What is Hampering the Performance of Software Reliability Models? A literature review

Khaled M. S. Faqih

Abstract— **This article explores the critical factors and issues that impede the performance of software reliability modeling science. The literature review indicates that software reliability models have not delivered the desirable deliverables that they are intended to realize. The current work suggests that the reasons for such performance incompetence of the software reliability modeling are attributed to eight major causes. Based upon the findings of the current study, a simple framework is proposed to provide guidelines to developing software organizations in order to improve the performance of software reliability modeling concept.**

Index terms — NHPP models, reliability modeling, software modeling performance, software quality.

I. Introduction

Computer systems are booming exponentially. Indeed, their correct functioning has become extraordinarily critical to human lives. For example, on March 31 in 1986, a Mexican airline crashed into a mountain because the software system did not correctly process the mountain position. Apparently, it is improbable to carry out many daily activities without the help of computer systems controlled by software. It is an observable fact that the size and complexity of software system has grown massively and indeed the trend will definitely continue in the future.

Computer system reliability has become an increasingly imperative standard in measuring service continuity. System performance is measured by how long it provides service without malfunctioning. Successful operation of any computer system depends on its software components. Thus, it is very important to ensure that the underlying software will operate correctly, perform its intended functions properly and fully deliver its desirable output. Nevertheless, the shear size and enormous complexity of current software have increased the unreliability of the system. This state has led largely to greater awareness of software reliability domain. However, to express the quality of the software system to the end users, some objective attributes such as reliability and availability should be measured. Software reliability is the most dynamic quality characteristic which can measure and predict the operational quality of the software system during its intended life cycle.

The most common approach to developing software reliability models is the probabilistic approach. The probabilistic model represents the failure occurrences and the fault removals as probabilistic events. There are numerous software reliability models available for use according to probabilistic assumptions. They are classified into various groups, including error seeding models, failure rate models, curve fitting models, reliability growth models, Markov structure models, and non-homogeneous Poisson process (NHPP) models . The NHPP-based models are the most important models because of their simplicity, convenience, and compatibility.

The forthcoming exploratory paper targets the contemporary state of the of the software reliability domain. Its primary focus is to explore the limitations of the models that are primarily designed to improve software reliability. The ever increasing of software complexity and size has led to the propagation of software reliability models. In fact, software are embedded everywhere, so they must be designed to operate reliably not disastrously, and their departure from user requirements must be controlled and corrected in order to prevent any harmful consequence to their environment

This article is structured as follows: Section 2 addressees NHPP-based models. Section 3 presents a literature review of the topic. Section 4 provides a

Manuscript received December 07, 2008.
Khaled M. S. Faqih is with Al al-Bayt University, Mafraq, Jordan (phone: +9622777715746; e-mail: km_faqih@aabu.edu.jo).

discussion which lists the critical issues that hamper the performance of the software reliability models. Finally, section 5 concludes the article.

II. NHPP Models

The most popular and tractable models are NHPP models. The NHPP group of models provides an analytical framework for describing the software failure-occurrence or fault-removal phenomenon during testing. These models are normally based upon different debugging scenarios, and can catch quantitatively typical reliability growth concept observed in the testing phase of software products. The NHPP software reliability type models are based primarily upon two major assumptions. First, the numbers of failures observed in disjoint time intervals are independent. Second, the mean and the variance of the number of software failures observed up to a given instant of time coincide with each other. To the best knowledge of the author, these two assumptions have never been proved accurate.

As a general class of well-developed stochastic process model in reliability engineering, NHPP models have been successfully used in studying hardware reliability problems. They are especially useful to describe failure processes which possess certain trends such as reliability growth and deterioration. Therefore, an application of NHPP models to software reliability analysis is then easily implemented. Therefore, many proposed software reliability models have been derived on the premise of NHPP concept. What will happen if such concept gets abolished?

However, there have been too many criticisms for considering software reliability models as a provider of therapy to software ills and unreliability problems. Indeed, this paper is triggered by the work presented by Cai et al. [1], whereby they cast too much doubt on the validity and appropriateness of using the non-homogeneous Poisson process (NHPP) framework for software reliability modeling. Their empirical observations and statistical hypothesis testing have tentatively proposed that software reliability behavior does not follow a non-homogeneous Poisson process in general, and does not fit the Goel–Okumoto NHPP [11] model in particular. Undeniably, such findings are in need of further explorations before any solid conclusions are drawn from them. Nevertheless, there are numerous empirical studies available for validation of NHPP modes [2– 5]. Historically, models that have based their modeling on the concept of NHPP framework have played an influential role in software reliability modeling. However, Cai et al. [1] have stressed on the fact that no controlled software experiments have been conducted to validate or invalidate the NHPP framework concept statistically. Due to the pessimism pointed out by many researchers, this exploratory study intends to categorize the factors that impede the performance of software reliability models from providing the appropriate remedy to software ills and unreliability headaches associated with software performance once it is deployed in the field.

