ISSN:2319-7900

TWO APPROACHES WITH UNCERTAINTY FACTOR ANALYSIS

Arashdeep Kaur
Assistant Professor
Chandigarh

Abstract : Software Reliability is considered to be an important factor affecting system reliability. Reliability estimates are used for various purposes: during development, to make the release decision; and after the software has been taken into use, as part of system reliability estimation, as a basis of maintenance recommendations, and further improvement, or a basis of the recommendation to discontinue the use of the software. Black box and white box are the two approaches for the reliability estimation. Factors like test coverage, number of failures etc influence the reliability of software in one or the other way. In this paper, a review to proposed black box and white box reliability models is made taking into account the uncertainty factors of black box models affecting the reliability

1. Introduction

The size and complexity of software systems have increased during the past few decades. The data from industry show that the size of the software for various systems and applications has been growing exponentially for the past 40 years [1]. Because of this ever-increasing dependency, software failures can lead to serious, even fatal, consequences in safety-critical systems as well as in normal business.

Software reliability is a critical technological challenge for the 21st century; as software plays a greater role in our society, the reliability of that software becomes a key concern. Reliability, in the general engineering sense, is "the probability that a given component or system within a specified environment will operate correctly for a specified period of time."A software failure occurs when the observed behavior of a software system departs from its specified behavior. Software failures are ultimately the result of faults in a program, which are the human mistakes made during the construction of the system. Availability is measured as the probability of a software service or system being available when needed. Reliability and availability are often defined as attributes of dependability, which is "the ability to deliver service that can justifiably be trusted" [2]. For measuring and predicting system reliability, we use the following basic notions (John D. Musa and Okumoto, 1987; Laprie and Kanoun, 1996): mean time to failure (MTTF) defines the average time to the next failure; mean time to repair (MTTR) is the
Average time it takes to diagnose and correct a fault, including any reassembly and restart times;
Mean time between failures (MTTF) is simply defined as MTBF = MTTF + MTTR; the failure rate is the number of failures per unit time. Software architecture is the first asset that describes the system as a whole. Architecture defines the system structure comprising the components, their externally visible properties and their relationships among each other [3]. By analyzing the reliability and availability prior to system implementation, time and resources are significantly saved.

Reliability and availability prediction from the architectural descriptions is a challenging task for two main reasons:

- Reliability is strongly dependent on how the system will be used. Since reliability and availability are execution qualities, the impact of faults on reliability differs depending on how the system is used, i.e. how often the faulty part of the system is executed. The analysis of different ways and frequencies to execute the system is a challenge to R&A prediction, especially when the usage profiles of the system are unknown beforehand.
- The reliability of software architecture depends on the reliability of individual components, component interactions, and the execution environment. The reliability of a component depends on its internal capabilities, e.g. implementation technology, size, and complexity, information about which might be unavailable, or not yet exist, while architecting. Furthermore, components rely on other components, interactions between components, and on an execution environment, the reliability of which may be unknown. Most of these problems appear mainly due to *uncertainty* involved in reliability parameters and the factors that contribute to software reliability estimation should be identified.

Currently there exist two very broad categories for estimation of the reliability of software systems which are called white-box and black-box models. The group of white-box models consists of several kinds of models that are used to estimate the reliability of software systems, based on the knowledge of their internal structure and processes going on inside them. This knowledge may be expressed by different means, such as architecture models, test case models, etc. On the other hand, the group of black-box models encompasses much larger number of methods that treat the software as a monolithic whole, i.e. as a black-box. We define uncertainty as the deviation of the reliability estimate given by the model, from the 'true' reliability of the system. Factors influencing uncertainty include characteristics of software [4], such as program complexity, test coverage, development environment and many others, appearing during the development lifecycle.

2. Measurement Techniques: Necessary Data

Just as a reminder, the title's question is worth repeating: What data is necessary? Data should not be collected only because it can be done. This would be just wasteful. First of all, a goal should be defined properly that lead to questions that can be answered by collecting data.

