The Performance of Software Reliability Models: A View Point

KHALED M. S. FAQIH*
Dep. of Information Systems, Al al-Bayt University, Mafraq, Jordan University, JORDAN
(Received on July 24, 2012, revised on February 08, 2013)

Abstract: The primary intention of the present study is to investigate the critical factors and issues that are accountable for poor performance of software reliability models. Critically reviewing the literature indicates that software reliability models have not delivered fully the desirable outcomes that they are intended to achieve. The findings of the current study reveal that the reasons for performance incompetence of the software reliability models in effectively improving the reliability of the software are attributed to 14 major flaws. The implications of these flaws on the performance of software reliability models have been explored.

Keywords: Software reliability models, performance, software engineering, flaws.

1. Introduction

Computer software systems are flourishing exponentially and software systems have a huge bearing on many aspects of modern day living. It is well recognized that software fails unexpectedly. Therefore, reliability of software has become the most important objectives for software engineering science. Beyond doubt, software 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. Thus, it is very important to ensure that the underlying software will operate correctly, perform its intended functions properly and fully deliver its desirable deliverables. However, the sheer size and massive complexity of current software have increased the unreliability of the system [1]. Indisputably, the software reliability discipline is still struggling to establish a software reliability estimation and prediction framework. Worse, information technology projects, in general, have certain attributes that make them prone to failure. Some of these issues that characterize IT projects are still unknown and may be hard to pin down [2].

Software reliability models are mathematically-based tools used to analyze software in order to assess the reliability of software and provide in-depth view of software working and functionalities and identify hidden faults. The most widely used approach to developing software reliability models is the probabilistic approach [3]. 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. In reality, the ever increasing of software complexity and size has led to the propagation of software reliability models. As a result, a proliferation of software reliability models has emerged as people try to understand the characteristics of how and why software fails, and try to quantify software reliability. Without a doubt, more than 100 different models have been proposed during the past 40 years for estimating software reliability but many of software practitioners, developers, and users do not know how these models can be effectively operated. Having a plethora of software reliability models in action today does not imply that they can deliver the desirable outcomes. Unfortunately, in spite of extravagant claims for reliability...
model efficacy, it must be acknowledged that no single model has emerged that can be universally recommended to provide the silver bullet solution for appropriately assessing software reliability for all situations [4]. Worse, we are not even in a position to be able to decide a priori which of the many models is most suitable in a particular context [5].

This study throws the light on software reliability models, its objective is to identify the factors and issues that impede the performance of the software reliability models in improving the reliability of software product. Software reliability models are classified into six groups, including error seeding models, failure rate models, curve fitting models, reliability growth models, Markov structure models, non-homogeneous Poisson process (NHPP) models, and Markov structure models. Of which, the NHPP-based models are the most important models because of their simplicity, convenience, compatibility, and tractability. Each group has its technique to estimate software reliability. The remainder of this paper is organized as follows. Section 2 presents a literature review of the topic. Section 3 provides a discussion which lists the critical factors and issues that impede the performance of the software reliability models. Finally, section 4 concludes this work.

2. Literature Review

The software reliability modeling technique has been used for almost over four decades. A myriad number of software reliability models have been recommended [6], and the earliest models include the Jelinski and Moranda model [7], the Shooman model [8], the Nelson model [9], and the Littlewood–Verrall model [10]. 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.

Keller and Sneidewind [11] 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 [12] 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. Goel and Okumoto (henceforth called G-O model) [13] 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 S-shaped model came into being in the early eighties of the last century, where Ohba and Kajiyama [14] proposed the most widely used inflection S-shaped model. Yamada et al. [15] 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 [16] 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. Yamada et al. [17] 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 [18] proposed the hyper-exponential model to describe the fault detection process in 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. So 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 [19] 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.

Kapur and Garg [20] suggested some modifications to 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. [21] 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. [22] 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 [23] 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.

An important class of software reliability models called software reliability growth models (abbreviated as SRGMs). The software reliability growth model is known as one of the useful mathematical tool for quantitative assessment of 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 [13]. 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 [24]. Later, few SRGMs were developed taking into account causes of the S-shapedness [14,25]. 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 [26] 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, one of the most serious limitations is the expected total number of inherent software faults calculated by the software reliability growth models that are highly sensitive to time-to-failure data [27]. Moreover, even though it is generally known that the software reliability growth models cannot be applied to safety-critical software due to the extremely long time required for the reliability to grow to acceptable levels.