III. Literature review

The concept of software reliability modeling has been utilized for almost over three decades. A countless number of software reliability models have been recommended, and the earliest models include the Jelinski and Moranda model [6], the Shooman model [7], the Nelson model [8], and the Littlewood–Verrall model [9]. Some of these models have recently been to some extent falsified because of the sweeping assumptions they made in their derivation and method of operation.

Schneidewind [4] formulated an error detection model that has been extensively utilized in large number of applications. The idea behind this model is that the current fault rate might be a better predictor of the future behavior than the observed rated in the distant past. Musa [10] established a model that has been considered as one of the most widely used software reliability models which use execution time rather than calendar time in its calculations. Musa's basic model assumes that the detections of failures are independent of one another, perfect debugging is assumed, and the fault correction rate is proportional to the failure occurrence rate.

Goel and Okumoto (henceforth called G-O model) [11] suggested the time dependent failure rate model, assuming that the failure intensity is proportional to the number of faults remaining in the software. For instance, G-O model presents a stochastic model for the software failure phenomenon based on a NHPP. This fundamental assumption of G-O model is somewhat crude. Yet, it is a simple model for the description of software failure process. The G-O model was transformed by Grottke and Trivedi [12] for the sake of renovation so that the model might resemble an infinite failures NHPP model, and the new version is called the truncated Goel-Okumoto model.

Most of the models mentioned above have operationally concentrated their therapy during software testing phase, where defects are identified and eliminated and therefore software reliability has the tendency to grow. Also, most of these models are based on the assumption of perfect debugging, where they assume that there is one-to-one correspondence between the failures observed and repaired. However, this hypothetical assumption of perfect debugging has never been proved to be accurate.

The concept of S-shaped model came into being in the early eighties of the last century, where Ohba and Kajiyama [13] proposed the most widely used inflection S-shaped model which was considered a novel mechanism for predicting and solving software reliability issues. Yamada et al. [14] suggested a model based on the concept of failure observation and the corresponding fault removal phenomenon, and it was recognized as an important advancement in software modeling approach. Musa and Okumoto [15] recommended both the basic execution time model and Log Poisson model respectively. This model differs from the Musa's basic model in that it reflects the view that the earlier discovered failures have a greater impact on reducing the failure intensity function than those encountered later. The model assumes that the software is operated in a similar manner as the anticipated operational usage, the detections of failures are independent of one another, the expected number of failures is a logarithmic function of time, the failure intensity decreases exponentially with the expected number of failures experienced, and there is no upper bound on the number of total failures. Yamada et al. [16] put forward a model with two types of faults in order to widen the scope of mathematical reliability models. Their modified exponential model assumed that the software contains two categories of faults namely, simple and hard. Both faults are modeled independently and consequently the fault removal process is the linear sum of the two models.

Ohba [17] proposed the hyper-exponential model to describe the fault detection process in a module structured software, assuming that a software consists of different modules. Each module has its own unique characteristics and therefore the faults uncovered in a particular module have their own peculiarities. Consequently, the fault removal rate for each module is not the same. Ohba recommended that a separate modeling for each module can be established and the total fault removal phenomenon is the linear sum of the fault removal process of all modules. Yamada and Osaki [18] suggested two classes of discrete time models. One class describes an error detection process in which the expected number of errors detected per test case is geometrically decreasing while the other class is proportional to the current error content; a modeling diversion from the norm, however.

Kapur and Garg [19] modified the G-O model by introducing the concept of imperfect debugging; it was regarded as an important new form of assumption that may hold true naturally. Kimura et al. [20] suggested an exponential S-shaped model which describes the software with two types of faults namely, simple and hard. They suggested that that the removal of simple faults can be illustrated utilizing the exponential model techniques while the removal of hard fault is illustrated using the delayed S-shaped modeling approach. Zeephongsekul et al. [21] presented a model describing the case when a primary fault introduces a secondary fault. Zeephongsekul's assumption is an important development in modeling scheme and has certainly introduced somewhat a reasonable explanation to the nature of things that may happen in software domains. Chang and Leu [22] proposed a non-Gaussian state space model to formulate an imperfect debugging phenomenon in software reliability in order to predict software failure time with imperfect debugging. This type of modeling has been found to be suitable for tracking software reliability.