Program Size

Several models use the size or complexity of a program as input. A well-known metric for measuring program size is the lines of code metric (LOC) which is deceivingly simple. One problem with LOC is the ambiguity of the operational definition. Which lines are to be counted? Surely executable lines are counted, but what about two executable statements in one line? Lines containing data declarations only? Empty lines? Comments? Obviously, this problem can and has to be handled by a clear definition of LOC.

Test Phase

Data collected during the test phase is often used to estimate the number of software faults remaining in a system which in turn often is used as input for reliability prediction. This estimation can either be done by looking at the numbers (and the rate) of faults found during testing or just by looking at the effort that was spent on testing.

Failure Data

Of course, information about observed failures can also be used for software reliability assessment. Data collected includes, e.g., date of occurrence, nature of failures, consequences, fault types, and fault location. In the case when field data is not available and testing does not yield a sufficient amount of failure data, fault injection can be applied.

3. Black Box Reliability Models

Software reliability estimation with black box Models dates back to the year 1967 when Hudson modeled program errors as a stochastic birth and death process. In the following years, a lot of models have been developed building on various stochastic Properties. Tests are generated from the specified functional properties of the program [Howden [6]], based on its operational profile [Musa [7]].

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. Additionally, in some situations, if the software being developed is the first of its kind, the operational profile may simply be unavailable. Pasquini et al. [1996] conduct a study to investigate the sensitivity of the reliability growth models to the predictions in the operational profile. As testing proceeds, it is easier for a test case to increase coverage in the earlier part of testing than in the later phases. Thus it becomes increasingly more difficult to design test cases which will execute unexercised parts of the code, and detect faults in a program. As a result, the time between failures increases as testing time increases. However, the reliability of the software will increase only if the number of residual faults in the program is reduced.

The black box approaches measure the reliability of a piece of software only based on observations from the outside. Intuitively, some software quality attributes, such as performance or reliability is compositional - the quality of a larger System seems to be derived from the quality of smaller parts and their relationship to each other. Architecture-based approaches follow this intuition by looking at the coarse grained inner structure of software to measure the reliability.

4. Architecture-Based Reliability Models (White Box)

Large software systems are often composed from smaller blocks that bundle functionality. In architecture-based reliability prediction, these blocks are named components. Without the need to refer to a special definition components are just considered as basic entities of the software architecture. The architectural reliability models allow predicting the system reliability from the software architecture (containing components and connections between them) and the component reliability data. A major advantage of architectural reliability (or performance) prediction approaches is that it is possible to predict the system reliability already early during the software design phase [14]. Failure data of the composed system is not required, as it is the case for the black box approaches. Thus predicting the reliability of an application earlier in the life cycle, taking into account the information about its architecture, and the testing and reliabilities of its components, is essential.

Goseva-Popstojanova *et. al.* [11] classifies the existing architecture–based models into three broad categories: state–based, path–based and additive. State– based models use the control graph to represent software architecture and predict reliability analytically. Path–based models calculate software reliability considering the possible execution paths of the program. The execution paths may be determined using simulation, execution [12], or algorithmically [13, 14]. Additive models assume that component reliability can be modeled by a non-homogeneous Poisson process (NHPP) which leads the system failure process to be NHPP with cumulative number of failures and failure intensity functions that are the sums of the corresponding functions for each component [15]. Additive models do not consider the architecture of the application explicitly.

