World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:8, No:6, 2014

Software Reliability Prediction Model Analysis

L. Mirtskhulava, M. Khunjgurua, N. Lomineishvili, K. Bakuria

***Abstract*—**Software reliability prediction gives a great opportunity to measure the software failure rate at any point throughout system test. A software reliability prediction model provides with the technique for improving reliability. Software reliability is very important factor for estimating overall system reliability, which depends on the individual component reliabilities. It differs from hardware reliability in that it reflects the design perfection. Main reason of software reliability problems is high complexity of software. Various approaches can be used to improve the reliability of software. We focus on software reliability model in this article, assuming that there is a time redundancy, the value of which (the number of repeated transmission of basic blocks) *can* be an *optimization parameter*. We consider given mathematical model in the assumption that in the system may occur not only irreversible failures, but also a failure that can be taken as self-repairing failures that significantly affect the reliability and accuracy of information transfer. Main task of the given paper is to find a time distribution function (DF) of instructions sequence transmission, which consists of random number of basic blocks. We consider the system software unreliable; the time between adjacent failures has exponential distribution.

***Keywords*—**Exponential distribution, conditional mean time to failure, distribution function, mathematical model, software reliability.

I. Introduction

AN important quality attribute of a computer system is the degree to which it can be relied upon to perform its intended function. Evaluation, prediction, and improvement of this attribute have been of concern to designers and users of computers from the early days of their evolution. Until the late 1960's, attention was almost solely on the hardware related performance of the system. In the early 1970's, software also became a matter of concern, primarily due to a continuing increase in the cost of software relative to hardware, in both the development and the operational phases of the system [1]-[3].

First serious effort on software reliability started at Bell Laboratories in 1964. Nonetheless, some of the important works that appeared in the 1960s were by Haugk, Tsiang, and Zimmerman, Floyd, Hudson, Barlow and Scheuer, London, and Sauter [2].

Software is essentially an instrument for transforming a discrete set of inputs into a discrete set of outputs. It comprises of a set of coded statements whose function may be to evaluate an expression and store the results in a temporary or permanent location, decide which statement to execute next, or to perform input/output operations [1].

It is well known that fixing a fault in a program becomes increasingly expensive in later phases of software development. It is much more cost effective to fix as many faults as possible before releasing a program. Unfortunately, because it becomes harder to detect a fault as the software becomes more reliable, the cost of testing also increases [2], [3]. Moreover, as the lines of code increase, the point, testing is no longer cost-effective and the software has to be released. It has also been observed that modular testing is a testing effort required to fix a fault grows super-linearly [2]. Hence, modular testing with fewer lines of code would significantly reduce the overall effort required for testing.

Conventional approaches to software reliability growth modeling are black-box based, i.e. the software is treated as a black-box and its interactions with the external world are modeled [4], [5]. Tests are generated from the specified functional properties of the program based on its operational profile [6]-[8]. The internal structure of the program is not taken into account while generating the test cases. A stochastic model is calibrated using the failure data collected during the functional testing of the software, and this model is then used to predict the reliability of the software, and to determine when to stop testing. Thus the black-box approach relies on the assumption that the software is tested as per its operational profile.

Software reliability is affected by many factors during the life cycle of a software product, from the definition of the product to the operation and maintenance. All the activities within the software development life cycle are prone to introduce faults [9]-[11].

II. Software Reliability Metrics

- Reliability metrics are units of measure for system reliability
- System reliability is measured by counting the number of operational failures and relating these to demands made on the system at the time of failure
- A long-term measurement program is required to assess the reliability of critical systems [11]-[14].

Software reliability has been defined as the probability of a software product to insure operating without failure in a

L. Mirtskhulava is with the Department of Computer Sciences, Faculty of Exact and Natural Sciences, Ivane Javakhishvili Tbilisi State University. Tbilisi, Georgia, 0194 (Phone: +995 577 400144, e-mail: lela.mirtskgulava@tsu.ge).