A work reported by Pai and Hong [28] that a novel technique based on support vector machines with simulated annealing algorithms can be utilized to predict software reliability more accurately. 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 [20]. The model was tested experimentally and the results obtained indicate that the proposed model significantly outperforms the existing popular neural-network approaches.

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. [29] 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.

3. Discussion

Many software reliability models have been proposed in recent years to assess the reliability of software, but no good quantitative methods have been developed to represent software reliability modeling without excessive limitations. Therefore, 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.

This work recognizes that there are unresolved problematic issues encountering software reliability modeling techniques which are accountable for impeding their performance. The rationale for that is the probabilistic behaviors of software are never straightforward to manipulate. This study has thoroughly explored most of the frequently used software reliability models. Their assumptions and methods of operations have been methodically investigated. The findings of this study suggest that the performance deficiency of the software reliability modeling techniques has been attributed to the following underlying 14 major flaws that have been observed to be responsible for the poor performance of the software reliability models.

● **Complexity of Software**: It is clear that most software systems are characterized by complexity. Software complexity is the number one cause of unreliability in computation today. Recently, the size scale of software systems and their technical complexity have become much more complicated and apparently the trend will continue to grow in the future. This mammoth growth in size and complexity poses a great challenge to software reliability engineering and has become a major concern for practitioners and academics alike because complexity is a common source of error in software. Therefore, it is impossible to identify accurately current software reliability and formulate any model to judge their future reliability. As many models as there are and many more emerging, none of the models can capture a satisfying amount of the complexity of software [30]; constraints and assumptions have to be made for the quantifying process. In conclusion, Widheden [31] stated that software complexity is too abstract, multifaceted and not sufficiently well defined as to allow it to become measured objectively.

● **Complexity of Software Operating Environment**: It is well documented that software reliability is extremely sensitive to the surrounding environment [32]. Software operating environment are extremely complex, diverse and nondeterministic [33]. Certainly, the software reliability models are not normally equipped with the real data (such as user inputs) related to the operating environment in which the software application is running. As a result, software systems interact with unpredictable and highly dynamic
environment. Various reliability assessment methods for software products in the operational phase have been recommended. Though, none have adequately modeled the software operational environments without conflicts. These problematical issues of the operating environment complicate the efficient operation of the software reliability models in improving the reliability of the software. Even worse, the complexity of software operating environment has recently grown dramatically. Definitely, this phenomenon will remain a major source of noise for software reliability modeling techniques for years to come.

● **Difficulty in Building Software Operational Profile:** The operational profile is a quantitative characterization of how a system will be used in the field by customers [1], the concept of operational profiles was introduced by Musa et al. [34]. It is a fundamental concept which must be understood in order to apply software reliability engineering effectively and with any degree of validity. It is well known that software reliability growth models are sensitive to errors in operational profile [1]. Software reliability engineering techniques have been hit hard by their incapability to deliver an accurate operational profile of the software once it is put into operation. Definitely, it is an upheaval task to attempt to develop an operational profile [1] because it requires anticipating the field usage of the software and a priori knowledge about the application and system environments. Singh et al. [35] have suggested that it is impossible to identify accurate operational profile but by considering various hybrid sources of information, more near operational profile can be constructed. 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 software reliability models that have the capacity to suggest an accurate operational profile for the software product.

● **Complexity of Software Testing Process:** Software reliability models are applicable mathematical tools used to analyze software in order to assess its reliability during testing. Certainly, software testing has long been one of major paradigms for improving and estimating software reliability. Therefore, the improvement of software reliability depends predominantly upon the performance of software testing schemes. However, software testing is a highly complex, extremely labor-intensive, exuberantly time-consuming, devilishly error-prone, exceedingly resource-hungry, largely ad hoc, intellectually challenging, and managerially and technically demanding process [1,36, 37, 38,39,40,41].