Various Poisson and binomial type models

In this section we have provided with various Poisson and binomial type models. Models based on the binomial distribution are finite failure models, that is, they postulate that a finite number of failures will be experienced in infinite time. Models based on the Poisson distribution can be either finite failure or infinite failure models, depending on how they are specified. Different models were developed with different assumptions. Still there are many factors that influence these models. Factors can be any uncertainty parameters. Any of the classical models can be made Bayesian by specifying appropriate distributions for one or more of their parameters. Interestingly, most of the Bayesian models use the exponential model as a starting point (e.g., Littlewood and Verrall, 1974 [16] ; Goel, 1977 [17]; Littlewood, 1980; Jewell, 1985; Littlewood and Sofer, 1987 [18];Thompson and Chelson, 1980; Kyparisi and Singpurwalla 1984). It seems, however, that the Bayesian approach suffers from its complexity and from the difficulty in choosing appropriate distributions for the parameters. Every model has some positive and negative impediments. Depending upon the requirements and the usefulness of the model, an appropriate model is chosen. Added to this is the fact that most software engineers do not have the required statistical background to completely understand and appreciate Bayesian models. The latter is perhaps the main reason why these models have not enjoyed the same attention as the classical models. All the facts must be defined properly.

S.no	***Poisson Type***	***Binomial Type***
1	Musa (1975)	Jelinski and Moranda (1972)
2	Moranda (1975, 1979)	Schick and Wolverton (1973)
3	Schneidewind (1975)	Wagoner (1973)
4	Goel and Okumoto (1979)	Goel (1988a)
5	Yamada and co-workers (1983)	Littlewood (1981)
6	Yamada and Osaki (1984)	

Table1: Classification of various models based on two types.

5. Uncertainty factors in black box reliability models

In the group of black-box reliability models, we focus on the so called Software Reliability Growth Models (SRGMs) as they are most mature and have a wide variety of application areas. They use the observed failure information and predict future failures that reflect the growth of reliability. Software reliability is one of the important factor that is considered for each and every software. Tools such as CASRE [9], SMERFS [8] are available for analyzing SRGMs. These models depend only on the number of failures observed or time between failures. SRGMs are in use since early 1970s. Three models that represent different groups of SRGM and found more suitable for safety critical applications are discussed here. Jelinski-Moranda [20] is one of the basic models which assumes exponential failure rate.
Musa-Okumoto [22] model assumes that the software is never fault free and is recommended for safety critical applications. This assumption must be taken as nothing can be totally error free. Littlewood Verrall [21] is applicable when there are no failures during testing or when failure data are not available. Moreover this model accounts for fault introduction during error correction process. These three models are considered in this paper as they represent each family of the black box model group and are suitable for safety critical applications. Since the black-box models rely on failure data, the reliability estimate obtained depend on various factors that can bring in uncertainty. These factors can be grouped into one of the following:

- Test coverage
- Number of failures
- Time between failures

Uncertainty factor	***Measure***	***Models influenced***

ISSN:2319-7900

Test coverage	Percents	Jelinski-Moranda Musa Okumoto Littlewood-Verrall
Number of failures	Number	Jelinski-Moranda Musa Okumoto Littlewood-Verrall
Time between failures	Time duration	Jelinski-Moranda Musa Okumoto Littlewood-Verrall

Table2: Classification of models based on uncertainty factors.

First, it is not possible to be sure that the test domain completely covers the actual requirement specifications to verify the functionality of each of the sub-systems and their interfaces. For instance, if not enough test-cases are executed, some (even rarely used) branches of the application logic may remain insufficiently tested. When a finite number of errors in the software are identified and removed, the number of remaining failures encountered in subsequent time intervals is less. This dependency on the discovery of *number of failures during different time intervals* brings in significant uncertainty.

6. Conclusion

The framework presented in this paper addresses reliability of safety critical software systems from a different perspective than the classical reliability models. It identifies different factors that bring uncertainty in reliability estimation. For this purpose different black-box models that exist today are discussed here. These factors need to be analyzed properly when developing a software system. Here we have concluded that any change in these factors influence the models in one or the other way. Thus future work must be considered for the various white box reliability models taking into account different white box models.