M. Khunjgurua is with IT Department, United Financial Corporation JSC. Tbilisi, Georgia, 0112 (Phone: +995 555 252555, e-mail: m.khunjgurua@gmail.com)

N. Lomineishvili is with the program development division, Ministry of Economy and Sustainable Development of Georgia. Tbilisi, Georgia, 0108 (Phone: +995 558 366733, e-mail: nino.lomineisvili@gmail.com)

K. Bakuria is with the Department of Information Technologies, Georgian Technical University, Tbilisi, Georgia. 01 (Phone: +995 599 702717, e-mail: kobabak@yandex.ru)

specified environment, for a given amount of time. Based on this definition, software reliability prediction has been defined as a forecast of how reliable a software product will be in the future, based on data available so far. Software reliability tends to improve over testing and operating time, due to errors being removed. This is the reason for the models being named reliability growth models [15]-[17].

For software products, continuous availability is necessary, and reliability is an important component of this. Even so, there can be defects in software products that cause system failure. In order to avoid these situations and to decrease support expenses, companies want to deliver to the client's reliable software. Developing reliable software is one of the hardest problems of the IT industry. Pressure brought by schedule, resource limitations and unrealistic demands can negatively impact the reliability. Knowing the reliability of a delivered product is a difficult issue. After reaching the clients, the reliability is indicated by the feedback coming from them, under the form of reports, complaints, or compliments. But, by this time, it is too late to change anything: that is why companies selling software want to know ahead of time the product reliability. Reliability models try to do this.

The most important cause of defects in software is bugs, which means incorrect implementations. Even the most talented programmers produce software with defects. The software products complexity is too big, at this moment, to be handled by people. With all the progress in programming techniques, such as splitting the programs in small modules, using evolved programming languages and complex developing tools, results are still far away from perfection, and the programming productivity has not increased significantly in the past two decades.

The most unpredictable defects in software manifest only after a specific combination of values for input data or certain external events that were not predicted by the programmer. Such combinations appear with low probability during normal testing procedures, so they often make it to the operating phase. Also, new versions are built on older versions, fixing defects found and adding new functionality. Even so, the process of fixing defects often introduces new defects, because the effects of a fix have unpredictable consequences. Defect prediction deals with estimating the number of defects. Although other terms have been used to describe it, such as estimation, fault estimation, we should clarify the difference between the two notions. Defect estimation has been defined by Nayak as a process of identifying different types of defects of a software product, aiming to reach high quality. However, defect prediction helps in estimating the quality of a product before it is released [2].

III. Software Requirements

A. Functional and Nonfunctional Requirements

Functional requirements describe the functions that the software is to execute; for example, formatting some text or modulating a signal. They are sometimes known as capabilities. *Nonfunctional* requirements are the ones that act to constrain the solution. Nonfunctional requirements are sometimes known as constraints or quality requirements. They can be further classified according to whether they are performance requirements, maintainability requirements, safety requirements, reliability requirements, or one of many other types of software requirements. These topics are also discussed in the Software Quality KA [8].

B. Quantifiable Requirements

Software requirements should be stated as clearly and as unambiguously as possible, and, where appropriate, quantitatively. It is important to avoid vague and unverifiable requirements which depend for their interpretation on subjective judgment ("the software shall be reliable"; "the software shall be user-friendly"). This is particularly important for nonfunctional requirements. Two examples of quantified requirements are the following: a call center's software must increase the center's throughput by 20%; and a system shall have a probability of generating a fatal error during any hour of operation of less than 1 * 10−8. The throughput requirement is at a very high level and will need to be used to derive a number of detailed requirements. The reliability requirement will tightly constrain the system architecture.

C. System Requirements and Software Requirements

In this topic, system means "*an interacting combination of elements to accomplish a defined objective. These include hardware, software, firmware, people, information, techniques, facilities, services, and other support elements.*" as defined by the International Council on Systems Engineering (INCOSE00). *System* requirements are the requirements for the system as a whole. In a system containing software components, *software* requirements are derived from system requirements. The literature on requirements sometimes calls system requirements "user requirements." The Guide defines "user requirements" in a restricted way as the requirements of the system's customers or end-users. System requirements, by contrast, encompass user requirements, requirements of other stakeholders (such as regulatory authorities), and requirements without an identifiable human source.

