STM JOURNALS

Journal of Artificial Intelligence Research & Advances
ISSN: 2395-6720 (Online)
Volume 4, Issue 3

Constraints and Limitationsin Software Reliability Prediction

Archana Kumar[1], Abhinav Juneja[2,*],Sapna Bajaj[3]
[1]Director, Delhi Institute of Technology and Management, Gannaur, Sonipat, Haryana, India
[2,3]Ph.D. Scholar, Al-Falah School of Engineering and Technology, Faridabad, Haryana, India

Abstract

Software, as entered into our lives in all aspects. We are now so dependent on technology that it has become very critical for our lives. All the modern technological development, be it in any field incorporates the usage of software directly or indirectly. Due to this the need for reliable software is also gaining importance. A software in deeply tested before being brought into its professional application but still it is very difficult to guarantee its reliability and own that its is 100 percent free from bugs and will not lead to failures. In this paper we have discussed various aspects of software development which are very critical for its reliability and the various limitations associated with the true prediction of the degree to reliability of the software. There is always a possibility that the Software may have latent bugs even after passing all levels of test. There has been a lot of research in this area and statisticians have given a number ofsoftware reliability growth models for reliability prediction, still there are issues while choosing the appropriate model and making a compliance with the standard assumptions of the selected models. Due to this reliability prediction accuracy is still a complex area.

Keywords: *Calender time, error, failure data, failure rate, hardware reliability, software reliability*

****Address for Correspondence***E-mail: abhinavjuneja@yahoo.com

INTRODUCTION

Computers are bringing revolutionary changes to our life with their involvement in most human-made systemsthrough sensing, communication, control, guidance and decision-making. When the requirements for anddependencies on computers increase, the crises of computer failure also increases. The impact of hardware and software failures range from issues like malfunctions of home appliances, economic loss like compromise of banking systems to life-critical like failures of flight systems and medical software. As the applicability of computer operations becomes more essential and complicated in the modem society, the need for reliability of computer software becomes more important and critical. In fact, computer software had already become the major source of reported outages in many systems. This trend has been stressed by the fact that hardware components of a system become increasingly reliable, and software starts to dominate the cause of computer system failures and outages. With the increase in demand of software driven application gadgets, its size, complexity, and criticality also increases. Today, the growth in utilization and dependency on software components is largely responsible for the high overall complexity of many system designs, since it is the integrating potential of software that has allowed designers to contemplate more ambitious systems encompassing a broader and more multidisciplinary scope[1–3].

The intention of this paper is to describe the software reliability engineering, its various implementation techniques and the limitations in prediction of reliability associated with any so called tested software. There is a lot of gap between the reliability that we can predict for our software under testing and the actual reliability encountered when the software is actually tested and operated in the actual user environment. The paper unfolds certain aspects that limit the analysts in accurate prediction of inherent faults in the software. We start our discussion with a brief distinction between software and hardware reliability. After that we appreciate the underlying

differences between fault and failure associated with software. In the proceeding sections we discuss the reliability incorporated at different stages of software development process, overview of different software reliability models based on their approach to make software reliable and then we finally discuss the underlying unreliability which still limits and constraints the accurate prediction of remaining and latent errors in the software that may or may not lead to its failure at a later stage.

SOFTWARE VERSUS HARDWARE RELIABILITY

There are some foundational differences between hardware and software failures that impact the analysisbetween hardware versus software reliability. The primary difference between the two is in the underlying mechanisms causing failures. With hardware, failures are ofteninitiated by physical processes related to stresses imposed by the operating environment. Specifically, failures are due to ageing or the components degrading with time lapse, deteriorating, or being subjected to environmental shocks. Contrastingly in software, there is nothing to "wear out". The major stimulus mechanism for failures is the remaining faults within the software which have not either been expected or unfolded. These faults may be the result of errors in coding or implementation of design or requirements specifications. However, just the presence of a fault is not enough to cause a failure. First the software must be executing, and second the input being processed during execution must be such that the fault will be encountered under just the right set of conditions that result in a failure[1].Reliability isgenerally a key driver in many system performance criteria, e.g., mission reliability and life-cycle cost[4]. Similarly, reliability and quality analysis within each stage of the design process can greatly reduce the life-cycle cost of a product.