The earlier software reliability growth models (SRGMs) were developed to fit an exponential reliability growth curve and they are known as exponential SRGMs [11]. In other cases, where there was a need to fit the reliability growth by an S-shaped curve, some available hardware reliability models depicting similar curve were used by Ohba [23]. Later, few SRGMs were developed taking into account causes of the S-shapedness [13,24]. As a result there are a large number of SRGMs, each being based on a particular set of assumptions that suits a specific testing environment. Satoh and Yamada [25] have explained SRGMs based on discrete analogs of a logistic equation that have exact solutions. The deliverables of this type of modeling are accurate estimates of parameters, even with small amounts of input data. However, the generated deterministic-based equations of these models have not facilitated the yielding of a distribution of the estimates.

Lately some constructive NHPP software reliability models with change-point have been recommended by Zou [26]. Shyur [27] included both imperfect debugging and change-point problem into NHPP modeling scheme, and Huang [28] incorporated both generalized logistic testing effort function and change-point parameter into software reliability modeling. Most of the previous works concentrated on the case of single change-point, however.

Pai and Hong (2006) [29] utilized a novel technique based on support vector machines with simulated annealing algorithms to predict software reliability. They concluded that their results concerning reliability prediction were more accurate than other prediction models. A technique where support vector regression blended with genetic algorithms was applied to predict software reliability [30]. The model was tested experimentally and the results obtained indicate that the proposed model significantly outperforms the existing neural-network approaches.

Finally, a modeling scheme based on the concept of Gompertz curve has been utilized commonly in Japanese software companies to estimate the number of residual faults in testing phase of software development. Ohishi et al. (2008) [31] proposed a stochastic model called the Gompertz software reliability model based on non-homogeneous Poisson processes. They assessed the performance of Gompertz software reliability model in terms of reliability assessment and failure prediction. Based on the numerical observations, authors concluded that the proposed Gompertz software growth model was rather attractive comparing with the existing growth models. Indeed, all these recent novel attempts have been made to improve the performance of software reliability models since they have been under continuous fire from people everywhere as they are unable to deliver the right solutions for achieving error-free software systems.

IV. Discussion

Pressures have been mounted on software academics to achieve an enhancement in the performance of software reliability models because many skeptics deem that the deployment of these models in software domain is a fruitless notion; even some have gone too far by affirming that reliability models as a terminology should never be part of the active vocabulary of software dictionary. There is a general consensus among software community members that it often takes a life time of therapy to remove all software ills, and sometimes even that does not work at all. This is as such because some software faults are hard to pinpoint using the best practices of testing or the instant never comes for some bugs to be triggered.

The current work recognizes that there are unresolved problematic issues encountering reliability modeling techniques which are responsible for hampering to a high degree their performance. The rationale for such issues is that the probabilistic behaviors of software are never straightforward to manipulate. This article has investigated most of the commonly used reliability models. Their assumptions and methods of operations have been thoroughly addressed. The current article suggests that the deficiency in software modeling techniques has been attributed to the following underlying eight major causes that have been observed to be accountable for performance deficiency of the software reliability models.

● **Unfounded types of assumptions**. Huge varieties of assumptions have been considered to facilitate the mathematical treatment of the reliability software domain to be able to develop a tractable model in order to achieve plausible results, many of those assumptions have been proposed without either theoretical or practical justification. However, numerous studies have shown clearly that those assumptions are not truly representative of reality. Also, one group of reliability models make the sweeping and crude assumption that the testing is conducted homogeneously and randomly, that is, the test data are chosen by some random mechanisms from the external environment and the software using these data are tested based on the assumption that conditions are homogeneous. Indeed, such sweeping assumption has not been justified. Furthermore, the widespread use of the stochastic-based assumption approaches to describe fault-detection process behavior may be invalid. Virtually, all current software reliability growth models assume that software failures occur randomly in time, an assumption that has never been experimentally tested despite being criticized by a number of authors in the field over the years. Effectively, as the reliability of the software improves with time such assumption becomes catastrophically invalid.

● **Complexity of the software**. It is obvious that most software systems are characterized by complexity of structure and shear size. This implies that it is impossible to identify their current reliability and formulate any model to judge their future reliability. Any introduction of some additional sweeping assumptions for constructing a reasonable model may harm the concept of reliability as a whole. As many models as there are and many more emerging, none of the models can capture a satisfying amount of the complexity of software; constraints and assumptions have to be made for the quantifying process. Therefore, it is not possible to see a single reliability model able to incorporate all factors which are thought to influence software reliability.