Reliability is defined as the probability that a device will perform its required function under stated conditions for a specific period of time. Predicting with some degree of confidence is very dependent on correctly defining a number of parameters. For instance, choosing the distribution that matches the data is of primary importance. If a correct distribution is not chosen, the results will not be reliable. The confidence, which depends on the sample size, must be adequate to make correct decisions. Individual component failure rates must be based on a large enough population and relevant to truly reflect present day normal usages. There are empirical considerations, such as determining the slope of the

failure rate and calculating the activation energy, as well as environmental factors, such as temperature, humidity, and vibration. Lastly, there are electrical stressors such as voltage and current.

IV. Reliability Demonstration Testing

A reliability demonstration test is a test to determine if a component or system has achieved a specified level of reliability. Typically, a test plan specifies a test environment, operational profile, test duration and the number of permissible failures. The system or component is then operated according to the plan and the number of observed failures recorded. The component is rejected if the number of observed failures is above the number of permissible failures [15].

V. Model Description

The overall system reliability can be estimated from the individual component reliabilities by modelling the interactions between the components using one or more of the building blocks.

In order to estimate the reliability of software, we consider a mathematical model in the assumption that in the system may occur not only irreversible failures, but also a failure that can be taken as self-repairing failures that significantly affect the reliability and accuracy of information transfer. We focus on software reliability model in this article assuming that there is a time redundancy, the value of which (the number of repeated transmission of information) is a parameter that allows to optimize.

We introduce the following notations. Assume:

n is the number of the blocks. A sequence of instructions forms a basic block if: a) the instruction in each position dominates, or always executes before, all those in later positions, and b) no other instruction executes between two instructions in the sequence.

$\Phi_j(t)$ - the probability distribution function of time of the instruction sequences transmission, which consists of n basic blocks if transmission is started from j-th block ($j = \overline{1,n}$);

$F_j(u)$- the distribution function of transmitted block length;

r-1-the quantity of basic block transmission repetition before repair

α-error rate

$G(\upsilon)$- the distribution function of recovery.

Main task of the given paper is to find a time distribution function (DF) of instructions sequence transmission, which consists of random number of basic blocks.

When creating the given model, we used the following assumptions (A):

We consider the system software unreliable; the time between adjacent failures has exponential distribution;

Block Coverage: It is the Number of the basic blocks that have been executed by the test cases. A basic block, or simply a block, is a sequence of instructions that. The instructions in any basic block are either executed all together, or not at all.

Where the block length distribution low has the form: $F_j(u)=1(t-\tau_j)$, $j = \overline{1,n}$, where τ_j - time of transmission of j-th block;

Control of correctness of information is made after the reception of the next block and the time spent on control is considered negligible.

Under assumptions (A) in case of failure origin block transmission is repeated until receiving accurate block but no more than r-1 times. System need to be repaired after r - times repetition. The block transmission is renewed after the repair from distorted block repetition.

System software model, starting with j-th block transmission is described by the following integral equations:

$$\Phi_j^{(i)}(t) = \int_0^t dF_j(u) e^{-\alpha_j u} \Phi_{j+1}^{(1)}(t-u) + \int_0^t dF_j(u) \left[1 - e^{-\alpha_j u}\right] \Phi_j^{(i+1)}(t-u) \tag{1}$$

$$\Phi_j^{(r-1)}(t) = \int_0^t dF_j(u) e^{-\alpha_j u} \Phi_{j+1}^{(1)}(t-u) + \int_0^t dF_j(u) \left(1 - e^{-\alpha_j u}\right) \int_0^{t-u} dG(\upsilon) \Phi_1^{(1)}(t-u-\upsilon) \tag{2}$$

where $j = \overline{1,n}$

The boundary condition has the following form:

$$\Phi_{n+1}(t) = 1 \tag{3}$$

Taking the Laplace-Stieltjes transform to (1), (2) и (3), we obtain:

$$\overline{\Phi}_j^{i}(s) = \overline{f_j}\left(s+\alpha_j\right)\overline{\Phi}_{j+1}^{(1)}(s) + \left[\overline{f_j}(s) - \overline{f_j}\left(s+\alpha_j\right)\right]\overline{\Phi}_j^{(i+1)}(s); \tag{4}$$

$$\overline{\Phi}_j^{(r-1)}(s) = \overline{f_j}\left(s+\alpha_j\right)\overline{\Phi}_{j+1}^{(1)}(s) + \left[\overline{f_j}(s) - \overline{f_j}\left(s+\alpha_j\right)\right]\overline{g}(s)\overline{\Phi}_1^{(1)}(s) \tag{5}$$

$$\overline{\Phi}_{n+1}(s) = \frac{1}{s};$$

Denote:

$$\overline{f_j}(s) = \int_0^\infty e^{-st} dF_j(t)$$

$$\overline{\Phi}_j^{(i)}(s) = \int_0^\infty e^{-st} \Phi_j^{(i)}(t) dt$$

$$\overline{g}(s)=\int_0^{\infty} e^{-st}dG(t);$$
$$j=\overline{1,n};\ i=\overline{1,r-2}$$

Let $\alpha_j=\alpha_i=\alpha$; $F_j(u)=F_i(u)=F(u)$ and for j=n:

$$\overline{\Phi}_{\mathrm{n}}^{(\mathrm{r-1})}(S)=\frac{\overline{f}(s+\alpha)}{S}+\left[\overline{f}(S)-\overline{f}(s+\alpha)\right]\overline{g}(s)\overline{\Phi}_1^{(1)}(S)$$

for i=r-2:

$$\overline{\Phi}_{\mathrm{n}}^{(\mathrm{r-2})}(S)=\frac{\overline{f}(s+\alpha)}{S}+\left[\overline{f}(S)-\overline{f}(s+\alpha)\right]\overline{\Phi}_{\mathrm{n}}^{(\mathrm{r-1})}(S) \quad (6)$$

In (6) instead of $\Phi_{\mathrm{n}}^{(\mathrm{r-1})}(S)$ substituting its value, we obtain:

$$\overline{\Phi}_{\mathrm{n}}^{(\mathrm{r-2})}(S)=\frac{\overline{f}(s+\alpha)}{S}+\left[\overline{f}(S)-\overline{f}(s+\alpha)\right]\times \quad (7)$$
$$\times\left\{\frac{\overline{f}(s+\alpha)}{S}+\left[\overline{f}(S)-\overline{f}(s+\alpha)\right]\overline{g}(s)\overline{\Phi}_1^{(1)}(S)\right\};$$

For i=r-3

$$\overline{\Phi}_{\mathrm{n}}^{(\mathrm{r-3})}(S)=\frac{\overline{f}(s+\alpha)}{S}+ \quad (8)$$
$$\left[1+\overline{f}(S)-\overline{f}(s+\alpha)+\left(\overline{f}(S)-\overline{f}(s+\alpha)\right)^2\right]+$$
$$+\left[\overline{f}(S)-\overline{f}(s+\alpha)\right]^3\overline{g}(s)\overline{\Phi}_1^{(1)}(S);$$

Finally:

$$\overline{\Phi}_{\mathrm{n}}^{(\mathrm{i})}(S)=\frac{\overline{f}(s+\alpha)\left\{\left(\overline{f}(S)-\overline{f}(s+\alpha)\right)^{r-i}-1\right\}}{S\left[\overline{f}(S)-\overline{f}(s+\alpha)-1\right]}+ \quad (9)$$
$$+\left[\overline{f}(S)-\overline{f}(s+\alpha)\right]^{r-i}\overline{g}(s)\overline{\Phi}_1^{(1)}(S);$$