FAILURE IN CONTRAST WITH FAULT

Maintaining a clear demarcation between failures and faults is very critical while applying software reliability. Since software reliability is integrally associated with software failures, any discussion of software reliability must start with a definition of software failure[1]. A formal definition of a software failure is a deviation of the operation of a software system from its requirement specification, while using the concept of software reliability. The notion of requirement specification is not limited to paper agreements done while preparing the software requirement specification rather may be extended to cover anything relating to the customer's satisfaction with the product to be more precise it should be a validation by the customer and not mere verification. Examples of software failures might be simply the absence of part of the output report or some format even, some software failure might result in the system being completely inoperable but recoverable by reinitializing the system software and any such condition not desired by the customer may be associated with software failures.

Conversely, a fault is a defect that has not been discovered in the software during its different phases of development. Examples of a fault might simply be an uninitialized variable,an incorrectly coded program statement, an incorrect implementation of a design or requirement specification. Software failures are an external manifestation of the presence of a fault. However, there is not necessarily a one-to-one correspondence between fault and failure. A fault may stimulate many failures if the fault is not neutralized after the first failure is encountered. Also, different faults may cause failures to occur at different rates. On the other hand, a fault may stimulate no failure at all This would be the situation if the customer uses the software product in a way that the fault is never encountered under the right conditions to cause a failure.Customers are not concerned per se with how many faults there are in a software product. They are concerned with how often the software will fail for their intended use and how costly each failure will be to them.

INCORPORATING RELIABILITY IN SOFTWARE

The software product life cycle into four phases: product definition, product design and

implementation, product validation, and product operation and maintenance [1].

Definition of Product

Proper product definition is essential for having a successful product on the market. The primary output of the definition phase is a requirement specification for the product. Reliability objectives should be included as an explicit part of the requirements specification. The first step in setting reliability objectives is to define what a failure is from the customer's perspective. Next, failures should be categorized by the impact they have on customers. A key step is understanding customer's tolerance to failures of different categories and customers willingness to pay for reduced failure rates in each failure category. The information developed in each of these steps can then be used to develop reliability objectives for the product. After such objectives are established for the product as a whole, they must be allocated among the hardware and software components within the product. In effect, a reliability budget is being established for each of the components within the product. Once reliability objectives are established for software components, the following two important items are needed to proceed .

1. An Operational Profile that determines how the Customer will use the Product (field of use for the product being developed).
2. Estimates Relating Calendar Time or Execution Time(CPU time).
3. Design and Implementation of Product

The primary purpose of the design and implementation phases is to turn a requirement specification for a product into a design specification, and then to implement the design into the product. One activity during the design phase is allocating and budgeting software reliability objectives among software components. An analysis must be conducted to ascertain whether the reliability budget can be attained within the proposed design. Another important activity should be to certify the reliability of "reused'' software (not only application software but also system software such as operating system software and communication interface software). There is a strong push to reuse software developed for one product within another product. However, the reliability of the reused software may not be sufficient to satisfy the needs of the new product. Before reusing software in a new product, its reliability should be established through reliability testing under the operating conditions intended for the new product. It is important to use the operational profile for the new product in reliability testing. These are many verification activities that should also be going on during the implementation phase. Verification activities such as inspections, unit testing, and integration testing are intended to "verify" that what was actually produced within a development stage is the same as what was intended to be produced. Verification activities, such as inspections, can be applied to such development stages as specifying requirements, design, and coding (unit implementation).The intention of verification activities is to minimize the propagation of the number of faults introduced in one development stage to another stage. At this time, there is little that can be done to estimate software reliability using measures from such product verification [5].

PRODUCT VALIDATION PHASE

The primary thrust of validation activities is to certify the product is suitable for customer use. Product validation is associated with evaluating a software product at the end of development to ensure it complies with the initial requirements for the product. Product validation activities for software products generally include system test and field trial activities. Software reliability measurements are particularly useful in this phase in conjunction with reliability testing to monitor the progress of testing and to help in making product release decisions. The sequence of activities during testing typically proceeds as follows. Failures and the corresponding execution time (from the start of testing) are recorded during testing, Statistical techniques are used to estimate the parameters of software reliability execution time model components based on the recorded failure data.

Product Operations and Maintenance

The primary thrust of the operations and maintenance phases is to transfer the product into the customers day-today operations, to

support the customer in the use of the product, and to repair or fix faults within the software that are impacting the customers use of the product. The reliability of the currently operating software should be great enough so that introducing the new software release will not reduce the reliability below a "tolerable" level for the end user of the software product. Failure data collected in the customer's operating environment can be used to verify the customer's perceived level of reliability to the measured reliability of the product.