● **Complexity of the reliability models**. A mathematically complex reliability models have emerged in the literature, however, an extensive validation of these models seems to be lacking. In fact, despite the existence of diversified and numerous models, none of them can be recommended enthusiastically to potential users. It is true that mathematically-intensive expressions have been comprehensively utilized to develop reliability models that are characterized by tremendous mathematical strength; some of those models are not amenable to any type of simplifications. The shear complexity of those models has caused difficulty in implementing them in software domain for the benefit of improving software reliability.

● **Weakness of the reliability models**. It has already been said, despite the existence of many diversified reliability models none of them can be recommended enthusiastically to provide the proper therapy to software ills. Furthermore, reliability models are unable to account for the "thoroughness" with which the code may have been tested. Admittedly, most of the concepts utilized in software reliability modeling have been applied inappropriately from hardware reliability. Such unjustified migration of these concepts has put the software reliability modeling techniques in jeopardy. Furthermore, one common aspect of all existing models is that their functionalities are probabilistic-based. This aspect has been clearly considered as a poor trend in software reliability modeling methodologies. Another common characteristic of all of these models is that they uniformly treat software as a black box. Black box technique does not handle the internal structure of the software, it focuses primarily on how software deals with the external environment. Apparently, most reliability models seem to lack the needed strength to excel in eliminating errors in software environment. Such superficiality of the reliability models clearly characterizes most of the existing reliability models. Not forgetting that software reliability has been looked at by many of being to a high extent non-scientific issue which implies that any attempt to quantify is pointless.

● **The misconception of fault and failure phenomena**. The argument of most reliability prediction models is that failure rate is directly proportional to the number of faults in the program; this may be considered unrealistic to a certain degree because such assumption has never been validated either theoretically or experimentally. However, based on empirical studies, there is a degree of indication to corroborate with such assumption. The expected conclusion of this assumption is that the failure rate will be reduced. Reliability models do not critically include the solid fact that software normally has various types of faults and each one necessitates different strategies and different magnitude of testing efforts to remove it. Consequently, if such fact is ignored the models may deliver gravely misleading outcomes that hamper the reliability of the software products. Unfortunately, all models assume that faults removal do not introduce new ones. This assumption is referred to perfect debugging. Nevertheless, such broad assumption can be seen absolutely valid because sometimes fault fixing cannot be seen as a deterministic process that resulting in perfect removal of the fault. However, new faults could be introduced as a consequence of fixing one fault. Also, it has been considered without justification that each fault contributes equally to the failure rate. However, different software faults do not affect the failure probability equally. Furthermore, for some modeling techniques, failures are assumed to be independent; this aspect has never been justified as such.

● **Inaccurate modeling parameters**. Most reliability models lack enough experimental data to be used to derive accurate parameters for the reliability models before transferring them completely to the software domain. However, the parameters values never get validated to prove accurate. Most of those models use parameters which are not even justified. In reality, there are many uncertainties surrounding those parameters and they can rarely be estimated accurately. The lack of enough experimental data has been considered as a stumbling block for the success of reliability models.

● **Difficulty in selecting the reliability models**.
Many strongly believe that selection of the reliability model to match the software environment has been for long a formidable task and fraught with uncertainties. There are great variations of models available in literature. Indeed, this overabundance of reliability models causes the problem of model selection. As a result there are no universally accepted methodologies to how selection of the reliability model that corresponds correctly to the software environment undergoing reliability measurements could be done.

● **Difficulty in building software operational profile.**
Software reliability models have been hit hard by their incapability to deliver an accurate operational profile of the software once it is put into operation. Indeed, it is an upheaval task to attempt to develop an operational

profile. Admittedly, even though the science of reliability has been around for long, software reliability discipline is still struggling to establish certain methodologies and techniques for building software operational profiles. It is imperative though to see a new generation of reliability growth models that have the capacity to build an operational profile of the software product.