Technically, a complete testing of a moderately complex software module is infeasible. Also, testing can only show the presence of errors and it gives no idea about defects still uncovered. Even advanced testing techniques produce inconclusive outcomes with respect to identifying hidden faults. Even worse, the quantitative relationships between software testing processes and the delivered software reliability are not clear. Also, it is not understandable how software testing quantitatively affects software reliability behavior and how quantitative software reliability goals can be achieved through testing [42]. Therefore, there is no way by which the actual reliability of software can be effortlessly and accurately determined through currently used current software testing methods.

● **Unfounded Types of Assumptions:** A great number of assumptions have been proposed to facilitate the theoretical treatment of the reliability software modeling in order to develop a mathematically tractable model to achieve plausible results, most of those assumptions have been proposed without either theoretical or practical justification [1,43]. However, numerous studies have shown clearly that those assumptions are not truly
representative of reality [43]. Without a doubt, such dubious and unjustified assumptions can make the reliability estimation too optimistic relative to real situations [1]. Virtually, all current reliability models assume that software failures occur randomly in time [44], an assumption that has never been experimentally tested despite being criticized by many authors over the years. Effectively, as the reliability of the software improves with time such assumption becomes catastrophically invalid. Admittedly, the space here does not allow exploring all assumptions considered in the domain of software reliability engineering.

● Complexity of Reliability Models: A mathematically complex reliability models have emerged in the literature, however, an extensive validation of these models seems to be lacking. A survey conducted in the late 1990s by the American Society for Quality reported that only 4% of the participants responded positively when asked if they could use a software reliability model [45]. Based on this survey the main disadvantage of software reliability models are their enormous complexity which makes them too difficult to be implemented by users [46]. 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. As a result, the great complexity of those models has caused difficulty in their application in software engineering domain for the benefit of improving software reliability.

● Weakness of Reliability Models: Despite the existence of many diversified reliability models, they 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. In fact, software reliability is not as well defined as hardware reliability. The unjustified migration of hardware reliability concepts to software domain has been clearly considered as a poor trend in software reliability modeling methodologies [1,47]. More, all software models’ functionalities are probabilistic-based; this inappropriate mathematical modeling technique could not help reliability models fully acquire the needed strength to operate effectively in the harsh environment of the software to improve software reliability. Another common characteristic of all of these models is that they uniformly treat software as a black box. Black-box software reliability models are too general and make too many assumptions to be applied confidently in assessing reliability [48]; these models are primarily concerned with how the software deals with external environment and do not handle the internal structure of the software. Such superficiality of the reliability models clearly characterizes most of the existing reliability models.

● Lack of Proper Reliability Software Metrics: Li et al. [49] reported that it is vital to select and use appropriate software reliability metrics in software reliability engineering. Also, the success of software reliability models relies heavily on the incorporation of proper metrics in the development process of these models. Yet, metrics continue to lie at the margins of software engineering. One key problem associated with reliability metrics is that they are not consistently validated due to the poor methods of validation and non-acceptance of metrics on scientific grounds. More, some studies have indicated that most existing software reliability metrics lack predictive power [49, 50]. Indisputably, these reasons and many others have strongly contributed to the lack of popularity of metrics in assessing the software reliability. As a consequence, there is a pressing necessity to establish a unified ontology to identify, describe, incorporate and
understand reliability-related software metrics [51], this might to some extent motivate users to adopt metrics to enhance the performance of software reliability models.

- **Complexity of Quantifying Software Reliability:** Measuring software reliability is a difficult problem to handle quantitatively [52] because we don't have a good knowledge of the nature of software. Although extensive research has recently focused on the quantification of software reliability, its quantification has been considered a tough undertaking. Even after many years of research, quantification of software reliability remains fundamentally controversial [53]. Eom [54] suggested that obtaining a precise quantitative estimate of the software reliability is nearly impossible because of the many qualitative characteristics of the software that cannot be measured directly. Tyagi and Sharma [55] reported that software reliability cannot be measured accurately and efficiently with mathematical models because reliability is a real-world phenomenon with associated real-time issues. Categorically, over 100 software reliability models have been developed, but how to quantify software reliability still remains predominantly unsolved.

- **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. However, 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. Some reliability models assume that faults removal process does not introduce new ones. This assumption is referred to 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 is unrealistic because sometimes fault fixing cannot be seen as a deterministic process that resulting in perfect removal of the fault [56]. Also, it has been considered without justification that each fault contributes equally to the failure rate. But different software faults do not affect the failure probability equally [57]. Furthermore, for some modeling techniques, failures are assumed to be independent; this aspect has never been justified.