SOFTWARE RELIABILITY ASSESSMENT MODELS

Nearly all existing models of software reliability may be categorized into four basic types[6]:

a) Failure Count Category: These models take into account number of faults or failures uncovered in specific intervals of time. Keys assumptions of these models here are that the testing intervals are independent to each other, faults uncovered during non overlapping time interval are independent of each other[6].Typical examples of these models are Shooman's exponential model, GO-NHPP model.
b) Time between failures Category: These models provide estimate of the times between failures. Major assumptions of these models include independent times between failures, equal probability of exposure for each fault, no new faults injected during correction. Typical examples of such models are Goel and Okumoto's imperfect debugging model.
c) Fault Seeding Category: This type includes models to predict number of faults in the program at initial time via seeding of external faults. Major assumptions in these models include that seeded faults are distributed randomly in the program and seeded faults have equal probability of being detected. Typical example of this category is Mill's seeding model.
d) Input Domain based category: These models predict the reliability of a program when the test cases are sampled randomly from well known operational distribution of input program. Major assumptions here are that input profile distribution is known, random testing is used, and we may partition the input domain into equivalence classes. Typical example of this category is Nelson's model.

CONSTRAINTS AND LIMITATIONS IN ACCURATE SOFTWARE RELIABILITY PREDICTION

Although software reliability is accepted as a key attribute in software quality, and is defined as the probability of failure free software operation for a specified period of time in a specified environment .The residual faults in the software system directly contribute to the failure rate, causing software unreliability. The problem of measuring software unreliability can be approached by obtaining the estimates of the residual number of faults in the software.Assessing software reliability in a testing phase of a software development process is one of the important issues to develop a highly reliable software system[5]. The number of faults that remain in the code is also an important measure for the software developer, from the point of view of planning maintenance activities[2].Most of the reliability models which have been designed so far have been designed keeping in view the assumptions that:

1. A software fault is fixed immediately upon detection, and no new faults are introduced during the debugging process. This assumption of instantaneous and perfect debugging is impractical, and should be amended in order to present more realistic testing scenarios.
2. The time lag between the detection and debugging of a fault is not explicitly accounted for in the traditional software reliability models, as it complicates the failure process significantly, making it impossible to obtain closed-form expressions for various metrics of interest. However, the estimates of the residual number of faults in the software are influenced not only by the detection process, but also by the time required to debug the detected faults.
3. Debugging process affects the number of faults remaining in the software and consequently its reliability, and makes a

direct impact on the quality of a software product.

4. The other stringent assumption is that of perfect debugging. Studies have shown that most of the faults encountered by customers are the ones that are reintroduced during debugging of the faults detected during testing. Thus imperfect debugging also affects the residual number of faults in the software, and can at times be a major cause of its unreliability, and hence customer dissatisfaction.
5. Conventional software reliability models cannot account for this difference between the detected and debugged faults[4], simulation offers a powerful, yet simple alternative to take this difference into consideration. There is a need to design such models which keep into account the reliability deterioration due to debugging process.

CONCLUSION

Software reliability prediction is a very complex process and it needs the selection of an applicable software reliability growth model (SRGM) to give a reliable prediction. A software reliability growth model may suit one particular software development but may fail in the other. The standard assumptions taken into account by software reliability growth models are also not validated in actual run time environment. There is no thumb rule to quantify a particular model that may suit well for all software under development.

REFERENCES

1. William W. Everlett Software Reliability Measurement. *IEEE Transaction on Selected Areas in Communication.* Feb 1990; 8(2):
2. Wen-Li-Wang, Mei–Hwa Chen. Heterogeneous Software reliability Modelling. *Proceedings of the 13th International Symposium on Software Reliability Engineering (ISSRE'02),IEEE,2002.*
3. Ormon SW, Cassady CR, Allen G. Greenwood Reliability Prediction Models to Support Conceptual Design. *IEEE Transactions on Reliability.* Jun 2002; 51(2):
4. Gokhale SS, Lyu MR, Trivedi KS. "Software reliability analysis incorporating fault detection and debugging activities",IEEE Transaction Reliability,55(2),281-292(2006).
5. Musa JD. Introduction to Software Reliability Engineering and Testing.*Article in IEEE*, 1997.
6. Inoue S. Member, IEEE, and Shigeru Yamada, Member, IEEE, Generalized Discrete Software Reliability Modeling with Effect of Program Size. *IEEE Transactions on Sysbernetics-Part-A: Systems and Humans*. Mar 2007; 37(2).
7. Krajcuskova Z. Software Reliability models. Deptt. Of Radio Electronics, Slovak University of Technology, Slovak Republic, *IEEE.* 2007.

Cite this Article

Archana Kumar, Abhinav Juneja, Sapna Bajaj. Constraints and Limitations in Software Reliability Prediction. *Journal of Artificial Intelligence Research & Advances*. 2017; 4(3): 45-49p.