A proposed simple framework is suggested to improve the current software reliability modeling techniques. It is critical to have software built to facilitate, enhance, and improve the working modes of the reliability models. The proposed framework can be utilized as guidelines for anyone wishing to establish a reliability model. The guidelines are as follows:

1. The success of the framework can never materialize unless the aforementioned assumptions are heavily avoided.
2. The framework must make sure that any data populated within the system must be tested and validated before deployment into the reliability model.
3. It is imperative to estimate new parameters that based on the measurement of numerous execution time intervals between failures.
4. An appropriate framework must be able to identify all factors that feed unreliability in software design domain. It is more beneficial to pin down concealed factors that trigger unreliability than caring for reliability after software design. Thus, there must be a focus upon accurate measurement, meticulous testing, powerful error verifying techniques, effective fault detecting mechanisms, and precise means of correcting mistakes to achieve zero-error code.
5. It is qualitatively well-established fact that moving from randomness-based approach to uncertainty-based approach will successfully support the reliability platform framework to improve its performance. Therefore modeling necessitates the use of the techniques of fractal sets and chaos theory rather than probability theory.
6. Utilization of metrics to measure software reliability can be helpful for any proposed reliability framework. At each phase of the development life cycle, metrics can identify potential areas of problems that may lead to problems or errors.
7. Any suggested software reliability framework must not be based on hardware concepts. The direct application of hardware concepts to software domain is fraught with uncertainties and can be catastrophic.
8. Any framework proposed must be able to support the software products along its life cycle, including its operational life. This appears to be due primarily to widespread recognition of the benefits gained from applying the operational profile. The logic of driving automated testing with an operational profile is becoming increasingly compelling.
9. The knowledge, from the developer's perspectives, of software operational profile is a paramount determinant in facilitating the software reliability measurement. It is a customary phenomenon in software industry terrain to see the operational profile of the software that has been supposed in the early stages of the software life cycle far differs from the one that has been supposed during software development phase including the testing phase and the actual operational environment.

The suggested simple framework will prove useful since it has incorporated some of the components as seen viable by the author to resolve the inherited problems associated with existing software reliability techniques.

V. Conclusion

The purpose of the current in-depth probing study is to pinpoint the limitations of the software reliability models. Most of the articles cited in the current study primarily concerned with the mathematical formulation of the mathematical models through making sweeping assumptions that provide mathematical tractability. Those assumptions are, in most cases, unjustifiable. It seems that there is a growing acceptable trend in software reliability in making certain assumptions without justifications. In fact, the common deficiency of most reliability models is the assumptions that they make. Also, most models developed to handle software reliability problems are not tested and validated by using real data. Admittedly, it is unclear to what extent each of those models contributes to the improvement in software reliability.

The current study clearly shows that the performance deficiency associated with those models is still an outstanding issue. Therefore, one can conclusively say that silver bullet solutions to this dilemma are remote because large spectrums of studies have clearly shown that no one knows where the best solution lies.

Finally, the present work suggests that either the contemporary methodologies that handle the reliability concept in application to software domain are immature, or the software reliability models and their mighty mathematical strength have been introduced somehow to a harsh environment (the software environment), which is not even amenable to any type of mathematical analysis.

REFERENCES

[1] K. Y. Cai, D. B. Hu, C. G. Bai, H. Hu, T. Jing, “Does software reliability growth behavior follow a non-homogeneous Poisson process”, *Information and Software Technology*, 50,2007, pp. 1232–1247.

[2] T. Nara, M. Nakata, A. Ooishi, “Software reliability growth analysis –application of NHPP models and its evaluation”, *Proc. IEEE International Symposium on Software Reliability Engineering*, 1995, pp. 251–255.

[3] A. Wood, “Predicting software reliability”, *Computer*, 1996, pp. 69–77.

[4] T. Keller, N.F. “Schneidwind, Successful application of software reliability engineering for the NASA Space Shuttle”, *Proc. IEEE International Symposium on Software Reliability Engineering (Case Studies)*, 1997, pp. 71–82.

[5] K.C. Gross, “Software reliability and system availability at Sun”, *Proc. 11th International Symposium on Software Reliability Engineering*, 2000.

[6] Z. Jelinski, and P. B. Moranda, “Software reliability research”, *Statistical computer performance evaluation (Edited by W. Freiberger)*, Academic Press, New York, 1972, pp. 465-497.

[7] M. L. Shooman, “Probabilistic models for software reliability prediction”, *Proc. the Fault-Tolerant Computing Symposium*, 1972, pp. 211–215.

[8] E. C. Nelson, “A Statistical Basis for Software Reliability Assessment”, *TRW-SS-73-03*, 1973.

[9] B. Littlewood, J. Verrall, “A Bayesian reliability growth model for computer software”, *Applied Statistics*, 22(3), 1973,pp. 332–346.