- **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 [57]. However, the parameters values never get validated to prove accurate. Most of those models use parameters which are not even justified. There are many uncertainties surrounding those parameters and they can rarely be estimated accurately [58]. Because of the uncertainty issue, the reliability computed from the reliability models, which are functions of these parameters, is not sufficiently precise [60]. Furthermore, it has been reported that most software reliability models’ parameters are hard to estimate because they are nonlinear [59]. This lack of accurate parameters poses a big challenge in the uncertainty analysis of the software reliability modeling. Definitely, the lack of enough experimental data has been considered as a stumbling block for the success of reliability models [67].

- **Difficulty in Selecting 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 [61]. There are great variations of reliability models available in literature, but no model can be used in all cases [1], because all models are based on assumed empirical formulas that are not applicable to all situations [62]. Irrefutably, this overabundance of reliability models causes the problem of model selection. In fact, 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.

- **Lack of Industry Enthusiasm in Software Reliability Modeling**: There has been little interest in software reliability modeling in industry because reliability is seen as a cost not a value. Also, many software practitioners do not regard the reliability attribute of a software product as important as its functionality or innovation and therefore it takes very low priority throughout the development cycle of software [1, 63]. Few practitioners are willing to try out emerging techniques on software reliability engineering because its competitive advantage and cost-effectiveness are less obvious than that of other software quality attributes such as performance [63]. Thus, not too much emphasis has been paid to the adoption of software reliability modeling across various application domains [1].

- **Time and Cost Constraints**: Time and cost constraints significantly hamper the performance of software reliability models because these constraints severely limit the effort put into software reliability improvement. Though prolonged testing is desirable from a reliability point of view, it adds significant cost to the software development [36, 64], but releasing software with unacceptable reliability is also very costly. In fact, theoretical possibilities of exhaustive testing of a software are not practical under real life scenario because it requires too much time and too much cost [38]. Moreover, if a software product is released too late, the additional development cost and the risk of missing a market window could be substantial [65]. Admittedly, most software companies are frequently faced with budgetary constraints, which may limit their ability to effectively achieve an acceptable reliability level before delivering a software product [66]. However, it is hard to balance development time and budget with software reliability. The biggest challenge encountering the software industry today is to produce highly reliable software in time within an acceptable minimum cost.

4. **Conclusion**

This study is an attempt to achieve greater understanding of the critical factors and issues adversely influencing the performance of software reliability models in their pursuance of improving the reliability of software product. The argument presented in the current research evidently points to one irrefutable fact: assessing software reliability is extremely complicated and exuberantly costly, and the state of practice falls short of expectations. In practice, however, it seems customarily acceptable trend to make certain assumptions without justifications in reliability modeling domain. 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. Indeed, the sheer size and complexity of the software are the major contributing factor of software reliability problems. Beyond doubt, it is unclear to what extent each of these software reliability models contributes to the improvement in software reliability.

The findings reveal that software reliability models are extensively challenged by various problems and limitations which significantly hamper their performance under all operational contexts. The present work categorically believes that the performance deficiency associated with those models is still an outstanding issue. Therefore, it can be conclusively concluded that silver bullet solutions to this dilemma are inaccessible because large number of studies have undoubtedly shown that no one knows where the best solution lies. In real situations, it is impossible to remove all the software ills; nevertheless, by applying sound software engineering principles software reliability can be improved to a great extent.
Finally, in the light of the study findings, the current study 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 analysis.

References


Khaled M. S. Faqih received B.Sc. (Hons.) in aeronautical and astronautical engineering degree from the University of Southampton, England, in 1983. He joined the Royal Jordanian Air Force. He received a master’s degree in Information Technology from the University of Science Malaysia, Malaysia, in 2002. He has been working as a lecturer since 2002 at Al al-Bayt University, Jordan (Information Technology faculty), teaching various topics in information technology fields. He received a Ph.D. in Management Information Systems, in 2010, from the University of Banking and Financial Sciences, Jordan. His research interests, in addition to the aerodynamics of natural flight, include software engineering, web usability, web computing, online and mobile shopping adoption, and healthcare informatics.