For j=n-1

$$\overline{\Phi}_{\mathrm{n-1}}^{(\mathrm{i})}(S)=\overline{f}(s+\alpha)\overline{\Phi}_{\mathrm{n}}^{(1)}(S)+\left[\overline{f}(S)-\overline{f}(s+\alpha)\right]\overline{\Phi}_{\mathrm{n-1}}^{(\mathrm{i+1})}(S); \quad (10)$$

$$\overline{\Phi}_{\mathrm{n-1}}^{(\mathrm{r-1})}(S)=\overline{f}(s+\alpha)\overline{\Phi}_{\mathrm{n}}^{(1)}(S)+\left[\overline{f}(S)-\overline{f}(s+\alpha)\right]g(S)\overline{\Phi}_1^{(1)}(S); \quad (11)$$

For i=r-2

$$\overline{\Phi}_{\mathrm{n-1}}^{(\mathrm{r-2})}(S)=\overline{f}(s+\alpha)\overline{\Phi}_{\mathrm{n}}^{(1)}(S)+\left[\overline{f}(S)-\overline{f}(s+\alpha)\right]\times$$
$$\times\left\{\overline{f}(s+\alpha)\overline{\Phi}_{\mathrm{n}}^{(1)}(S)+\left[\overline{f}(S)-\overline{f}(s+\alpha)\right]g(S)\overline{\Phi}_1^{(1)}(S)\right\}= \quad (12)$$
$$=\overline{f}(s+\alpha)\overline{\Phi}_{\mathrm{n}}^{(1)}(S)\left[1-\overline{f}(S)-\overline{f}(s+\alpha)\right]+$$
$$+\left[\overline{f}(S)-\overline{f}(s+\alpha)\right]^2 g(S)\overline{\Phi}_1^{(1)}(S);$$

as a result, we obtain:

$$\overline{\Phi}_{\mathrm{n-1}}^{(\mathrm{i})}(S)=\frac{\overline{f}(s+\alpha)\left\{\left(\overline{f}(S)-\overline{f}(s+\alpha)\right)^{r-i}-1\right\}}{S\left[\overline{f}(S)-\overline{f}(s+\alpha)-1\right]}\times$$
$$\times\overline{\Phi}_{\mathrm{n}}^{(1)}(S)+\left[\overline{f}(S)-\overline{f}(s+\alpha)\right]^{r-i}\overline{g}(s)\overline{\Phi}_1^{(1)}(S);$$

$$j=\overline{1,n};\ i=\overline{1,r-1}$$

For i=1

$$\overline{\Phi}_{\mathrm{j}}^{(1)}(S)=a(S)\overline{\Phi}_{j+1}^{(1)}(S)+b(S)\overline{\Phi}_{1}^{(1)}(s),... \quad (13)$$

where

$$a(S)=\frac{\overline{f}(s+\alpha)\left\{\left(\overline{f}(S)-\overline{f}(s+\alpha)\right)^{r-1}-1\right\}}{\overline{f}(S)-\overline{f}(s+\alpha)-1}; \quad (14)$$

$$b(S)=\left[\overline{f}(S)-\overline{f}(s+\alpha)\right]^{r-1}\overline{g}(s); \quad (15)$$

where

$$\mathrm{f(0)=1;\ g(0)=1;}\ \ j=\overline{1,n}$$

first-order difference equation (8) is solved by successive substitution:
for j=n

$$\overline{\Phi}_{\mathrm{n}}^{(1)}(S)=\frac{a(s)}{S}+b(s)\overline{\Phi}_1^{(1)}(S); \quad (16)$$

for j=n-1

$$\overline{\Phi}_{\mathrm{n-1}}^{(1)}(S)=\frac{a^2(s)}{S}+b(s)\left[1+a(s)\right]\overline{\Phi}_1^{(1)}(S); \quad (17)$$

under j=n-2

$$\overline{\Phi}_{\mathrm{n-2}}^{(1)}(S)=\frac{a^3(s)}{S}+b(s)\left[1+a(s)+a^2(s)\right]\overline{\Phi}_1^{(1)}(S); \quad (18)$$

$$\overline{\Phi}_{\mathrm{n-2}}^{(1)}(S)=\frac{a^{j+1}(s)}{S}+\frac{a^{j+1}(s)-1}{a(s)-1}b(s)\overline{\Phi}_1^{(1)}(S)...$$

under j=n-1, we have:

$$\overline{\Phi}_1^{(1)}(S)=\frac{a^n(s)}{S}+\frac{a^n(s)-1}{a(s)-1}b(s)\overline{\Phi}_1^{(1)}(S) \quad (19)$$