[10] J. D. Musa, “A Theory of Software Reliability and Its Applications”, *IEEE Transactions on Software Engineering*, SE-1, 1975, pp. 312-327.

[11] A. L. Goel and K. Okumoto, “Time Dependent Error Detection Rate Model for Software Reliability and other Performance Measures”, *IEEE Transactions on Reliability, R*-28(3), 1979, pp. 206-211.

[12] M. Grottke, and K. S. Trivedi, “On a Method for Mending Time to Failure Distributions”, *Proc. International Conference on Dependable System and Network, Los Alamitos*, 2005, pp. 560-569.

[13] M. Ohba and M. Kajiyama, “Inflection S-shaped Software Reliability Growth Model”, IPS, Japan, *Proceedings WGSE Meeting*, Vol. 28, 1983.

[14] S. Yamada, M. Ohba, and S. Osaki, “S-shaped Reliability Growth Modeling for Software Error Detection”, *IEEE Transactions on Reliability*, R-32, 1983, pp. 475-478.

[15] J. D. Musa and Okumoto, “A Logarithmic Poisson Execution Time Model For Software Reliability Measurements”, *Proceedings of the* 7th *International Conference on Software Engineering*, Orlando, F1, 1984, pp. 230-237.

[16] S. Yamada, S. Osaki, and H. Narihisa, “Software Reliability Growth Models with Two Types of Errors”, *Recherche Operationnelle/Operations Research (RAIRO)*, 19, 1985, pp. 87-104.

[17] M. Ohba, “Software Reliability Analysis Models”, *IBM Journal of Research and Development*, 28, 1984, pp. 428-443.

[18] S. Yamada and S. Osaki, “Discrete Software Reliability Growth Models”, *Applied Stochastic Models and Data Analysis*, Vol. 1, 1985, pp. 65-77.

[19] P. K. Kapur and R. B. Garg, “Optimal Software Release Policies for Software Reliability Growth Model under Imperfect Debugging”, *Recherche Operationnelle / Operations Research (RAIRO)*, 24, 1990, pp. 295-305.

[20] M. Kimura, S. Yamada, and S. Osaki, “Software Reliability Assessment for an Exponential S-shaped Reliability Growth Phenomenon”, *Computers and Mathematics with Applications*, 24, 1992, pp. 71-78.

[21] P. Zeephongsekul, G. Xia, and S. Kumar, “A Software Reliability Growth Model Primary Errors Generating Secondary Errors under Imperfect Debugging”, *IEEE Transactions on Reliability*,R-43(3),1994, pp. 408-413.

[22] Chang, Y. C. and Leu, L. Y., “A state space model for software reliability”, *Ann . of the Inst. Stat. Math.*, Vol. 50, 1998, pp. 789-799.

[23] M. Ohba, “Inflection S-shaped Software Reliability Growth Model”, In: Osaki S. and Hotoyama Y. (Eds.), *Lecture Notes in Economics and Mathematical Systems*, Springer-Verlag, 1984.

[24] S. Yamada, Y. Tamura, and M. Kimura, “A Software Reliability Growth Model for a Distributed Developm-ent Environment”, *Electronics and Communications in Japan*, Part 3, 83(12), 2000, pp. 1446-1553.

[25] D. Satoh and S. Yamada, “Parameter Estimation of Discrete Logistic Curve Models for Software Reliability Assessment”, *Japan J. of Industrial and Applied Mathematics*, 19-1**,** 2002, pp. 39–53.

[26] F. Z. Zou, “A change-point perspective on the software failure process”, *Software Testing, Verification and Reliability*, 13,2003, pp. 85–93.

[27] H. J. Shyur, “A stochastic software reliability model with imperfect debugging and change-point”, *Journal of Systems and Software*, 66, 2003, pp. 135–141.

[28] C. Y. Huang, “Performance analysis of software reliability growth models with test-effort and change-point”, *Journal of Systems and Software*, 76, 2005, pp. 181–194.

[29] P. Pai and W. Hong, “Software reliability forecasting by support vector machines with simulated annealing algorithms”, *The Journal of Systems and Software*, 79, 2006, 747–755.

[30] K. Chen, “Forecasting systems reliability based on support vector regression with genetic algorithms”, *Reliability Engineering and System Safety*, 92, 2007 , pp. 423–432.

[31] K. Ohishi, H. Okamura, T. Dohi, “Gompertz Software Reliability Model: Estimation Algorithm and Empirical Validation”, *The Journal of Systems and Software*, 2008, doi: 10.1016/j.jss.2008.11.840.