References

[1]W.S. Humphrey, "The Future of Software Engineering: I," Watts New Column, News at SEI, vol. 4, no. 1, March, 2001

[2]Avizienis, A.,Laprie, J.C.,Randell,B.: Fundamental Concepts of Dependability. LAAS-CNRS. p. 21 (2001)

[3] Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, Reading, 452 p (1998)

[4] Zhang, X and Pham, H., An analysis of factors affecting software reliability, In Journal of Systems and Software, 50(1), 2000, pp. 43

[6]Howden, W.E. (1980), "Functional Program Testing," IEEE Transactions on Software Engineering 6, 2, 162–169. Howden, W.E. (1985), "The Theory and Practice of Functional Testing," IEEE Software 2, 5, 6–17

[7] Farr, W., Software reliability modeling survey, in: M.R. Lyu (Ed.), Handbook of Software Reliability Engineering, McGraw-Hill, New York, 1996, pp. 71–117.

[8] Fair, W., and Smith, O., Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) User's Guide, TR 84- 373, Revision 1, NSWC, December 1988.

[9] Gokhale, S., Architecture-Based Software Reliability Analysis: Overview and Limitations, In IEEE Transactions on Dependable Security Computing 4(1): 32-40 (2007).

[10] Nikora, A., Computer Aided Software Reliability Estimation User's Guide (CASRE), Version 3.0, 2002

[11] K. Goseva-Popstojanova and K. S. Trivedi. "Architecture–based approach to reliability assessment of software systems". *Performance Evaluation*, 45(2-3), June 2001

[12] S. Krishnamurthy and A. P. Mathur. "On the estimation of reliability of a software system using reliabilities of its components". In *Proc. of Eighth Intl. Symosium on Software Reliability Engineering*, pages 146–155, Albuquerque, New Mexico, November 1997.

[13] D. Hamlet, D. Woit, and D. Mason. "Theory of software reliability based on components". In *Proc. Of Intl. Conference on Software Engineering*, pages 361–370, Toronto, Canada, 2001.

[14] S. Yacoub, B. Cukic, and H. Ammar. "Scenario-based analysis of component-based software". In Proc. of Tenth Intl. Symposium on Software Reliability Engineering, Boca Raton, FL, November 1999.

[15]M. Xie and C.Wohlin. "An additive reliability model for the analysis of modular software failure data". In *Proc. Sixth Intl. Symposium on Software Reliability Engineering*, pages 188–193, Tolouse, France, October 1995

[16]B. Littlewood and J. L. Verrall, A Bayesian Reliability Model with a Stochastically Monotone Failure Rate, IEEE Transactions on Reliability **R–22**(2) (1974).

17]A.L. Goel, *Summary of Technical Progress on Bayesian Software Prediction Models*, RADC-TR-77–112, Rome Air Development Center, Rome, NY, 1977 W. S. Jewell, Bayesian Extensions to a Basic Model of Software Reliability, IEEE Transactions on Software Engineering **SE-11**(12) (1985).

[18]B. Littlewood and A. Sofer, A Bayesian Modification to the Jelinski-Moranda Software Reliability Growth Model, Journal of Software Engineering **2**, 30–41 (1987).

[20] Jelinski, Z. and Moranda, P. (1972). Software reliability research. In Statistical Computer Performance Evaluation, W. Freiberger, editor, 465–484. New York: Academic Press. IEEE Trans. Reliability, R-34, 216–218

[21] Littlewood, B : The Littlewood-Verrall model for software reliability compared with some rivals. Journal of Systems and Software 1: 251- 258 , 1980

[22] Musa, J. D. and Okumoto, K. 1984. A logarithmic poisson execution time model for software reliability measurement. In *Proceedings of the 7th international Conference on Software Engineering*, 1984. International Conference on Software Engineering. IEEE, 230-238.

[23] M. Famelis, R. Salay, and M. Chechik. Partial models: Towards modeling and reasoning with uncertainty. In 34th International Conference on Software Engineering (ICSE 2012), pages 573–583. IEEE, 2012.

[24] N. Esfahani and S. Malek. Guided exploration of the architectural solution space in the face of uncertainty. Technical report, 2011.

[25] K. Popper. The logic of scientific discovery. Routledge, 1959 exploration of architectural solution space under uncertainty. In Proceedings of the 2013 International Conference on Software Engineering, pages 43–52. IEEE Press, 2013