Finally, we obtain:

$$\overline{\Phi}_1^{(1)}(s)=\frac{a^n(s)\left[a(s)-1\right]}{s\left[\left(a(s)-1\right)-\left(a^n(s)-1\right)b(s)\right]} \quad (20)$$

Conditional mean time to failure:

$$-T_m=s\overline{\Phi}_1^{(1)}(s)\Big|_{s=0}^{1}=\frac{a^n(s)\left[a(s)-1\right]}{\left[\left(a(s)-1\right)-\left(a^n(s)-1\right)b(s)\right]}\Big|_{s=0}^{1} \quad (21)$$

where

$$a(0) = 1 - b(0) \;;\quad b(0) = \left[1 - \bar{f}(\alpha)\right]^{r-1}$$

After solving (12):

$$-T_m = \frac{\tau_b + \left[1 - \bar{f}(\alpha)\right]^{r-1}\left[\tau_{\hat{a}}\,\bar{f}(\alpha) - \tau_b\right]}{\bar{f}(\alpha)b(0)} \times \qquad (23)$$
$$\times \left[\frac{1}{a^n(0)} - 1\right]; r \geq 3$$

where

$$\tau_r = -g'(0) \;;\; \tau_b = -f'(0)$$

VI. Conclusion

In this paper, we have investigated a queuing-based model for software analysis, which allows investigating the problem of determination of the probability characteristic of time of the blocks performance with account of failure origin. We focused on software reliability model assuming that there is a time redundancy for recovery of the system, the value of which (the number of repeated transmission of the sequences) is a parameter that allows to optimize.

We have found a time distribution function (DF) of instructions sequence transmission, which consists of random number of basic blocks.

VII. Background and Related Work

This section provides a brief overview of relevant background and related studies in the analysis of software reliability prediction.

Dhillon in his book "Applied Reliability and Quality: Fundamentals, Methods and procedures", Chapter 7 presents some aspects of software reliability evaluation models. In this chapter two mathematical models: Mills and Musa models are presented. Firs model was developed by arguing that the faults remaining in a given software program can be estimated through a seeding process that makes an assumption of a homogeneous distribution of a representative class of faults. Thus, both seeded and unseeded faults are identified during reviews or testing and the discovery of seeded and unseeded faults permits an assessment of remaining faults for the fault type in question. Second model is based on the premise that reliability assessments in the time domain can only be based upon actual or real execution time, as opposed to elapsed or calendar time, because only during execution a software program really becomes exposed to failure-provoking stress. Some of the important assumptions pertaining to this model are as follows: Failure intervals follow a Poisson distribution and are statistically independent; failure rate is proportional to the remaining defects; execution times between failures are piecewise exponentially distributed [1]-[10].

There is a definite need for reliability and quality professionals working in diverse areas to know about each other's work activities because this may help them, directly or indirectly, to perform their tasks more effectively. At present to the best of author's knowledge, there is no book that covers both applied reliability and quality within its framework. More specifically, at present to gain knowledge of each other's specialties, these specialists must study various books, articles, or reports on each of the areas in question. This approach is time consuming and rather difficult because of the specialized nature of the material involved [1].

