive performance management.

7) The guidelines given in proactive performance management can be followed for good results in reliability and performance by combining best practices in both the fields.

REFERENCES

[1] M. Lyu, "Handbook of Software Reliabilty Engineering," IEEE Computer Society Press, McGraw Hill, New York, 1996.

[2] J. D. Musa, "Software Reliability Engineering," McGraw Hill Book Company, New York, 1999.

[3] H. Pham, "System Software Reliability," Springer, Berlin, 2006.

[4] Cagataycatal and Banudiri, "A Systematic Review of Software Fault Prediction Studies," *Expert Systems with Applications*, *Elsevier*, Vol. 36, No. 4, 2009, pp. 7346-7354. doi:10.1016/j.eswa.2008.10.027

[5] Cagataycatal, "Software Fault Prediction, A Literature Review and Current Trends," *Expert Systems with Applications*, *Elsevier*, Vol. 38, 2011, pp. 4626-4636.

[6] L. S. Srinath, "Reliability Engineering," 3rd Edition, Affiliated East-West Pvt. Limited, Ellis Horwood, 1998.

[7] T. Nakagawa, "Advanced Reliability Models and Maintenance Policies," Springer, Berlin, 2008.

[8] C. U. Smith, "Software Performance Engineering, What Can It Do for You?" CMG, Michelson Presentation, Washington, 2011.

[9] C. U. Smith and L. G. Williams, "Performance Solutions, A Practical Guide to Creating Responsible, Scalable Software," Addison-Wesley, New York, 2003.

[10] R. Berry, "Trends, Challenges and Opportunities for Performance Engineering with Modern Business Software," *IEE Proceedings in Software*, Vol. 150, No. 4, 2003, pp. 223-229. doi:10.1049/ip-sen:20030806