My previous work is devoted to study and evaluation of data transmission through unreliable wireless channel, subjected to distortions on the physical layer. The time between neighboring failures is distributed according to Erlang ratio. The method of enhance of reliability of transmission through unreliable wireless channel (WCH) is suggested [18]. My previous work describes the study of special Erlang distribution model in wireless networks and mobile computing. Demonstrates the applicability of the Erlang distribution, where queueing model is considered as wireless channel where the interarrival times between failures have the Erlang Distribution [19]. In my previous works Modeling based approach is described for analyzing and evaluating Internet server system reliability and availability in this paper. In the given model the states are defined by the different kinds of failures of the server system. The Markov model evaluates the probability of jumping from one known state into the next logical state. The probabilities between transitions of the states are a function of the failure rates of the transitional probabilities from one to another state. The number of first-order differential equations is equal the number of the states of the servers. First-order differential equations are developed by describing the probability of being in each state in terms of states of the model [20].

References

[1] B. S. Dhillon, Applied Reliability and Quality: Fundamentals, Methods and Procedures. © Springer-Verlag London Limited 2007. P. 252
[2] Schick, G.J., Wolverton, R.W., A Analysis of Competing Software Reliability Models, IEEE Trans. on Software Engineering, Vol. 4, 1978, pp. 140-145.
[3] Dhillon, B.S., Reliability in Computer System Design, Ablex Publishing, Norwood, New Jersey, 1987.
[4] Dhillon, B.S., Reliability Engineering in Systems Design and Operation, Van Nostrand Reinhold Company, New York, 1983.
[5] Dhillon, B.S., Design Reliability: Fundamentals and Applications, CRC Press, Boca Raton, Florida, 1999.
[6] Pecht, M., Editor, Product Reliability, Maintainability, and Supportability Handbook, CRC Press, Boca Raton, Florida, 1995.
[7] Musa, J.D., Iannino, A., Okumoto, K., Software Reliability, McGraw-Hill Book Company, New York, 1987.
[8] Sukert, A.N., An Investigation of Software Reliability Models, Proceedings of the Annual Reliability and Maintainability Symposium, 1977, pp. 478-484.
[9] Mills, H.D., On the Statistical Validation of Computer Programs, Report No. 72-6015, 1972. IBM Federal Systems Division, Gaithersburg, Maryland, U.S.A.
[10] Musa, J.D., A Theory of Software Reliability and Its Applications, IEEE Transactions on Software Engineering, Vol. 1, 1975, pp. 312-327.

[11] Dunn, R., Ullman, R., Quality Assurance for Computer Software, McGraw-Hill Book Company, New York, 1982.
[12] Jim Darroch., Norman McWhirter. Demonstrating Software Reliability: A White Paper. Emerson Network Power™.
[13] Jim Darroch. Demonstrating Software Reliability: A White Paper. A White Paper from Emerson Network Power™ Embedded Computing.
[14] Kline, M.B., Software and Hardware Reliability and Maintainability: What are the Differences? Proceedings of the Annual Reliability and Maintainability Symposium, 1980, pp. 179-185.
[15] Grant Ireson, W., Coombs, C.F., Moss, R.Y., Handbook of Reliability Engineering and Management, McGraw Hill Book Company, New York, 1996.
[16] Dhillon, B.S., Reliability Engineering in Systems Design and Operation, Van Nostrand Reinhold Company, New York, 1983.
[17] Dhillon, B.S., Kirmizi, F., Probabilistic Safety Analysis of Maintainable Systems, Journal of Quality in Maintenance Engineering, Vol. 9, No. 3, 2003, pp. 303-320.
[18] Lela Mirtskhulava, Mathematical Model of Prediction of Reliability of Wireless Communication Networks. Cambridge, United Kingdom 10-12 April 2013. UKSim-AMSS 15th International Conference on Computer Modeling and Simulation. IEEE transactions. pp. 677- 681.
[19] L.Mirtskhulava, Member, IAENG, G. Gugunashvili, M. Kiknadze, Modeling of Wireless Networks as Queuing System. Proceedings of the World Congress on Engineering and Computer Science 2013 Vol II WCECS 2013, 23-25 October, 2013, San Francisco, USA.
[20] Lela Mirtskhulava, Revaz Kakubava, Natela Ananiashvili and Giorgi Gugunashvili. Internet Reliability and Avaliability Analysis Using Markov Method. 2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation. Cambridge, UK. 26-28 April 2014.